Synthetic Data Generation for Recommendation Improvements

IMA Math-to-Industry Bootcamp VI
Capstone Project for C.H. Robinson

Muharrem Baris Otus, Samanwita Samal, Lauren Snider,
Sijie Tang, Miao Zhang, Ahmed Zytoon [*]

August 2, 2021

[*]Muharrem Baris Otus:barisotus@gmail.com; Samanwita Samal:ssamal@iu.edu; Lauren Snider:lsnider@tamu.edu; Sijie Tang:stang1@uwyo.edu; Miao Zhang:mzhan33@lsu.edu; Ahmed Zytoon:amz56@pitt.edu

1 Executive Summary

When a client submits an order to C.H. Robinson, a variety of complex factors play into the final decision of C.H. Robinson to either accept or reject that order, including contractual obligations, current carrier availability, and profitability. The Recommender System designed by the company's data science team provides an initial appraisal of orders, by which certain orders are auto-accepted and remaining orders are sent to the account team for further evaluation. Since the company's responses to its order requests are vital to the well being of both the company and its employees, the performance of the Recommender System is itself of extreme importance. However, there is currently no system in place to reliably evaluate improvements to the Recommender System. The goal of our project was to address this problem by constructing a platform which generates synthetic data on which improvements to the Recommender System can then be tested and assessed. Data in this sense simply means transactional information of an order, such as the origin and final destination of an order, the weight of an order, the rate of an order, etc.

There are several advantages to testing on synthetic data rather than historical data. For example, since real data is limited by its size or the scope of its information, testing on synthetic data can help in building a more robust Recommender System which can better handle diversity in order input. Even more, the synthetic data can be designed to reflect the primary factors that go into the decision making process. Finally, testing on synthetic data de-risks any modifications to the Recommender System, reducing the chance that actual client orders are negatively affected in the improvement process.

The platform constructed by our team combines three kinds of probabilistic models in order to generate synthetic data which closely mirrors historical data of C.H. Robinson. Together these models capture the statistical structure underlying the data and recognize interactions between certain fields of order information, up to a certain level of uncertainty. Accounting for uncertainty in our models is a desirable trait since data in the real world is inherently messy. These characteristics of our generative platform enable the synthetic data it outputs to reflect true order information. To confirm that the data produced by our model does indeed mimic true order data, our team implemented an algorithm which attempted to distinguish between our synthetic data and the historical data. The results indicated that the two kinds of data were largely indistinguishable, further validating that our platform is performing as we hoped.

In summary, the data generating model designed by our team provides the necessary platform on which improvements to C.H. Robinson's Recommender System can be safely and effectively evaluated. Enhancing the current system in place will greatly benefit the company on many levels, and utilizing the synthetic data output by our model is a vital first step towards those crucial improvements.

2 Overview of Problem and Approach

Our team was tasked with building a generative parametric model of C.H. Robinson's order decision process which could be leveraged to test new recommendation algorithms for the improvement of the current Recommender System. This model should be generative in the sense that it generates synthetic data which is similar to actual historical order data of C.H. Robinson in terms of distribution and capturing the relationships between variables. The data set provided by C.H. Robinson to train our model consisted of the following: time information (week_id, weekday, lead_days), order information (order_distance, order_origin_weight, order_num_stops, order_equipment_type), location information (origin_dat_ref, dest_dat_ref), Recommender System output (color), monetary information (rate_norm, est_cost_norm), and the company's final decision on an order (CurrentCondition). Though the origin and destination information of orders had been generalized to the DAT KMA (Dial-a-Truck Key Market Area) level in the form of 3-digit zip codes, we chose to further generalize origin and destination of orders to the DAT Zip Zone level. DAT designates ten Zip Zones in the United States, labeled Z0 through Z9. When examining the order volume associated with each possible Zip Zone origin and destination pair, our analysis indicated that the highest concentration of orders (nearly 14,000) had as both origin and destination Z3, followed by orders from Z9 to Z9 and then orders from Z7 to Z7. Thus, we decided to only consider the subset of orders from Z3 to Z3 to train our first model. We later examined the second and third most common pairs, as well.

