1-1-2015

A Customer Choice Modeling Framework For Assortment Planning Of Configurable Products In Automotive Industry

Farah Dubaisi
Wayne State University,

A CUSTOMER CHOICE MODELING FRAMEWORK FOR THE ASSORTMENT PLANNING OF CONFIGURABLE PRODUCTS IN THE AUTOMOTIVE INDUSTRY

by

FARAH DUBAISI

THESIS

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

2015

MAJOR: MANUFACTURING ENGINEERING

Approved By:

____

Advisor Date

ACKNOWLEDGEMENTS

The authors wish to acknowledge her parents Talal Dubaisi and Elham Dubaisi as well as her sister Sarah Dubaisi and Brother Najib Dubaisi for their continuous support and motivation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

In order to survive in this highly competitive market, retail stores should come up with effective and efficient ways to manage their operations to yield the highest profit and customer satisfaction possible. Activities such as ordering of products, inventory management, and establishing relationships with suppliers, significantly contribute to operational cost incurred by the retailer. Thus optimizing them will lead to a boost in the company's profits. These days a major market focus is being directed toward 'Assortment Planning' which is defined as specifying the set of products and the level of product variations to be carried at each retail store in a way that will maximize the store's profit, subject to storage space constraints, customer service level, product availability, competition, and many other possible constraints depending on the retail store and type of product being studied. One important tradeoff that should be considered in assortment planning, is that increasing variety increases customer satisfaction but has a negative effect on operational cost. To mitigate this problem, the retail store managers should be able to understand the customer buying behavior at the point of sale, and their reactions toward not finding their desired product variant, whether this variant is stocked out or is not carried by the store.

Being able to predict the probability of selling a given product offered within a specific inventory mix, is a valuable asset not only for the retailers but also for the supply chain as a whole. In this context, retailers will adjust their carried assortment by ordering more of the higher sellers and less of the slow moving configurations. This in turn will reduce their holding and operating costs by fairly cutting down the average weeks a configuration stays on lot. Moreover, it will increase the level of customer satisfaction by lowering the possibility of stock outs. Finally, it will increase the revenue since more vehicles will be sold at full price and less promotions will be necessary to get rid of stationary inventory. As for the rest of the supply chain, manufacturers and suppliers will focus their production efforts and budgets on producing the popular configurations and will cut on the undesired ones to be able to replenish their retailers' inventories as quickly as needed. This will lead to a significant reduction in the complexity cost and starvation points in downstream stages of the supply chain.

Driven by the above mentioned benefits of estimating the selling probability, this thesis focuses on estimating the turn rate of configurations present in an existing assortment. As a definition, inventory turnover rate is a measure of the number of times inventory is sold or used in a time period. In other words, it's the probability that a vehicle will sell within 1 time period from its arrival to lot. This is numerically calculated as follows:

$$(Turn\ Rate)_i = \frac{1}{(Average\ weeks\ on\ lot)_i}$$

Where *i* is the configuration index.

Given that vehicles are configurable products, its attractiveness can be represented as a function of its individual features in accordance to the majority of customer choice models in the configurable products literature. The most widely used choice models in the literature of economy are the multinomial logit MNL, multinomial probit MNP, and mixed multinomial logit models. All of which equally estimate the deterministic portion of the configuration's attraction as a linear relationship between the individual features constituting it. However, they differ in the way they compute the error term which is added to the later to account for randomness and interactive relationship between features. In this thesis, we relied on Ford Mix Rate Modulated Patent to calculate the deterministic portion of the configuration attraction, which is compared to the turn rate variable explained above. This model is constructed to generate a matrix that combines the item configuration data with the inventory mix data at the feature level to output a configuration feature vector whose elements indicate the availability of an item feature in the inventory carried on lot. Consequently, each feature will be represented as a function of its availability in the configuration, regardless to any other feature in the configuration. This in turn will allow us to study its contribution on the final output which is the turn rate. However, this methodology has introduced another problem, which is the dependency of the turn rates on the corresponding inventory mixes that were available on lot. That means that for each inventory mix scenario we will have a new set of turn rates associated to each configuration. This will leave us with infinite variable space given that we have unlimited possible inventory mixes. Therefore, the second part of the thesis focused on removing this conditionality and generalizing the turn rates so that we have one turn set applicable to any kind of inventory mix.

The thesis will be organized as follows, a literature review section that will discuss other methodologies used by researchers to estimate probability of sales. In addition to in depth explanation about Ford’s patent and neural networks. Afterwards, a methodology section will be presented where all the details and assumptions followed by our model are explained. And lastly the results will be presented and validated in the result and validation section. We will wrap up this thesis with an insight of future applications and elaborations to our turn rate estimator model.

LITERATURE REVIEW

In the assortment planning literature, Ryzin and Mahajan 1999 where among the first to focus on the tradeoff between the higher revenues achieved by larger product variation and the extra operation costs imposed by this larger variety, including the inventory-related costs. In Ryzin and Mahajan1998 they studied assortment planning problem related to non-configurable products with a stochastic demand single period setting using Multinomial Logit (MNL) consumer choice model which is a utility based model based on the assumption that customers buy the variant that maximizes their derived utility. Their model optimized the initial inventory mix that should be carried by a retail store in order to maximize the store's profit, taking into account the effects of stock outs by studying customer behaviors at those instances. Facing a stock out, a customer is expected to either substitute to another variant, stock-out based substitution, or walk away. In Ryzin and Mahajan 1999, the same inventory allocation model was elaborated to account for assortment based substitution as well, which is the probability that a customer substitute his primary preference with another variant having in mind that his preference is permanently not carried by the store. Only one level of substitution was allowed, after which the customer is supposed to walk away. In both papers the price allocated to the variants was assumed to be exogenous to the model. A basic set of inputs for their formulation was the utility vector which includes the set of utilities assigned to each product variant in the offered assortment along with the no purchase utility. They interpreted these parameters as a measure of the net benefit to the consumer from purchasing each variant (or not purchasing) which is also called consumer surplus and is numerically represented as follows:

$$Uij= uij+€ij$$

Where *uij's* are the deterministic portion of the utilities and are further decomposed to a quality index minus the price of variant j,

uij=aij-pij.

€ij's are the set of mutually independent random error terms that account for the unobserved heterogeneity in the customers' taste. According to the IIA assumption, Independence of Individual Alternatives, this error term follows a Gumbel distribution with mean 0 and variance μ*π/6.

