Ian Ma
CSE481e UrbanSim Capstone
Final Paper

# Data Synthesis

## Introduction

The UrbanSim project is a simulation, open-source software that analyzes urban development in respect to land use, transportation, environment and public policy. The software weighs in on planning of urban development by forecasting different environmental, sociological, transportation and economic models. This is of particular interest to government officials who desire to test out these strategies before committing to long-term investments.

The capstone class at UW gives an opportunity for undergraduate students to contribute to its software development. I registered for this course both because of its helpful nature to society as well as its interesting problem of simulation. I was interested in making a significant impact to important decisions in a very real way. My quarter-long project that I chose had to do with generating synthetic data for simulation. The UrbanSim software can only simulate given a set of data. Without meaningful and accurate data, the simulation would be worthless.

What is available to researchers nowadays is a slough of information from census and other related government surveys about population. But the government has chosen to leave out important data to protect privacy and encourage anonymity. Researchers are left with census data of the whole population at an aggregate level in addition to a small percent of detail. One example of this small percent detail is from the Public Use Microdata Sample which provides around 100,000 records or 5% of the population in detail. The level of detail is down to how much time a certain person takes to commute to work as well as if their house has a complete kitchen.

The challenge that researchers face is how to estimate the other 95% of the population they know relatively little about. My project deals with this challenge attempting to use statistical and mathematical models assisted by computational algorithms for this estimation. The two models I focused on were maximum entropy and iterative proportional fitting.

## Ability to design and conduct experiments

At first, the biggest lead that I had was to start implementing maximum entropy. There happened to be a python package in SciPy that had an implementation already. In the ideal situation, I would only have to figure out what input parameters the SciPy implementation needed and decipher the output. This turned out to be a non-trivial task.

The maximum entropy package is fairly new. It comes with four examples and it is surprisingly well-commented. All four examples show a machine translation application from English to French, based off of examples from Berger et al., 1996.

Given a dictionary of text, the code finds the probability distribution of possible translations with maximal entropy subject to certain linear constraints.

The first example sets up the sample space given by (1) below as well as two linear constraints, given by (2) and (3). With these inputs as parameters, the example

```
(1)  p(dans) + p(en) + p(a) + p(au cours de) + p(pendant) = 1
(2)  p(dans) + p(en) = 3/10
(3)  p(dans) + p(a)  = 1/2
```

creates a maximum entropy model and estimates the unknown probabilities.

Example two illustrates additional features with the same constraints as input. It first initializes the model with a certain distribution using Monte Carlo sampling. It then runs the fitting using various algorithms such as CG, BFGS, LBFGSB, Powell and Nelder-Mead -- each with its own strengths and weaknesses. The problem with this example is that the Monte Carlo sampling relies on a package labeled Sandbox in python. The Sandbox package is labeled as "incomplete, poorly-tested, or experimental code." Unfortunately, the example breaks trying to use it.

Example three and four demonstrate conditional probability, one using a list of tuples as input parameters and the other using a feature matrix. What I started to realize was that these examples and the implementation as a whole was focused more as a natural language processing problem. It assumes that a single unit maps onto another single unit. In the data synthesis problem for UrbanSim, there are no clear mappings from one unit to another. Even if there are, the problem starts to break down when doing multiple dimensional joint distributions.

Nevertheless, I gave the examples a try. I designed two simple scenarios of a fair and twisted coin toss. With the fair coin toss, the sample space was heads and tails. The constraints are set up as shown below. When fitted in the maximum entropy model,

```
f0 = p(heads) + p(tails) = 1
f1 = p(heads) = 0.5
f2 = p(tails) = 0.5
```

the output is as expected with the probability of head taking 0.5 and the probability of tails taking 0.5. This shows that the model was able to fit to all the linear constraints without doing anything awkward to the probabilities. This was a validating test kind of like the identity property. The twisted coin toss played with the scenario that the coin can land on head, tails or neither (say on its side). The linear constraints are set up as below.

```
f0 = p(heads) + p(tails) + p(other) = 1
f1 = p(heads) = 0.5
f2 = p(tails) = 0.5
```

