**A framework to predict sales using historical transaction records from a payment app**

Rui Zhang and Winston Li

1. Background  
     
   Sales forecasting plays an extremely important role in business planning. It is a useful tool that extracts information from past and current sales data to intelligently predict future performances. With an accurate sales forecast in hand, decision makers can optimize marketing strategies and business expansion plans with more confidence. For instance, if your sales forecast says that 50% of your annual sales would occur during Christmas, you will need to ramp up your inventory and production in the fall to prepare for the sales peak. You may also want to hire more salespeople after Thanksgiving. If you can have accurate sales forecasts for your competitors, and thus a better understanding of their priorities and targeting, you can get a head start when planning for promotion campaigns. A good sales forecast can inform many other aspects of your business.  
     
   For this project, we implemented a framework to predict sales using historical transaction records from a payment app. The framework applies macroeconomic indexes to constrain total transaction numbers and tabular variational autoencoders (TVAE) to simulate individual transaction records. Github link is [here](https://github.com/rui-zhang-ocean/Predict_Transactions).
2. Data  
     
   The dataset used in this project is from a payment app, a tabular data with over 3 million transaction records from September 2018 to January 2020. Each record contains customer (age, gender, address, etc), account (account type, bank, etc), transaction (transaction type, amount, date, etc) and retailer information (retailer name, sectors, etc). As a starting point, I selected variables as follows: `Account Type`, `Consumer Gender`, `Consumer Age`, `Transaction Date`, `Normalized Retailer`, `SIC Description (sector)` and `Purchase Amount`.

Table 1. Samples of selected variables in the tabular data



1. Methods

Synthetic data generators take input tabular data, learn a model from it, and output new synthetic data which has similar statistical properties as the input data. The most known application is to generate data that is not easily accessible or available due to private concern. According to a recent [benchmark study on synthetic data generation techniques](https://github.com/sdv-dev/SDGym) from MIT, tabular variational autoencoder (TVAE) synthesizer outperformed other synthesizers and is recommended.

The basic idea of autoencoders is to construct an encoder and a decoder as neural networks and to learn the best encoding-decoding scheme using an iterative optimisation process. The encoder is similar to dimensionality reduction techniques such as principal component analysis (PCA) which converts input data into representations in latent space, and a decoder randomly samples from the latent space and reverses that into tabular data. The training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative processes.

Here I used TVAE synthesizer to load historical transaction data as input, trained the model and output simulated transaction data. The output data reflects the consumption habits of the customers from the historical time period, but lacks constraint of the total transaction counts. Therefore, I set up a generalized linear regression model among historical transaction counts and macroeconomic indexes (GDP, CPI, unemployment rate, etc) and apply that to forecast future transaction counts. Besides, I also need to forecast future economic indexes based on historical indexes. The proposed workflow are as follow:

a) Given a specific year and month in future, for instance, February 2021, forecast the macroeconomic indexes based on the historical indexes.

b) Forecast the transaction counts N in Feb 2021 using the regression model.

c) Generate N transaction records using the trained TVAE synthesizer.

* 1. Generating synthesized data using tabular variational autoencoder (TVAE)
     1. Data cleaning and feature engineering

The tabular data was well maintained with less than 2% missing or invalid values (i.e. negative age), so for missing or invalid values I simply removed them.

Categorical variables with a number of values smaller than 10 were treated with one-hot-encoding, including `Account Type` (debit or credit), `Consumer Gender` (male or female) and `SIC Description`. Note that `SIC Description` only kept top 9 values with maximum counts and grouped the rest records into `other`.

I treated `Age` as a continuous numerical variable at first but the synthesized age distribution didn't look optimal. A large chunk of the spectrum between 30 to 50 years old were missing. I fixed it by converting age into `age range` (i.e. 20-25, 25-30, 30-35, etc) as the model input, and later convert the simulated range back to `Age` by randomly selecting a number from each range. This is a classical approach to transform continuous numerical values into discrete fixed width bins, and consider these discrete values as categories.