Kok and Fisher 2007 also derived an exogenous probabilistic choice model for commodity non-configurable products and utilized a novel substitution estimating approach by applying the estimation maximization technique to the sales data. They modeled the consumer buying behavior as a function of three decision variables: 1- Whether or not to buy from a subcategory, 2- which variant to buy, 3- how many to buy. This technique has been heavily used in marketing literature and can be expressed mathematically in the following manner:

$$dj = K\pi Pj$$

Where K is the number of customers, π is the probability of purchasing incidence, Pj is the choice probability of variant j and qj is the quantity purchased by per purchase incidence. To account for substitution, the effective demand of variant j is expressed as follows:

$$Dj = dj + \sum_{K\phi N} \propto kj * dk + \sum_{K\in N} \propto kj * Lk$$

Where dj is the direct demand to variant j, $\propto kj$ is the probability of a customer substituting from variant k to j, dk is the direct demand for variant k,and Lk is the unmet demand of variant K. The first summation represents the assortment based substitution where the customer substitute from a variant that doesn't belong to the carried assortment N to variant j in N. However, the second summation represents the stock out based substitution, where a customer replace a variant k which belong to *N* but is currently out of stock. The distinctive aspect about this paper is that it involves a real life assortment planning problem along with real sales data, an approach to estimating the parameters of the model, and a workable algorithm validated by the real data. They presented an iterative optimization heuristic for the assortment planning and inventory problem with one-level, stock-out based substitution subject to shelf space, lead time, and discrete maximum inventory level constraints.

Yucel et al. 2009 branch out from the Kok et al and introduce a mix integer optimization model for the joint problem of product assortment, inventory management, and supplier selection. The output of this model is the optimal order quantities for each product, as well as the product types that should be included in the assortment. Another novel approach of demand estimation was presented in Ozturk et al. 2009. A special focus on how to estimate stock out based substitution has been given in this paper. Their base model utilized the point of sale data and inventory transaction records to estimate the probability of substitution as well as sale probability under the assumptions of discrete time stochastic customer arrival rate, length of the POS interval is short enough to ensure that the probability of having two arrivals during the same interval is negligible and can be assumed zero, and only one level of substitution is allowed with probability psi δ. With the above stated assumptions, arrival rate λi and sale probabilities of each variant *i* offered in an initial inventory $Io,\ Si, Io$ can be expressed as follows:

$$\lambda \mathrm{i} = \lim(\ T \rightarrow \infty) \frac{1}{T} \sum Ai(n)$$

Where *Ai (n)* is an indicator binary variable that is equal to 1 if a customer demanding product *i* arrived in period n, and 0 otherwise

$$Si, Io = \ \lambda i + \sum \alpha ij * \lambda j(1 - \theta j, Io)$$

Where, αij= δ*$\frac{\lambda j}{\sum \lambda l}$ is the probability of substituting product *i* with j. The latter is defined according to the market share based model

Si, Io Is the probability that a customer will purchase item *i* at POS interval n. Io is the inventory status at the beginning of the POS interval, Θj is a binary indicator variable that is equal to 1 if product *i* is available in Io and zero otherwise.

All of the above papers focus on consolidated commodity products such as beverages, food, detergents, shampoo, etc… Goker et al. 2009 on the other hand discusses the assortment selection and pricing for configurable products such as computers, mobile phones, cars, etc… They categorized the components that constituted the product into required components and optional ones. They defined a variant's surplus which is the difference between the customer utility from a variant and the costs incurred by the firm for the variant. The variants that the company should choose to include in the configuration are those with the highest surplus. They also defined an attraction factor that rates the attraction of the whole configuration, and with that the company can choose which configuration to include in their assortment. The MNL choice model has been implemented to estimate demand without accounting for any type of demand substitution.

On the other hand, Ford came up with a different assortment planning approach, at the level of dealers. Their goal was to generate order recommendation to dealers that better addresses the given dealer's market, and maximize their profits through reducing holding costs impacted by the average weeks a vehicle is set to spend on lot, and costs associated with exchanging vehicles between dealers. Ideally when a dealer orders the right configuration mix, they are expected to sell faster, encounter less stock outs and apply less price promotions to get rid of slow moving items. In their Smart Inventory Management System, SIMS, model they utilized statistical analysis and neural network to predict vehicle turn rates in a given inventory mix. In the beginning a dealer's market is defined as a circle of 25 miles radius around the target dealer, this will allow the inventory of the target dealer to be viewed in context of other inventory available in dealer market. Accordingly a weighted market inventory for the target dealer is calculated using a weighting function that associates metrics to the inventory available at competing dealers based on their distance from the target dealer and adds the above product to the inventory available at the target dealer. In other words, if a given dealer has 10 blue Fusions while another competing dealer 10 miles away with a weight w= 0.2 has 4 fusions, then the weighted market inventory will be 10+0.2*4=10.8. The weighting function is a monotonically decreasing function with a value 100% when distance is zero, and slightly higher than zero for distances more than 140 miles. Similar analysis is done for sales data. Secondly, the inventory and sales data are broken into the feature level, then configuration feature vectors are normalized according to the mix rate modulated technique which will be explained in further details later in this paper.

For computational simplicity, they applied PCA technique to reduce the dimensionality of the feature vector that will be later inputted into the neural network. In addition to the mix rate modulated feature vectors, a set of context variables will be inputted to the neural network which will capture all the market related characteristics such as dealer latitude and longitude, dealer item market inventory, retail/stock order type indicator, numbers of weeks on lot, dealer fraction of item market inventory, and market turn rate. A vector of binary variables representing the sold status of a given vehicle on lot on a given week, is passed to the neural vector in the form of target variable. Using survival analysis techniques, the neural network will be able to predict the turn rate of a vehicle with a given normalized feature vector, after being trained on a set of historical inputs and known outputs. During the training process the model will assign certain parameters to each input variable, to generate a predictive function for the designated output. The neural network training process will then adjust these parameters in response to the applied input variables to better match the output of the neural network with the real historical target values.

Later in the SIMS model, after they assign a turn rate for every configuration, mixed integer optimization model called "Feature Allocation Optimization" was formulated with an objective to minimize the difference between the target inventory mix rate and the current mix rate at the feature level subject to production and material availability constraints. The target inventory mix rate is set in a way that guarantees a balanced inventory where inventory mix for a given feature is aligned with its sales mix. For example the graph below shows the projected sales and mix rates of two variants of the engine. To attain a balanced inventory the top 2 curves should overlap as well as the bottom 2 curves.