Notice how there was no constraint on the probability of "other". That is, we want the model to fit this series of constraints to the best of its abilities with maximum likelihood estimates. Notice also that the linear constraints don't mathematically line up -- that is we expect heads to be 0.5, tails to be 0.5, and conclude that "other" should be 0. One neat

property of maximum entropy is that it evenly distributes the probabilities so that even when 0 probability is expected, it will still give it a very small estimate. The fitted output when run through the model gives heads and tails each to be 0.49903396414 and "other" to be 0.00193320717102.

The next step I took was to design a scenario that is more related to the data synthesis project. Given household income and the number of children, estimate the probabilities of the joint distribution. This can be pictured as a 2x2 matrix shown below.

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | | | 0.4 |
| | 1 or more | | | 0.6 |
| | | 0.8 | 0.2 | sum = 1 |

This example shows that 40% of the population have 0 children and 60% have 1 or more children. Likewise, 80% have income below $50k and 20% have income greater than or equal to $50k. The challenge in this problem is figuring out what probability goes inside the cells in the table. For example, what is the probability that a household income is less than $50k and it has 0 children? The naive way to choose the probability is to assume independencies between income and the number of children, and multiplying the marginals together. But in practice, we know that there probably are some dependencies between the two datasets and we are not able to make that assumption. Even if there are no dependencies, we can not assume that about every set of variables we will test in the future. To the best of my knowledge, we are not able to solve this using basic matrix algebra either since there is more than one solution. We can deduce five linear equations with four unknown variables but some of those equations overlap making it impossible to solve definitely. Below are two possible solutions where the table values fit to the marginal constraints.

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | .2 | .2 | 0.4 |
| | 1 or more | .6 | 0 | 0.6 |
| | | 0.8 | 0.2 | sum = 1 |

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | .3 | .1 | 0.4 |
| | 1 or more | .5 | .1 | 0.6 |
| | | 0.8 | 0.2 | sum = 1 |

The conclusion with knowing this is that we need a statistical model to give us the maximum likelihood estimate. An appropriate model to use would be maximum entropy.

Modeling after the examples, the linear constraints I found are given below, using a, b, c and d to label the table values that need to be estimated and r1, r2, c1 and c2 to denote the marginals.

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | a | b | r1 |
| | 1 or more | c | d | r2 |
| | | c1 | c2 | sum = 1 |

$$a + b + c + d = 1$$
$$a + b = r1$$
$$c + d = r2$$
$$a + c = c1$$
$$b + d = c2$$

It would seem that the implementation could figure this one out as well but it throws an exception about diverging to infinity. Until now, I still do not understand why it happens. I have used the five different fitting algorithms each failing for the same reason. I also tried converting this matrix into a conditional probabilities problem where I create a list of tuples using the marginals in the proportion that I want to test. For example, I make 2 (r1, c1) tuples out of 10 to represent that my expected outcome should be .2.

The challenge of this project turned into one of understanding maximum entropy rather than figuring out the implementation. I spent several weeks reading maximum entropy research papers. There are three general steps in the maximum entropy model. First is to figure out the linear constraints. Then estimate the parameters using Langrange dual of the entropy and its gradient. Finally, it computes a result with matrix-vector expressions. These papers were littered with terms that were unfamiliar to me. To figure all this out would be a PH.D pursuit and something that could not be done in the ten weeks that I had in the class. I decided to fall back on to a simpler model that could accomplish similar things called iterative proportional fitting (IPF).

Iterative proportional fitting is a mathematical procedure that has been around since 1940. It is now a fairly established technique to join the information from two or more datasets. IPF has been used in many areas of study. It works particularly well in geography and census-related scenarios because it provides useful estimates for individual-level attributes given data from an aggregate-level. This method actually became popular around 1991 when it allowed government officials to make better decisions regarding resource allocation.