We sought to use Bayesian methods in the design of this model. Before constructing the model, we had to determine the order of generation of our variables. We chose to assume independence of all order information except rate, cost, Recommender System output, and the company's final decision on an order. However, based on our choice to focus on a specific Zip Zone origin and destination pair when generating synthetic data, one might view these independent variables as in fact dependent upon choice of Zip Zone pair. We will overlook this distinction for now. We should also note that a possible improvement of our model might be to account for correlation between any of these independent variables. However, in our assessment of these variables, we found no obvious correlation.

After generating the independent variables of our model, our next step would be to generate rate and cost from a subset of these variables. Next, rate and cost would be used to generate the Recommender System's output; and finally, the company's final decision would be generated from rate, cost, and the Recommender System's output. We will now explain in more detail how we generated all independent variables via a best fit probability distribution or multinomial distribution, rate and cost via Bayesian hierarchical models, and color followed by current condition via a Bayesian Belief Network.

3 Best Fit Method to Generate Continuous Variables

We use the best fit method to find the approximate probability distribution of the continuous independent variables of our data set, which consisted of order weight and order distance. Using the scipy.stats package, we tested over 70 probability distributions.

Algorithm 1 Best Fit Method for continuous variables

```
1: Input our original data, and plot its distribution
2: Use different distributions (more than 70) to approximate the original distribution
3: Compare the SSE of the approximating distributions
4: The distribution with the minimal SSE is the best fit distribution of the original data
```

4 Multinomial Distribution to Generate Discrete Variables

We use multinomial distributions to generate data for all discrete independent variables, which consisted of lead days and week ID.

Algorithm 2 Multinomial distribution for discrete variables

```
1: Calculate the probability distribution of the original data
2: Use multinomial.rvs() function to generate the random data
3: Calculate the probability distribution of generated data
4: End if the probability distribution of the generated data is similar to the original data's
```

5 Bayesian Hierarchical Model

Since we assumed independence between the rate and cost variables, we implemented two Bayesian hierarchical models to generate both of these variables separately using a subset of the independent variables as predictors. To determine which of the independent variables were most highly correlated with rate and cost, we used the Decision Tree Regressor of the scikit-learn package. The results indicated that the four most highly correlated independent variables associated with both cost and rate are (in order of highest correlation) "order_distance," "order_origin_weight," "lead_days," and "week_id." Our first hierarchical models used the first three of these variables as predictors, whereas our final hierarchical models used all four as predictors. We should also note that all predictor variables used in hierarchical models were first normalized by a z-score calculation (mean and standard deviation of the entire column were used). The PyMC3 package was utilized to define, train, and test all models in this section.

Most hierarchical models that we tested during this project were partially pooled models with respect to either just zip zone origin or both zip zone origin and destination. We also tested a few pooled models, which we will further describe at the end of this section. All of our models were trained by sampling from the 154,929 non-null rows of our data set using the default sampler of PyMC3, NUTS (No-U-Turn Sampler). The general framework for all of our partially pooled models is as follows:

1. **Data Layer:** The observed data "y-like" is either rate or cost. The likelihood of the observed data is modeled as a linear regression of some subset of predictor variables.
2. **Process Layer:** The linear regression "y-hat" has its coefficients indexed by either zip zone origin alone (in which case we have 10 regression lines) or by both zip zone origin and destination (in which case we have 100 regression lines). The error of our likelihood ε is always assumed to be half-normal.

3. **Prior Layer:** Every linear regression has each set of coefficients associated with a common predictor taken from a common normal (or multivariate normal) distribution. For example, all distance coefficients are pulled from a common normal distribution; this *partial pooling* effect is desirable as we expect commonalities across zip zones.

4. **Hyperprior Layer:** The normal (or multivariate normal) distributions from which common coefficients are drawn each have their mean taken from a separate normal distribution and their standard deviation taken from a separate exponential distribution.

We will now describe our first hierarchical model and our final hierarchical model, though it should be understood that when we speak of a singular model, we are in fact describing two nearly identical models for both rate and cost, the only difference being the observed data. Perhaps there is a way to formulate a single model which predicts both rate and cost; even more, integrating covariance between rate and cost might be desirable, though this was not explored.

Our first partially pooled model included the distance, weight, and lead days variables as predictors. The likelihood was assigned a normal distribution and coefficients in the linear regression "y-hat" were indexed by just zip zone origin. The run-time of this model was under 10 minutes with no divergences. An immediate weakness of this model is evident when comparing the posterior distribution for either rate or cost, which is normal by this model, with the true distribution, which is right-skewed and more highly concentrated about the mean.