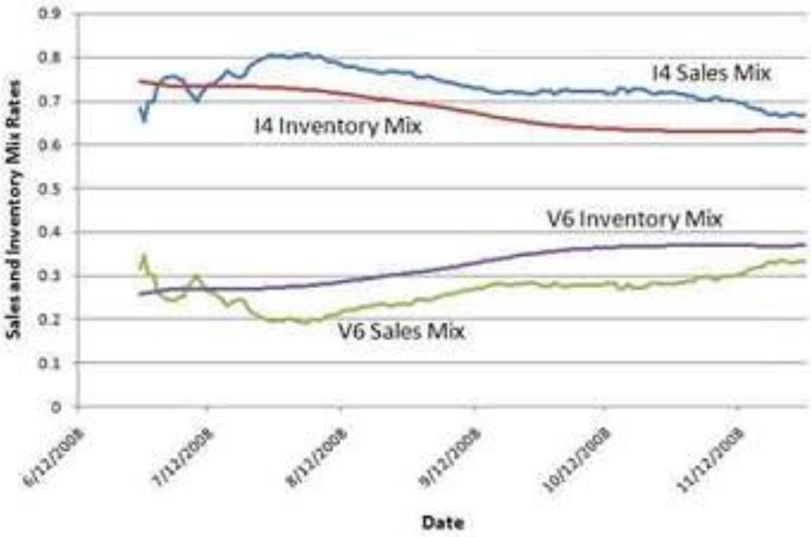


Figure 1 Sales and Inventory Mix Rates as a Function of Time

METHODOLOGY

Assumptions

1- Ignoring the Differences in The Buying Behavior From One Consumer to Another

The model considers that the customer’s choice is fully determined by the inventory mix available on lot, and is independent of the customer’s personal characteristics. It does not differentiate customers belonging to different age groups, financial statuses, region of residency, and so on. On the average, all customers in the US market are expected to have the same choice behavior when exposed to the same inventory mix.

2- Ignoring Seasonality in Sales Data

According to Ford’s analysis of their sales data, seasonality has a negligible effect on the projected turn rate of a variant of a given feature relative to other variants in the same feature family. The graph below, for example, shows that the variation of the relative turn rate of I4 to V6 engine is fairly constant over time. Thus we can safely ignore seasonality effects without leaving any bad impact on the accuracy and precision of the model.

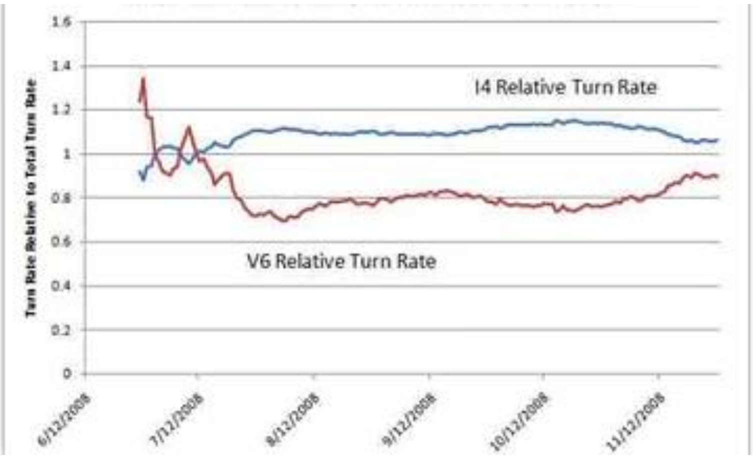


Figure 2 Variation of I4 and V6 Engine's Turn Rates as a Function of Time

1- Decomposing the US Market into Regional Submarkets

We decomposed the US market into 17 different regional markets, which are listed in appendix A. Dealers in a given regional market have access to the inventory available in the whole market and they can easily trade vehicles without extra costs incurred on their end. So, whenever a customer walks into a dealership he will have the freedom to pick from the inventory available in all the dealerships located in this given region. On the other hand, no interaction is allowed across separate markets.

2- Considering Most Popular Features in Defining the Available Configuration Set, Core Entities

A configuration is best defined as a combination or arrangement of a set of feature variants. Knowing that for The Car model there are two different technologies to start with, Standard and Hybrid. For standard technology we have 4 different super-families, Power and Handling, Interior, Exterior, and Safety, each having 4 different feature families on average, and 5 variants each on average. Therefore, an assortment can include up to 2^80 possible configurations. Plenty of design and manufacturing constraints will shrink down the size of the assortment into a set of buildable configurations, nevertheless the assortment will still be a fairly large one. For this reason, we had to reduce the dimensionality of the feature vector by focusing on primary features which are assumed to have the highest influence on customer choice. Configurations are then encoded by a binary vector, whose length is determined by how many features are being considered. For each feature in the list, a value of 1 is assigned in a field associated with it, if the feature exist with respect to the given configuration, and a value of zero otherwise. The table below shows all considered features.

1- Cargo Cover	2- Radio
3- Ultimate Package	4- Rear Heated Seats
5- Heated Seats	6- Reverse-Sensing System
7- Elite Package	8- Satellite Radio
9- Rear Entertainment System	10- Sound System
11- Roof Rack	12- Trailer Tow Package
13- Moon Roof	14- Special Wheels

Table 1 Core Entities Used in Our Model

3- Ignoring Feature Interactions

This assumption states that the relative attractiveness of a given feature is independent of the set of other features present in a configuration. In other words, it ignores the feature packaging effects on the customer’s choice. For simplicity in our analysis, we considered a package as a unity and treated it as a single feature. This assumption is derived from all Logit choice model which restrict the explanatory variables to be independent and have fixed utilities. In our case, the presence or absence of a given feature represents the independent variable, and the utility is reflected by its associated customer attractiveness. For example, the attractiveness the moon roof option is fixed over all possible configurations independent on what other features are offered with it.

4- Removing Censored Data

The SIMS model utilizes the survival analysis technique to estimate from a set of historical sales data, how likely a given configuration will sell when present with a set of other configurations. Some of the vehicle records available in the Car model dataset, doesn’t indicate a day of sale because their selling incidence didn’t occur by the time of the close of the study. This phenomenon is called “Censoring”, and will lead to underestimating the probability of sales of a given configuration if not properly mitigated. As a definition, a record is said to be censored when information on time to event is not available due to loss to follow-up or non-occurrence of outcome event before the trial end. In our model, we eliminated those records from the dataset that was inputted to the neural network after applying the mix rate modulation.

Model Formulation

Data Preprocessing

For preprocessing the dataset in hand, we applied the mix rate modulation technique in order to normalize the configuration feature vector. A feature level inventory mix rate is best defined as the ratio of vehicles available in stock at a given period and market, having a certain feature, out of the total number of vehicles carried in the inventory. The modulation technique will take these mix rates and will subtract them from the binary vector representation of the vehicles’ configurations available on lot in order to generate a compact representation of these configurations, reflecting its’ relation to the inventory mix defined at the feature level. Elements of these mix rate adjusted feature vectors are continuous variables bounded between -1 and 1 , because the mix rates themselves lies in the [0,1] interval and are positive whenever a feature is available in the given configuration, and negative otherwise. This modulation is characterized by its ease of reverse, since it’s almost always possible to recover the original binary configuration vectors, except for one case where all items carried in the inventory on that particular period do or don’t carry a particular feature. In this case in particular, we can remove that feature from our analysis because there will be no variation encountered at its level.