Population data is usually updated every couple years. In the case of census data, it is updated every ten years. A lot can change between these years. Schools could be failing, highways are getting damaged and poverty may be increasing in certain areas.

Without current data, it is hard for government officials to know how to redistribute its spending. In most cases, they might even be unwilling to fund a needed program when they do not see concrete evidence to support the need. Using IPF is critical in solving these scenarios. It takes prior knowledge and scales it to current knowledge.

Mathematically speaking, IPF starts with two or more data sets that represent marginals in a matrix. The cells in the table represent the corresponding joint distribution.

Before IPF:

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | a | b | r1 |
| | 1 or more | c | d | r2 |
| | | c1 | c2 | sum = 1 |

After IPF:

| | | Household Income | | |
|---|---|---|---|---|
| | | < 50k | >= 50k | |
| # of Children | 0 | A | B | R1 |
| | 1 or more | C | D | R2 |
| | | C1 | C2 | sum = 1 |

The tables above show before and after IPF. In the "before" table, r1, r2, c1, and c2 are the marginals while a, b, c, and d represent the joint distribution of the marginals. During IPF, the cell values a, b, c, and d, will gradually be adjusted into A, B, C and D. A, B, C and D satisfy the new constraints R1, R2, C1 and C2.

The gradual adjustments are in the form of iterations on each dimension of the matrix. Each cell is first multiplied by the row sum and then divided by the new row marginal. The "row" is dependant on which iteration it is and which dimension it corresponds to. For example, the first iteration in the table above would be ( a * (a+c) / R1 ), ( b * (a+b) / R1 ), ( c * (c+d) / R2 ) and so on. The second iteration would be operations on the columns -- ( a * (a+c) / C1 ), ( b * (b+d) / C2 ) and so on. In this example, the third iteration will go back to doing operations on the row.

There are two conditions that IPF can meet before terminating the iteration process to produce the final result. The first, and more useful, condition is that the difference in all cells of the nth iteration and n-1 iteration is less than a predetermined amount. For example, if I picked 0.1 to be the convergence allowance value, then the difference of the nth iteration and n-1 iteration of each cell must be less than 0.1. Just as a ballpark measurement, the iterations it takes before it converges in a 2x2 matrix as shown above is less than ten.

After I had done the initial research for IPF, it was time to implement it in Python. Luckily, the Numpy package provides many matrix manipulation methods that helped quicken but also complicate this process. Many times, the methods were there, it was just a matter of using the right ones at the right time. Here, I outline my implementation.

I start out with known marginals and known initial cell values. Having in mind that other people will be using my class, I decided to make it easier for them to input the right parameters. What I needed was marginals in each dimension but I allowed the user to just input Numpy arrays in one dimension. As a result, part of the setting up process in my implementation takes those marginal Numpy arrays and gives each one a different dimension. This is done by calling a reshape method on the first dimension with (<length of first array>, 1, 1), and reshape on the second dimension with the parameters (1, <length of second array>, 1) and so on.

```
x_marginals = array([1, 2, 3])
y_marginals = array([20, 21, 22, 23])
z_marginals = array([50, 51])

#Each marginal is given a different dimension using reshape, resulting in:

[[[1]]
 [[2]]
 [[3]]]

[[[20]
  [21]
  [22]
  [23]]]

[[[50 51]]]
```

Here is the place where I struggled the most. What I want to do with the initial values is to divide by the marginals. In order for me to do matrix operations on it, each marginal has to be expanded appropriately to fill up the shape of the table. For example, if I were to multiply each row of the table with the first dimension marginal, I would have the matrix operation below.

```
(Initial Table) x

             #y axis
           [[[1  1  1]]  #z axis = 1
#x axis  [[2  2  2]]
           [[3  3  3]]]

           [[[1  1  1]]  #z axis = 2
            [[2  2  2]]
            [[3  3  3]]]
```

```
[[[1  1  1]]  #z axis = 3
 [[2  2  2]]
 [[3  3  3]]]
```

Expanding the marginal like this was not the biggest challenge. Many methods could do this, such as repeat() and resize(). The biggest challenge lies in how to generalize the expansion of the marginals for each dimension. As I realized using repeat() and resize(), each implementation is dependant on which dimension it is in. This quickly breaks down for even the third dimension. The way I solved it I coined "unit and layering."