For our final partially pooled model, we added time information (week_id) as a predictor, jointly indexed coefficents of the linear regression by zip zone origin and zip zone destination, incorporated a covariance matrix, and prescribed a Student's t distribution for the likelihood "y-like." The incorporation of covariance was achieved with PyMC3's LKJ Cholesky covariance prior, by which the coefficients of the linear regression "y-hat" are drawn from a multivariate normal distribution and co-vary according to the predictor variables' covariance matrix. In comparison to our first model, the posterior distributions of rate and cost generated by this final model more closely mirrored the true distributions of rate and cost in that both were right-skewed and more concentrated about the mean. Additionally, we compared the WAIC (Widely Applicable Information Criterion) scores of both the first model and final model, which indicated that the final model had higher predictive accuracy. However, one disadvantage of the final model is its significant increase in run-time. We ran the final rate and cost models on a total of three computers, with run-time for both models ranging from 3.5 hours to nearly 6 hours. The cost model consistently had a longer run-time. Reparameterizing these models or switching to a non-centered version might reduce total run-time, though these avenues were not explored in this project.

In addition to considering a partially pool Bayesian model, we also trained two pooled models which ignored zip zone origin and destination. The first pooled model was identical to our first partially pooled model, though without zip zone information. This model did not converge well, as was evident in all traces of the coefficients. Similarly, we tested an unpooled version of the final model which ignored all zip zone information, again encountering the same problems. Since we expected the predictive accuracy of these models to be less than our partially pooled models, we did not explore any pooled Bayesian models further.

6 Bayesian Belief Network (BBN)

For the third layer in our model, we wish to generate two categorical variables, namely "color" and "CurrentCondition," from the continuous variables "rate_norm" (rate) and "est_cost_norm" (cost). However, we can currently only use a Bayesian hierarchical model to generate continuous variables, so in this step we are making use of what is called the Bayesian Belief Network (BBN).

BBN is a probabilistic graphical model that contains nodes and arrows between the nodes. Each node corresponds to a random variable and in our case, the random variables are columns in the data set. Existence of an arrow one node to another indicates that the latter node is dependent on the first node. In this case, the latter node is called a child node and the first node is called a parent node. This model specifies which variables are dependent, independent or conditionally dependent. It also tracks in real-time how each event's probability changes as new evidence is introduced to the model.

In our BBN, there are four nodes which are rate, cost, color and current condition. Here, color is dependent on both rate and cost whereas current condition is dependent on rate, cost and color. Since BBN cannot take continuous variables as an input, we first discretize rate and cost into four categories as follows: we first extract the minimum and maximum of the variable we wish to discretize and then divide this interval into four subintervals of the same length. After this step, we calculate probabilities for each category in rate and cost, which is the corresponding frequency of the variable in the original data divided by the total number of rows. Then, for a given pair of rate and cost categories, the algorithm computes the conditional probabilities for all colors. At this point, we have built up the BBN for generating color based on the probabilities and conditional probabilities calculated from the original data. Then, we put generated rate and cost from the Bayesian Hierarchical model into the BBN as a pair so that we can get the corresponding color for a given category. Finally, in order to generate current condition out of rate, cost and color, we follow a similar process.

Our results show that the distributions of color in both the original data and the generated data are similar, and the distribution of the current condition in the original data is very close to that of the generated data. Also, we notice that our generated data excludes some outliers that are seen for both color and current condition in the original data.

Algorithm 3 BBN

```
1: Extract the minimum and maximum of the column and divide the resulting interval into four
   (or possibly more depending on interest) subintervals of the same length.
2: Define a function that calculates the probability for each parent node and conditional proba-
   bilities for each child node based on the original data.
3: Define another function that takes the values of parent nodes as input, calculates the conditional
   probabilities of the child node by using the above function, and introduces a random variable
   generated by the uniform distribution on [0, 1] so that the final value of the child node is
   determined by the output of the random variable.
```

7 K-prototype Algorithm

We use the K-prototype clustering algorithm to test if the generated data and original data are easily separated. If the algorithm is unable to separate these two categories of data well, this means our generated data is very close to the original data. Then we can conclude that our generative model is performing well.