Furthermore, the modulation holds the following property, *Property1*, which states that the sum of the values across all features within a feature family for a given record are equal to zero, and the average value across all records for a given feature in any given time interval must be equal to zero. In order to satisfy this property, the configuration vector should include all possible alternatives for a given feature. For example if we have 2 extra options for car seats, “Front Heated Seats”, and “Front & Rear Heated Seats”, we should include 3 separate columns in the input matrix each standing for one extra option, and the additional column will represent the standard form of the feature, in our case it will be “No Heated Seats”. In this way all possible configurations will be covered, and the sum of normalized values across all features for a given record will be zero.

In our model, for the sake of reducing dimensionality of the input space, we only considered features which were chosen by The OEM’s marketing department to have the highest effect on the customer’s choice. Moreover, since we are not studying packaging effects and the interaction between features, we considered the packages which were offered to the US market in year 2007/2008 as a single feature.

In summary, the input feature vector will be a single row vector combining all the feature variants listed in the table below. Notice that Elite Package column has been duplicated because this package acts as an alternative to 2 different features, Rear Entertainment System and Roof Rack. Thus for the elements of our input feature vectors to sum up to zero according to the *Property 1*, this package column should be counted twice.

Feature	Variant 1	Variant 2	Variant 3
F1	Ultimate Package	Heated Seats	Non
F2	Elite Package	Rear Entertainment System	Non
F3	Elite Package	Roof Rack	Non
F4	Moon Roof	Non	
F5	Radio	Non	
F6	Rear Seats	Non	
F7	Reverse Sensing System	Non	
F8	Satellite Radio	Non	
F9	Sound System	Non	
F10	Trailer Tow Package	Non	
F11	Special Wheels	Standard Wheels	

Table 2 List of Variants for Each Feature Family[1]

Now that the feature vector is defined, it will be encoded as a binary array explained in assumption 4 and will contain 27 elements. Each vehicle in the dataset will be associated with a feature vector to describe which features it conveys. In the Car model sales dataset, we had 579 unique configurations. For simplicity we ranked those configurations randomly from 1 to 579, so that we can refer to each one by its assigned ranking rather than a 27 element binary vector.

Next step in the mix rate modulation process, is to categorize sales records based on their location, then discretize on a weekly basis. Take Boston area for example, we have 1128 vehicles

[1] The cells with “Non” values stand for the No feature option, empty cells means there is no third variant for that particular feature

arriving into the market in the period 2006/2007, total of 44 weeks. Those vehicles were randomly ranked from 1 to 1128 to keep track of each as it get transmitted from one week to another. We recorded all the vehicles available at a given week N, where N extend from 9 to 44, by considering those vehicles that arrived before week N and was either sold during week N or later. After attaining the inventory mix at week N, mix rates for every feature in the feature vector will be calculated by averaging the value of the feature element within the feature vector across all vehicles available on lot at that particular field. In other words, the mix rate of feature I is the fraction of vehicles carrying I out of the total number of vehicles available on lot. This mix rate is later subtracted from the 1 or 0 encoded in the feature field of each vehicle available in the inventory set yielding to a negative value in those vehicles that doesn't carry the feature, and a positive value less than 1 otherwise.

Meanwhile, another variable which indicates whether or not a given vehicle was sold on that given week is defined. We called this binary indicator variable as sold status, and assigned it a value of 1 if the vehicle sold week matches the current week in hand, and 0 otherwise. For example if a vehicle V arrives on week 5 and got sold on week 9, it will show up in the inventories of the weeks 5 through 9, and it will have a sold status =0 on weeks 5,6,7,8 and sold status=1 on the 9th week.

The same procedure is repeated for every week, and every region out of the 17 US regions. Using Matlab as a tool, the weekly mix rate modulated vectors for each region, were generated and stored in a matrix. We will refer to this matrix as 'Global' throughout this thesis. Moreover, the Matlab code will also generate a 'Sold Status' vector that indicates the sold status of each vehicle in the record.

Neural Network Regression Model

As previously mentioned, Neural Network is the tool used in our regression model to calculate the expected inventory turn rates at the configuration level. The input to the neural network is the Global matrix, which includes the set of mix rate modulated feature vector available in the 17 regional markets over the period extending from week 9 till 44. On the other hand, the Sold Status vector will constitute the neural network target variable.

Several limitations are imposed on this predictive model due to the nature of the sales dataset, and the kind of output we are anticipating. Consequently, the built in neural network tools, offered by Matlab, didn't generate the required level of accuracy in its generic form. For this reason, we customized our own neural network script that better mitigates the some of the encountered limitations.

Limitation 1: Limited Number of Records

The feature vectors inputted into the neural network is made up of 27 elements, as described in the data preprocessing section above. This large variable space requires a significantly huge number of records in order to capture the contribution of each input to the final outputted result which is the sold status in our case. The limited number of sales record available in our dataset will definitely introduce accuracy problems to the neural network.

Limitation 2: Data Sparsity and Class Imbalance

From the historical sales records, it is observed that on average a vehicle is expected to remain 6 weeks on lot before it gets sold. Thus, for each vehicle we will have 6 sold statuses equal to zero and only one sold status equal to one. The generated sold status vector will then have 6 times more zeros than ones. This sparse target vector will enforce limitations on the neural network's ability to learn how to classify vehicles as sold or not. Getting rid of censored data is one way to mitigate this problem. Another way is to replicate the records which have a sold status equal 1, six times each, so that a balance in the target vector is created. In that way the neural network will have better ability of predicting the sold status of a given feature vector. Note that class imbalance only effects the ability of a neural network to classify its outputs onto one of the classes mentioned in the target vector. However, it doesn't have any effects on the regression fit model which will still generate accurate turn rates even if the target vector is not balanced.