This idea takes advantage of the Numpy append() method. For each marginal, take a dimension that it currently is not. Take the current marginal as the "unit" and append over the chosen dimension n times, where n is the length of the table in that dimension. Call the new marginal the "unit" and repeat this process on other dimensions. This method guarantees that it will work for any dimension. The reason why we need the "unit" step is because append will only work if the two matrices share at least n-1 dimensions. The steps are outlined visually below.

```
#original in y dimension
[[[1]]
 [[2]]
 [[3]]]

#layering 3 times on the x dimension
[[[1  1  1]]
 [[2  2  2]]
 [[3  3  3]]]

#layering 3 times on the z dimension

[[[1  1  1]]
[[2  2  2]]
[[3  3  3]]]

[[[1  1  1]]
[[2  2  2]]
[[3  3  3]]]

[[[1  1  1]]
[[2  2  2]]
[[3  3  3]]]
```

Remember that the IPF operation involves multiplying each cell by its row sum as well. The row sums of each dimension are produced using sum_over_axis() method. Then it is "unit and layered" the same way as the marginals. One extra thing that needs to happen is that sum_over_axis() gives a result that is n-1 dimensions. The missing dimension needs to be added using expand_dim() method and then layered on that dimension. For some implementation oddities, the dimensions used in sum_over_axis() are reverse of the dimensions used for reshape() and append(). Therefore, on the ith dimension in respect to reshape() and append(), sum_over_axis() should be on the n-i dimension.

This IPF implementation was tested on examples from Paul Norman's, "Putting iterative proportional fitting on researcher's desk." The resulting two-dimensional matrices matched perfectly. I could not find an example of a three-dimensional case to test against. When I ran it the first time on a three-dimensional case, it gave me very awkward results with cells that don't match the marginals. I later realized that the marginals were incorrectly set. The problem was that all the marginals needed to sum up to the same number. When done correctly, the resulting estimates made sense. In the next section, I will describe how I applied my IPF implementation to the UrbanSim data.

## Ability to Design a Computing System

What makes UrbanSim so exciting is that it deals with real data. In a sense, this project is dealing with very realistic constraints. This became very apparently to me as I finished the IPF implementation and was looking for real data to test on. I spent a significant amount of time on www.census.gov where census data is made available to the public. I focused mostly on census 2000 data.

The census 2000 data in itself was difficult to sift through. It included four summary files that each had around 300 detailed tables relating to age, sex, households, families and housing units. There, American FactFinder provides an interface to select the tables of interest. Without a database of census data to work with, I had to manually pick out rows of interest to test with my program.

Within the census 2000 data was also something called 5-Percent Public Use Microdata Sample Files (PUMS). These files contained state-level data containing records from at least 100,000 individuals. The PUMS data is downloadable in text format and the information must be parsed out using predefined delimiters. There is an accompanying technical documentation that is about 800 pages long describing everything a researcher would want to know about the PUMS data including how the individuals were surveyed to how the text file is formatted. The information that I needed about how to parse the information out and what it meant was found on the 100th page of that document.

What I needed to do was to take the PUMS data and fit it into the census data through IPF. I picked two relatively easy datasets to test first. The first dataset was the number of households with phone availability. The technical documentation describes availability as households that have the actual phone as well as service for it. The other dataset was the number of households with complete kitchens -- meaning that they

include a refrigerator, sink and oven. These were appropriate initial test datasets since they were boolean values. Households either had these in it or it didn't (0 or 1). The results of the fit are fairly interesting. Columns represent phone availability and the rows represent complete kitchen.