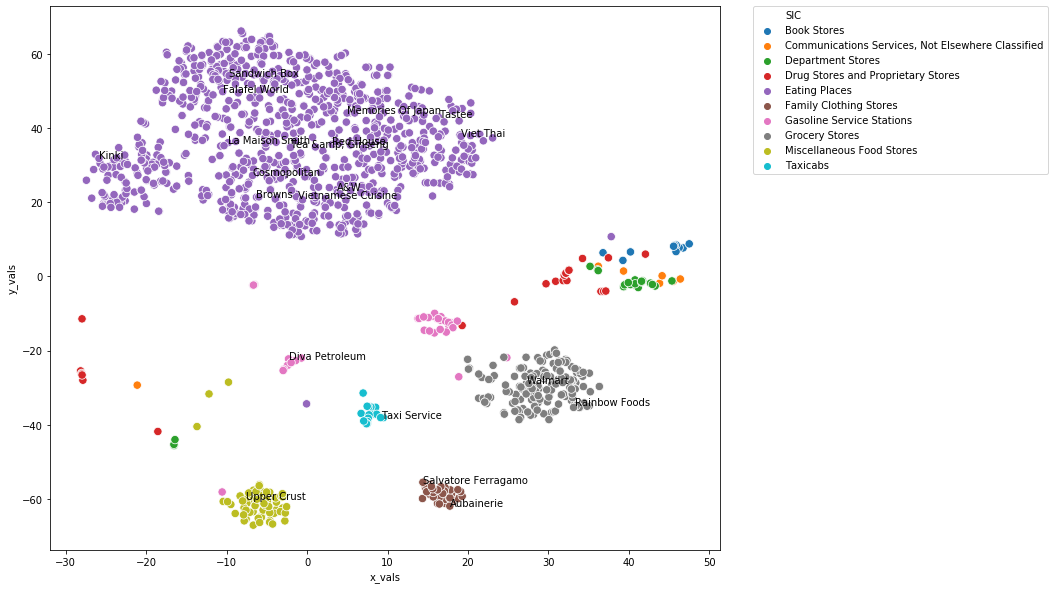
I broke down `Transaction Date` into two categorical values: `period of month` and `day of week`. `period of month` contains variables `start` (1-10 days of a month), `mid` (11-20 days of a month) and `end` (21 to the end of the month days) which refers to different periods of any given month. `day of week` refers to an individual day of the week (Mon, Tue, etc). This approach ignored the annual and monthly but focused on weekly and daily variations. The reason is that when forecasting transaction counts using macroeconomic indexes, which are often released monthly, we would have incorporated annual and monthly variability, thus the synthesizer doesn’t necessarily need to include that information from historical records.

`Normalized Retailer` is the most tricky variable to deal with. It contains over 2000 retailers so embedding is needed to reduce the dimensions to a manageable level. Here I applied `item2vec` to convert `Normalized Retailers` into 10 dimension embeddings. To provide readers with some context, in the natural language processing (NLP) domain, `word2vec` learns word embeddings that preserve the semantic characteristics of words and the relations between words by treating each sentence as `sets of words` that share the similar context. `item2vec` is one of its variants in the non-NLP domain, where it takes `sets of items` as equivalents as sentences (`sets of words`) to be the context and find out the relations between items. There are two major advantages of using embedding instead of one-hot-encodings. First, the reduced dimensions are manageable with limited memory. Second, it represents the distance or similarity of variables better.

“Researchers from the Web Search, E-commerce and Marketplace domains have realized that just like one can train word embeddings by treating a sequence of words in a sentence as context, the same can be done for training embeddings of user actions by treating sequence of user actions as context.” - Mihajlo Grbovic, Airbnb

The biggest challenge of training item embedding is how to define the `sets of items`, or how to find out the most relevant context for those items. For this project, I tested a few different grouping approaches on retailers: a) group retailers that have transactions with the same customer on the same date. b) group retailers that belong to the same sector and have transactions on the same date. c) group retailers that belong to the same sector and have transactions with the same customer.

It turns out that option c performs best, I plotted the scatter map of the retailers’ vectors after reducing the 10 dimensions to 2 dimensions. The color map indicates different sectors. It shows that the embedding can group retailers reasonably well.  
  
Figure 1. Retailer embedding clustered by sectors



I used the word2vec module from gensim to train the retailer2vec model. The word2vec module considers each sentence as the `sets` and has a parameter window size to determine the scope that we searched for `context` of each word. By contrast, in our retailer2vec implementation, the `meaning` of a retailer should be captured by all its neighbors in the same set. In other words, we should consider all retailers that belong to the same sector and have transactions with the same customer to define the `meaning` for each of those retailers. The window size then needs to be changed according to the size of each retailer set.

To address this problem without modifying the underlying code of the gensim library, one approach is to define a super large window size (e.g. 999999), which is way larger than the length of any retailer set in the training data. Although this workaround is not ideal, it does achieve acceptable results. The best option might be modifying the underlying codes in gensim.