Limitation 3: Variation in the Popularity of Configuration

Certain vehicles are significantly more popular than others and thus they should be given higher importance in the penalty function calculation. In other words, errors encountered in the predicted turn rates of those popular configuration must be highly penalized, as compared to those rarely ordered vehicles. Therefore, we introduced a weight vector into the neural network's penalty function in that way the neural will be trained to give more importance to those vehicles with higher weights. The weights are defined according to the formula below:

$$Wi = \frac{Number\ of\ Vehicle\ Weeks\ for\ Configuration\ i}{Total\ Number\ of\ Vehicle\ Weeks\ Records}$$

In this context, vehicle weeks represents how many times a given vehicle appears on lot before it gets sold and is numerically calculated by taking the product of the vehicle count by their associated weeks on lot:

$$\text{Vehicle Weeks for Configuration A} = \sum_{i=1}^{n} 1xWOLi$$

Where *n* is the number of times a dealer received vehicles of configuration A, *WOLi* is the weeks on lot spent by each vehicle I, and 1 is the count of vehicles received at each incident n, since each sales record represents 1 vehicle only. For example if we received 10 vehicles of configuration A, and each one remains 3 weeks before it got sold, the number of vehicle weeks for configuration A will be the product of the count with the weeks spent on lot, 10x3=30.

Customized Neural Network Script

Before inputting the Global matrix into the training function, it’s preferable to standardize the feature vector so that the mean of all inputs across one record is equal to zero. Standardizing the inputs and target variable is desirable because it rescales those variables so that their variability reflects their importance, makes training faster, and reduce the chances of getting stuck in local optima. Also, weight initialization, weight decay and Bayesian estimation can be done more conveniently with standardized inputs. For this matter, we utilized the ‘mapstd’ process function, which process the matrix by transforming the mean and standard deviation for each row to 0 and 1, for both the input and the output layers.

To start creating the neural network, one should first define the network object. For this purpose we used ‘newff’ function which by default creates a 2 layer feedforward neural network and requires 3 obligatory input arguments, input vector, target vector, and number of neurons in the hidden layers. Note that the output layer size is determined from the target vector. In addition to 3 other optional arguments which can be used to customize the functions used in the neural network including transfer function to be used in each layer and the utilized training function. If only three arguments are supplied, the default transfer function for hidden layers is ‘tansig’ and the default for the output layer is ‘purelin’. The default training function is ‘trainl’. The reason we chose newff to create a feedforward neural network although it has been obsolete since 2010, is that newff is the easiest to customize than any other feedforward function including ‘fitnet’, which is the new data fitting function, and ‘patternnet’, which is the pattern recognition function.

We already know that the optimal number of hidden neurons lies somewhere in the middle of the interval extending from the number of outputs to the number of inputs, in this case the interval is [1,27]. A good initial guess will be 10 neurons, thus we initialized our model based on that and then iteratively ran the network, recorded the average turn rate across all records and compared it to the average historical turn rate. After several trials, we figured out that the most precise results were generated when the number of hidden neurons was equal to 15.

For the hidden layer, we kept the default transfer function which is the ‘tansig’ function. The fact that ‘mapstd’ was used as a post processing function, limited our options to using either linear transfer functions or hyper tangent function at the output layer. However, since the output represents the probability of sales, the range of output variables should be bounded by 0 and 1. Therefore, the use of ‘purelin’ transfer is a must to satisfy both constraints. Knowing that, if linearity wasn’t mandatory, ‘logsig’ transfer function would have generated more accurate results due to its higher flexibility and degrees of freedom.

The penalty function used in this network is the ‘mse’ which compute the mean squared normalized error between the network outputs and the target outputs t. In Matlab, the syntax of this function is the following:

```
perf = mse(net,t,y,ew)
```

net	Neural network
t	Matrix or cell array of targets
y	Matrix or cell array of outputs
ew	Error weights vector defined above

Table 3 Parameter Definition on the Performance Function

Now this penalty function is passed to the neural network through the performance function, (net.performFcn), which is later used as part of the neural network training. During training, the weights and biases of the network are iteratively adjusted to minimize the network performance function using the backpropagation technique derived from the chain rule of calculus. Backpropagation perform gradient computation backwards through the network, and moves weights in the direction in which the performance function decreases more rapidly.

We had two different approaches to solve this problem, the first one is to treat it as a regression model that tries to fit a relation between the individual feature elements and the targeted binary sold status, SS, and will generated a continuous output, Yc, ranging between 0 and 1 representing the probability of sales for each record. In this case, the performance function will penalize the neural network by calculating the error between the Yc and SS. The second approach treats the problem as a classification problem, where the objective is to correctly classify each vehicle as being sold or not at a given week. Note that the train function will still fit a regression function between inputs and target vectors, and will still generate a continuous variable Yc, however the performance equation should now penalize the error between the classified output of the neural network with the SS target variable. In this matter, class imbalance problem should be mitigated by applying the balancing technique described above, and Yc should be converted to a binary classified variable which we called Yclassified. For this conversion we defined a threshold T=0.5 for Yc values, beyond which the vehicle is considered as sold. In other words, if Yc exceeds 0.5 then its corresponding Yclassied will be equal to 1, otherwise it will be zero. Below are the functions used to define Yclassified and the performance arguments:

```
Yclassified= round(y-T);
Performance = perform(net,t,Yclassified);
```

To study the effect of class balancing on the neural network, we ran a benchmark classifier model, where data were inputted with its unbalanced format, and the classification threshold was set to be=0.14, which is the average historical turn rate for all configurations calculated manually. We then compared the accuracy of this model with the balanced classifier model.

RESULTS AND VALIDATION

Using Matlab 2014a student version, we ran the two neural network codes explained above to evaluate the first set of results which is the conditional turn rate for each vehicle-weeks record in the dataset. We call those outputs conditional because their values are directly related to the inventory mix which was available on lot on a particular week in a given market location. For example the same vehicle which stayed on lot from week 3 till 7, will have 5 different turn rates values depending on the inventory mix that was available with it during each week. The goal now is to find a correct way of converting those conditional turn rates into unconditional, where each configuration is characterized by a single turn rate independent from the inventory accompanying it, this is consider the second set of outputs from our model.

Conditional Turn Rate Analysis

First, we ran the neural network in a “For loop” of 20, 60, and 100 iterations, and took the average of the turn rates across those iterations. Averaging is the simplest and most effective way to diminish the effects of randomness and noise in the generated readings. By averaging a set of replicated measurements, the signal-to-noise ratio, S/N will be proportionally increased with the square root of the number of measurements. This relation is expressed in the formula below:

$$\frac{S'}{N'} = \frac{nxS}{\sqrt{}nx\sigma\text{^}2} = \sqrt{}n\frac{S}{N}$$

Where *S’* and *N’* are the averaged signal and noise values, n is the number of readings, and S and N is the signal and noise strength for a single reading.

First step in the validation process, was to make sure that the neural network is precise where the coefficient of variation which is the ratio of the standard deviation to the mean of the turn rates predicted for a given configuration, is far less than 1. Then evaluate its accuracy as compared to the historical data. Resulting plots are shown in the sections below.