Algorithm 4 K-prototype

```
1: Create a "cluster_id" column: original data is denoted by "0" and generated data is denoted
   by "1"
2: Use the kprototypes() function
3: Obtain a new column "cluster_labels", with entry "0" referring to original data and entry "1"
   referring to generated data
```

8 Summary and Possible Improvements

The generative parametric model constructed by our team combines three different probabilistic methods to produce synthetic data which closely resembles the true transactional data provided by C.H. Robinson. We first determine the best fit distribution for continuous variables (order_distance, order_origin_weight) or multinomial distribution for discrete variables (lead_days, week_id) in order to generate the variables we assume to be independent.

Then, we use Bayesian hierarchical models to generate the rate and cost data. The input data is the generated continuous data and discrete data of our independent variables.

Next, we use the Bayesian Belief Network(BBN) to generate the Recommender System outcome (color) and the final decision of an order (CurrentCondition). The input data is the generated rate and cost data from the Bayesian hierarchical model.

Finally, in order to test the goodness of our model, we evaluate how our generated data compares to the original data. To accomplish this, we use the K-prototype clustering algorithm. The results indicated that the clustering model could not successfully separate the two categories of data, so we believe that our synthetic data is indistinguishable from the original data in the desired ways. It also shows that our model for generating synthetic data is pretty good.

There are several avenues for improvement of our model, in particular for improvement of each layer of our model. Since we are using the best fit distribution to generate continuous variables for specific zip zone origin and destination pairs, the noisiness of the continuous data for the majority of zip zone pairs makes approximating the true distribution via a probalitity distribution extremely difficult. Perhaps hierarchical modeling can instead be used to generate the independent columns in place of best fit distributions.

Several possible improvements for the Bayesian hierchical model were discussed in that section above, though one not mentioned is to increase the number of iterations used to achieve higher accuracy.

Fine tuning the categories for the continuous variables in the Bayesian Belief Network (BBN) can also give us better estimates of both Recommender System output and the final decision. Adding more nodes and factoring in dependency from the independent variables, like those related to time and location, can lead to a more comprehensive BBN model.

9 Approaches Tried but Not Continued

9.1 Naive Regression Method to Fit Distributions

Before we decided to get distributions for independent columns by sampling distributions from more than 70 distributions to find the best one, we tried to do a simple regression on a column where the input is not any other columns but a random variable sampled from a certain distribution, which we call the Naive Regression Method. We did it on order weight. We first normalized weight by minmax normalization. Then we fitted a linear regression with noise from standard Gaussian distribution and the input was generated from a normal distribution with size 1000. We got the intercept and coefficient of the input and produced new weight out of the regression formula. It turned out that there were many outliers, e.g., negative values which were not reasonable for weight. We also tried to sample the input from exponential distribution with $\lambda = \frac{1}{mean_{weight}}$. Unfortunately, the code did not work since it gave an error saying "Mass matrix contains zeros on the diagonal". Next, we fitted the weight into a quadratic model with input from a standard Gaussian distribution. The generated weight looked better though we still had some negative weight. When producing 100 values out of the model, 9% was negative while 7.2% was negative when generating 1000. It seems like there will be less outliers when generating larger data. However, it might cause an overfitting issue when using a nonlinear model.

We did not continue working on the method since we thought the best fit method was easier to implement and the time was limited too. The merit for this regression method is that it can fit the distribution for a random variable as a single distribution or a combination of several distributions, which might be helpful for the columns where the first half of its distribution is very different from the second half, like order weight. The disadvantage is that we have to decide which distribution the input is sampled from.

9.2 Naive Bayes Classifier

We implemented Naive Bayes Classifier to predict recommender color using only the estimated cost, order distance, and order weight columns. Since this classifier requires that the input variables be normally distributed, we applied the box-cox transformation to convert the order distance and order weight columns to have a normal distribution. The estimated cost column was normally distributed in the original dataset. The model had an accuracy (which is the ratio of the number of times it matched with the color in the dataset to the total number of data values) of 0.5851. This was a little over the null accuracy of 0.5815 which is the ratio of the maximum count of a color to the total number of data values. This could be due to the "naive" assumption that estimated cost, order distance, and order weight columns are independent although we know that the cost should go up as the order distance increases or the order weight increases.

10 Code

We used Jupyter notebooks to compile all code in Python for this project. All code can be found on Github at the following link: `https://github.com/lls133/IMA_Project_for_CHRobinson`