When reversing synthesized retailer embeddings back to retailers, the model will only check distances among retailers in the synthesized sector (`SIC Description`), and pick up the closest one.

* + 1. Train the synthesizer  
       After feature engineering, I got an input matrix of 46 columns and over 3 million records. I used TVAE synthesizer from [SDgym](https://github.com/sdv-dev/SDGym), where training and generating samples are both well wrapped up in the module. Note that the function requires specification of categorical columns and ordinal columns. However, when I explicitly specify categorical columns for the one-hot-encoded variables, the synthesized data had trouble producing reasonable results. An alternative would be to assume all one-hot-encoded variables as continuous numerical values and select the absolute maximum of synthesized results among all columns, which works just fine. More validations can be found in section 4.  
         
       Considering the large size of the input matrix, it is recommended to train the synthesizer with GPU. Google Colab with free GPU access or Google Cloud Platform with 300$ credit for any new users would be a good start.
  1. Using macroeconomic index to constrain total transaction counts in future
     1. Collect and forecast economic indexes  
        I started with 5 common macroeconomic indexes which are: gross domestic product (GDP), consumer price index (CPI), Toronto stock exchange (TSX), unemployment rate and exchange rate (CAD to USD). To build a regression model among the eco indexes and transaction counts, I will need to collect those data first. Statistics Canada provides [API services](https://www.statcan.gc.ca/eng/developers/wds/user-guide#a12-3) which provide access to data and metadata released on each business day. The API commands are further wrapped up by a Python library [stats\_can](https://stats-can.readthedocs.io/en/latest/) which can be easily called within any script to retrieve the latest eco index data. All that users need to do is to specify a `vectorID` for desired index as the input. It is worth noting that by default the `vectorID` is not shown on the index table of Statistics Canada website, the user needs to turn on `Display Vector identifier and coordinate` in the table settings to make it visible. I will give an introduction on how to use `stats\_can` to automatically retrieve data from Statistics Canada in a different article.

I used `[prophet](https://facebook.github.io/prophet/)`, an open source library developed by facebook, to predict the economic indexes. The library is for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. For instance, given GDP records in the last three years (36 monthly records in total), we can predict GDP for the next month.

* + 1. Generalized linear regression model  
       I started with a multiple variable linear regression as baseline, but found the forecasted transaction counts could be negative sometimes. A natural alternative would be to use a generalized linear model (GLM). Here I used a GLM with Poisson distribution and a logarithm link function, correlating historical monthly transaction counts with monthly economic indexes.  
         
       Ideally the forecasted transaction counts can be verified as the process continues with more data coming in every month.

1. Results  
   To verify if the synthesized data exhibits similar statistical properties as the input data. I examined the age and purchase amount distribution, barplots of purchase amount for top sectors and top retailers (Table 2).  
     
   Although the synthesized purchase amount can reflect the total distribution well, it has trouble resolving sales for each retailer, to deal with that, I applied a post-correction to `Transaction Amount` for each retailer by scaling with the average transaction amounts for each retailer from input data. With that post-correction, the purchase amount per retailer for synthesized data fits much better with input data, although some outliers are missed.  
     
   Table 2. Comparison between input and synthesized data

|  |  |  |
| --- | --- | --- |
| Variables | Input data | Synthesized data |
| Age |  |  |
| Total purchase amount |  |  |
| Purchase amount per sector |  |  |
| Purchase amount per retailer |  |  |

1. Discussion

This project provides a framework to forecast sales using historical transaction records from a payment app. The basic idea is to forecast total sales volume using macroeconomic indexes and then produce synthesized transaction records to fill that volume. The framework utilizes methods such as item embedding, tabular variational autoencoder (TVAE), time series forecasting and generalized linear models. It produces reasonably well forecasts that can be used to inform many aspects of a retailer’s business.   
  
However, there are still rooms for improvements. The retailer embedding can be improved by defining more sets that correlates relevant retailers. For instance, for the largest group `restaurant`, if we can scrape tags or consumption level ($$$) from apps such as Yelp, and define sets based on those information for training, the retailer embedding could better represent relations among all restaurants, which eventually could improve sales for the restaurant retailers. For now we still need post-correction on sales to get a better sales distribution for each retailer, it is expected that an improved retailer embedding could help getting rid of the post-correction.

For future work, I plan to deploy the model on Google Cloud Platform. At the start of every month, the model will automatically collect economic indexes from last month, forecast indexes and total transactions for the current month, and then generate current month transactions in detail. It will also allow retraining of the synthesizers when new transaction records come in and forecast months in the further future.