Regression Approach

For each record, we took the average turn rate of all the predicted values outputted from the neural network across the 100 iterations. We then recorded the standard deviation for those discrete values, and plotted it as a function of the mean. The scatter plot below proves that the coefficient of variation is fairly smaller than 1.

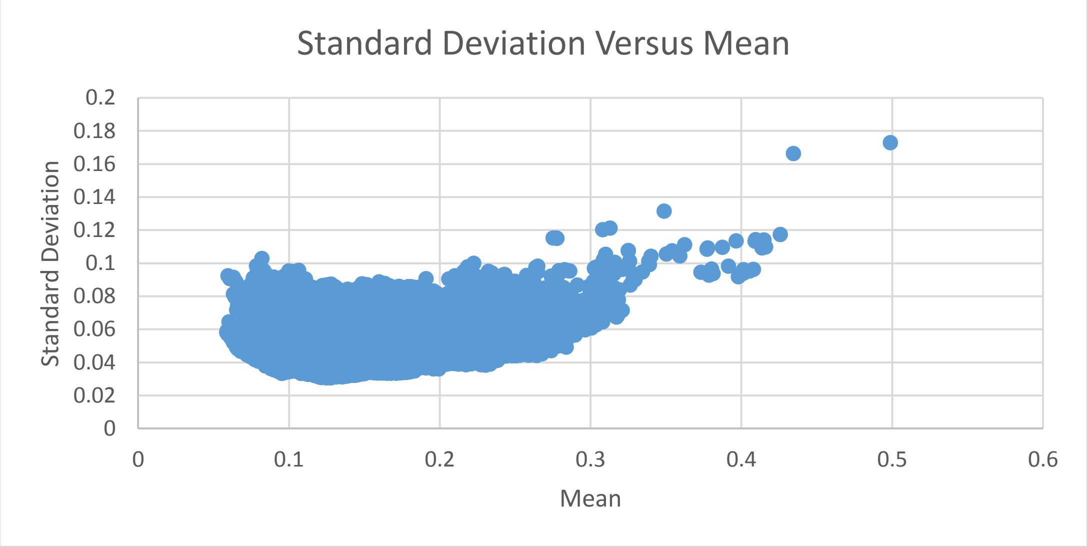


Figure 3 Standard Deviation As a Function of The Mean of the Neural Network Output

Due to the complexity of manually calculating the weekly conditional turn rates, we came up with a simplified way to perform the comparison between the neural network's outputs and the historical turn rates where we defined a variable called manual turn rate as shown below:

$$(Manual\ TurnRate)i = \frac{1}{(Average\ weeks\ on\ Lot)i}$$

Where, average weeks on Lot for configuration *i* is calculated by first grouping all vehicles received on lot belonging to configuration I, then recording how many weeks on lot each one spent, and later averaging those values.

The result will be a vector of 579 items, where each configuration is characterized by one manual turn rate. We refer to this vector to populate the turn rates for every vehicle available on lot in a given week and then compare them to the neural network estimates for that particular week. Definitely the neural network outputs won't perfectly match the manually calculated turn rates because they are conditional to the weekly inventory mix whereas the latter is an averaged value independent of the current week's inventory mix. However, this analysis can be accepted given the fact that the turn rate for a given configuration is supposed to slightly vary as the inventory mix changes. We expect the trend of the predicted versus historical turn rate plot to be as close as possible to a straight line. The graphs below show the various plots for the 20, 60 and 100 iteration model, where the R squared estimator is used to evaluate accuracy.

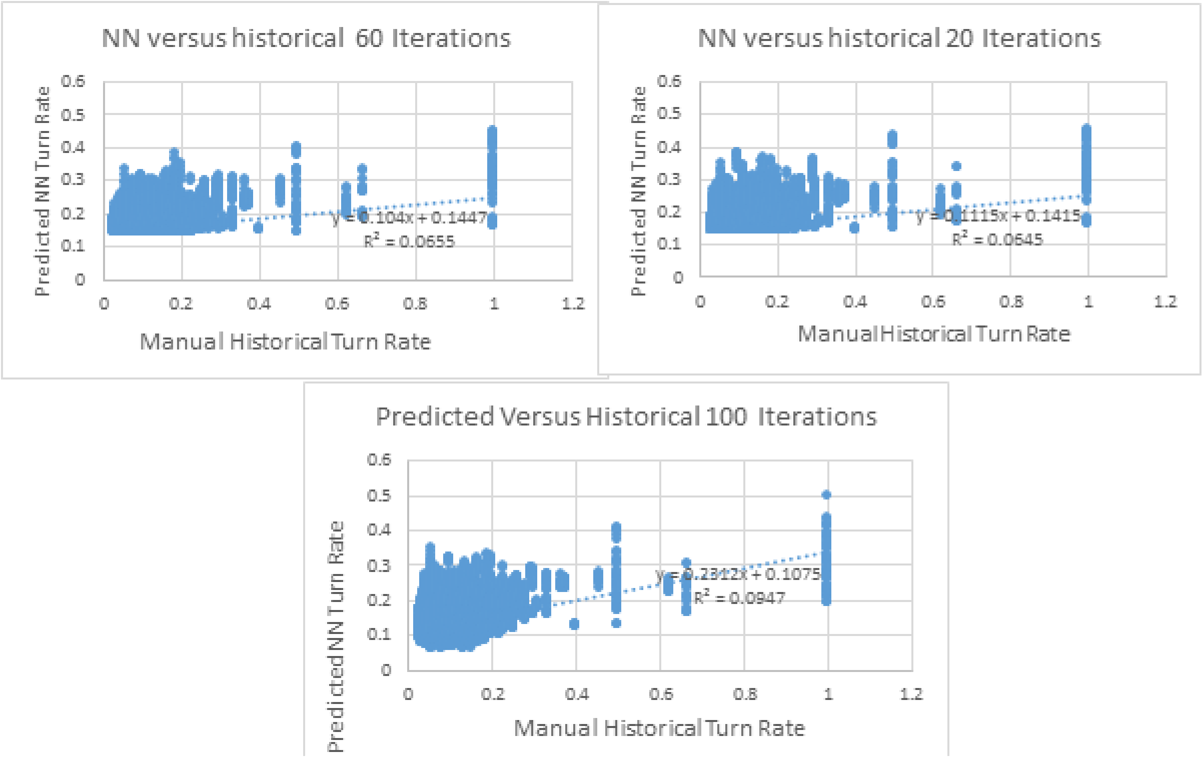


Figure 4 Conditional Regression Neural Network Predictions Versus Manual Historical Turn Rates

Classification Approach

Similar analysis was also applied to the balanced and unbalanced classifier model's outputs and results are plotted below. For the unbalanced model, we only made one run of 60 iterations because our purpose here is not to track the improvement in results as the number of iterations are increased, but to have a benchmark to measure the effect of data balancing.