Original:

| | Yes | No |
|---|---|---|
| Yes | 22264 | 1657 |
| No | 0 | 1365 |

Fitted: (marginals provided by census)

| | Yes | No | |
|---|---|---|---|
| Yes | 2419965 | 14937 | 2434902 |
| No | 0 | 16173 | 16173 |
| | 2419965 | 31110 | |

There are two points to notice from this. One is that there is a cell that is originally 0 and is left 0 after the fit. One problem with IPF is that once a cell reaches 0, it gets stuck in that pit with no chance out. Any kind of multiplication or division will not affect the 0. I tried replacing the 0 with a number like 0.000000001 and it makes quite a difference. When fit, the value is somewhere in the hundreds. I would be interesting to be able to determine what kind of significance that has on the data. Another point to notice about this data is that we are unable to test its accuracy using the true values. There is no data on the true values and it would be impossible to obtain records on the whole population.

## Ability to function in multi-disciplinary teams

Though I did not work on this project with any classmates, I still worked with the most multi-disciplinary team in my career. The UrbanSim project captures many areas of knowledge including geography, statistics, urban planning and computer science. It was inspiring and motivating working with the diverse expertise of the research faculty. They were critical in helping me figure out different parts of my project including parsing census data, iterative proportional fitting and maximum entropy.

## Ability to identify, formulate and solve problems

This class was a rare and fortunate experience for me allowing me to identify, formulate and solve a significant problem relatively on my own. The assignment was to synthesize household data but there were many parts to it, from researching models, to implementing them, to figuring out what data to use it on. Many times I had to break down a problem into more manageable pieces. When approaching a task, I split it up into sub-goals so that I can progress by iterations building up confidence and a good coding foundation. Solving implementation problems stretched my thinking when it dealt with high dimensions, unfamiliar models and mystifying data.

## An understanding of professional and ethical responsibility

As shown in the history of IPF, computer science in an integral part in everyday life. Even government officials rely on software to make decisions that impact many people. Decisions regarding transportation can affect poverty which can affect schooling for better or worse. Accurate data and appropriate forecasts can affect this avalanche of effects to a large degree. As a computer scientist, it is my responsibility to provide the tools necessary to help those around me.

## Ability to communicate effectively

This class has been a great experience in the sense of communicating on a very technical level. I was learning a lot of new statistical and geographic terms that I needed to use in the correct manner while I communicated with the experts on the team. The midpoint and final presentation also gave me a chance to explain a challenging technical problem completely fresh to classmates. I tried using many analogies and pictures to enhance that experience.

## Broad education necessary to understand the impact of computer engineering solutions in global, economic, environmental and societal contexts

What I regretted most when taking this class was not knowing enough about statistics and geography. It made me realize that I needed a broader education to support my computer science background. Especially in this field, computer science can be applied to so many areas of knowledge. It is essential to know at least a little of everything so that I can work in real, practical scenarios. Besides just thinking in algorithms, I need to think about the problems that they are used in. There are many other areas that computer science can find a place in but knowing those areas take a broad education that expands outside its own field.

## Recognition of the need for, and an ability to engage in life-long learning

Much of this project was centered on going out to research about the two different models. I mostly searched Google scholar for research papers. Going through this process, I noticed that I'm benefiting from those that have recognized a need for life-long learning. To keep progressing, we must keep building off each other and continue to learn and innovate. Computer science in an area of continual change and learning and that is what makes it so exciting.

## Knowledge of contemporary issues

I think of UrbanSim and my project as sort of a data mining problem. There is an enormous amount of unorganized and undecipherable information and the challenge is to organize it in a way that is both useful and clear to everyday people. A useful data mining solution gives users the ability to choose better decisions and also empowers them to envision different forecasted scenarios. Similar solutions in the industry today are Zillow and Farecast. There is always more information to be organized and made available.