Balanced

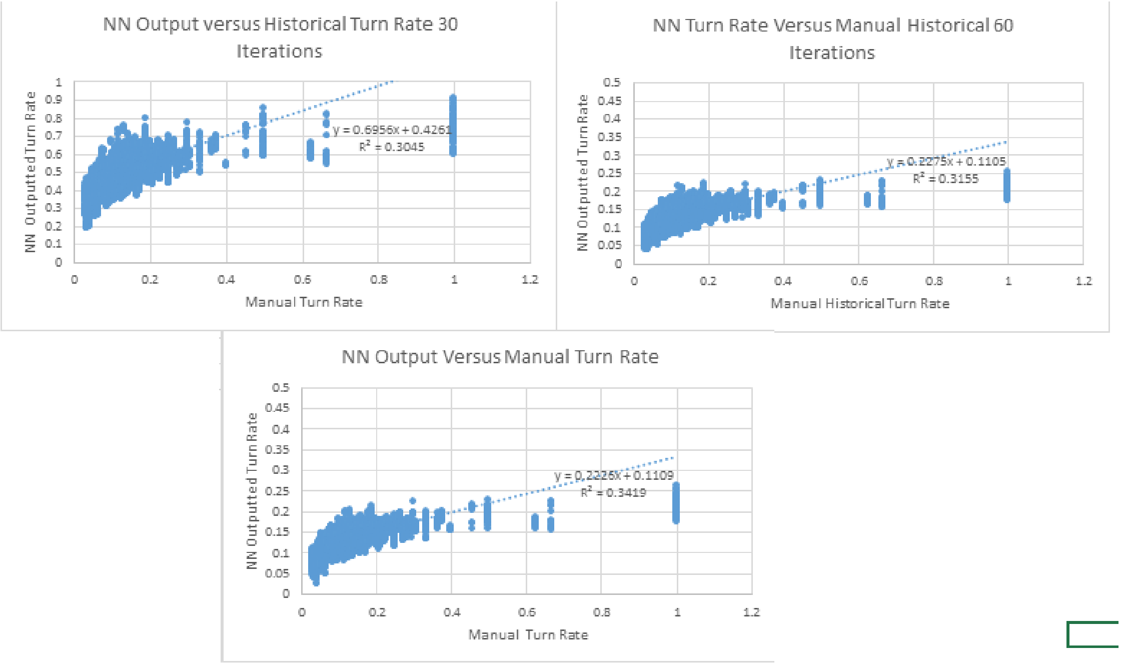


Figure 5 Conditional Classifier Balanced Neural Network Predictions Versus Manual Historical Turn Rates

Unbalanced

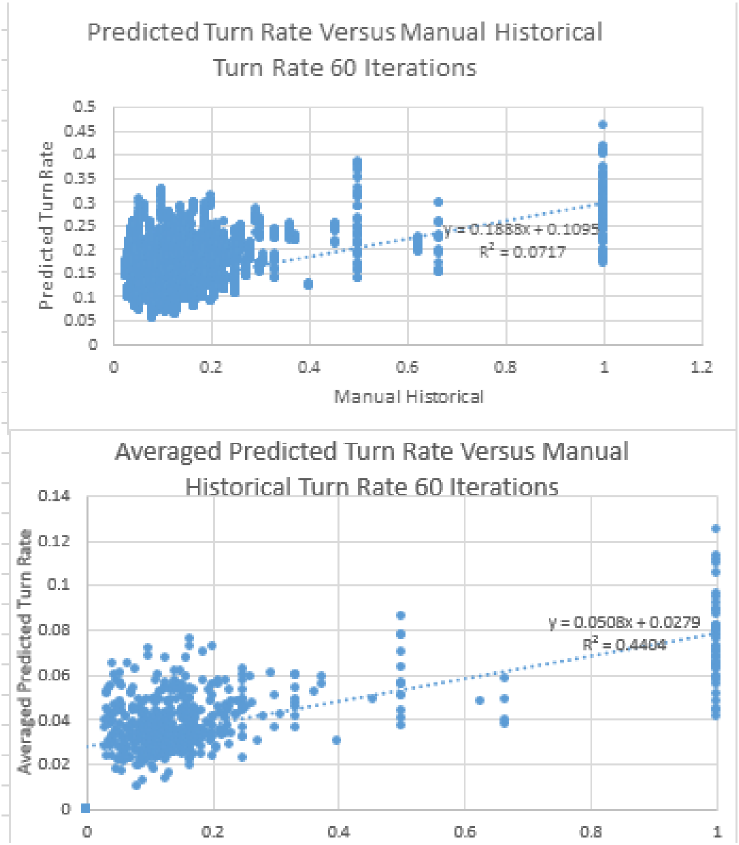


Figure 6 Conditional Classifier Imbalanced Neural Network Predictions Versus Manual Historical Turn Rates

Unconditional Analysis

The main objective of this research is to estimate turn rate for a vehicle configuration, depending on its feature vector. Thus for our output to be useful in real future applications, it should be undocked from the inventory mix that was available at the period when the configuration turn rates were calculated so that it becomes applicable at any given inventory scenario. For this reason, we should convert the first set of neural network outputs which are conditional upon the weekly inventory mix which was available in the historical dataset to an unconditional form where each configuration is characterized with one turn rate independent of the available inventory mix.

Mean as Transformation Function

Averaging the generated conditional turn rates for each configuration over the study period, 37 weeks, and the 17 different market locations is the simplest way to convert it to an unconditional form, given that the single records are characterized by low variability and fall in a small interval around the mean. The plots of the averaged turn rates versus the manual turn rates shown below reveal a high correlation among the two parameters which proves that the neural network's outputs are directionally correct.

Regression Model

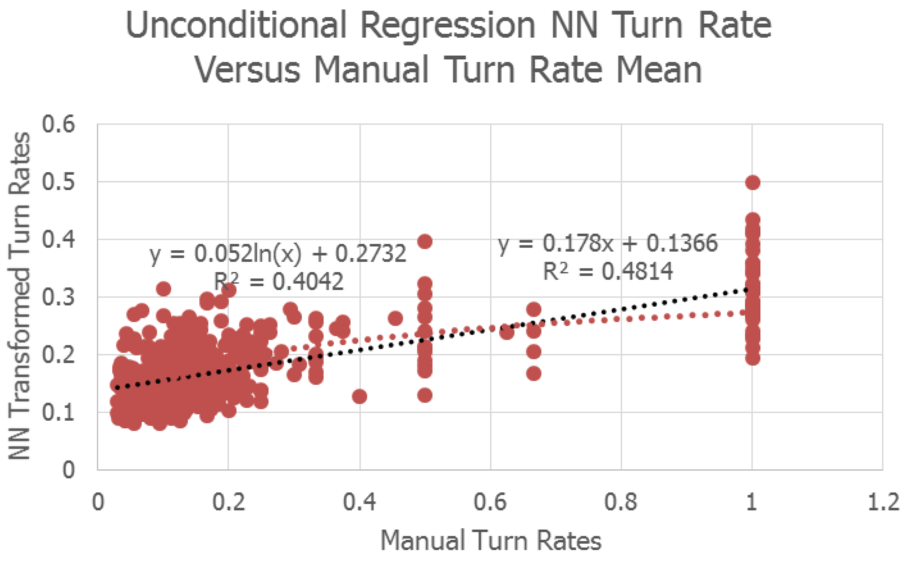


Figure 7 Averaged Turn Rates Versus Manual Historical Turn Rates For Regression Model

Classifier Balanced Model

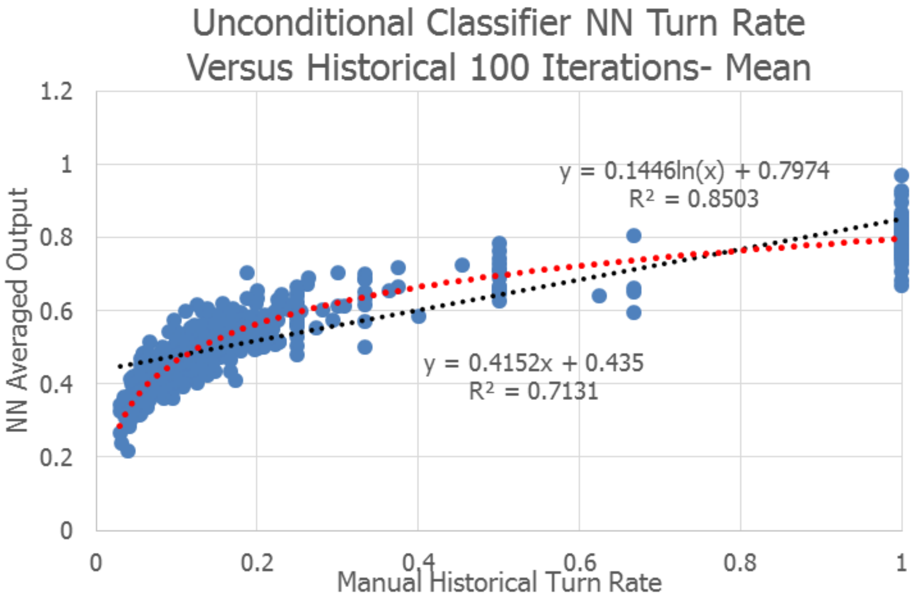


Figure 8 Averaged Turn Rates Versus Manual Historical Turn Rates For Classifier Balanced Model

Discussion

Looking only at the conditional plots in the section above, one can claim that the classifier model over performs the regression model due to the significant difference in the R squared of the linear fit of NN Outputs versus Manual Historical Averaged turn rates, 0.32 versus 0.07. However, after plotting the averaged NN Outputs and plotted it against the manual averaged turn rate, the goodness of the fit improved significantly in both of the models and they converged to a very close level of accuracy 62% versus 71%. This urged us to use different approaches to evaluate the accuracy of those models. On the other hand, as we compare the R squared parameter for the trend lines of the graphs for both forms of classifier model, balanced and unbalanced, we notice that the accuracy of the fit significantly increased from 0.44 to 0.499 for the unconditional averaged turn rates and from 0.0717 to 0.315 for the conditional un-averaged turn rates, as we introduced the data balancing into the classifier model. This signifies the important effect of data balancing on the classifier's accuracy.

To justify the reason why we chose to measure the weighted average error rather than the regular average error, we plotted the relative error associated with each configuration versus its popularity in the dataset. The graph below visualize the relationship between these 2 parameters.

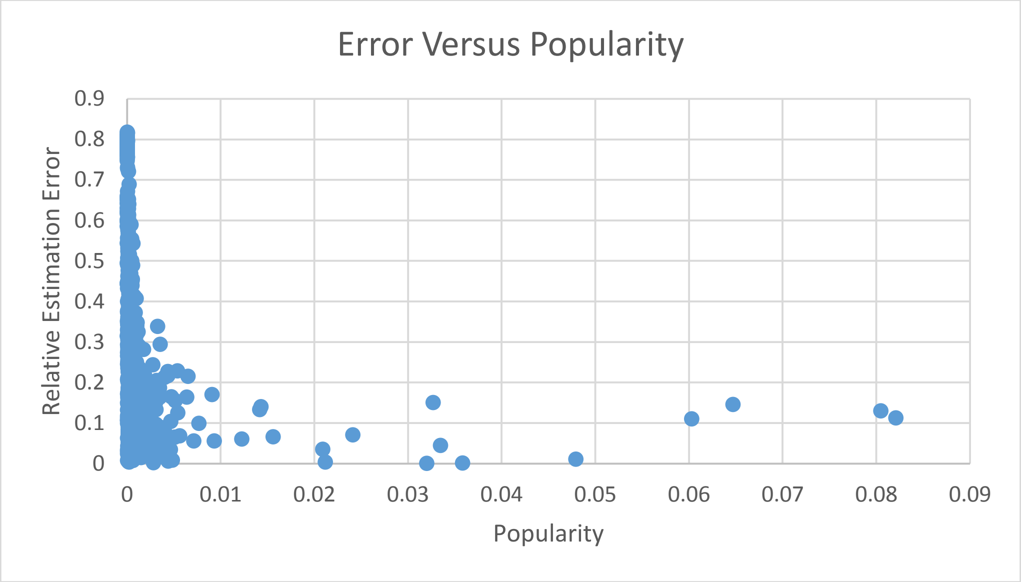


Figure 9 Relative Estimation Error as a Function of Configuration Popularity

It is shown in the graph above that the majority of configuration have a popularity level less than 0.05. This extremely low repetition of those configuration is blocking the ability of the neural network to learn about their sale's behaviors and thus leading to significantly high errors. However, for those more popular configurations with a popularity higher than 0.02, the error level considerably dropped to values below 0.2. As a conclusion, it is extremely important to input those weights in the error estimation function to be able to bias the average error value in favor of the popular configurations. Arriving to this conclusion gives us a positive intuition towards the ability of neural network to predict turn rates, since whenever we feed it with enough measurements for a certain configuration, it is able to predict the correct output with 90% accuracy. Therefore, if we received a better dataset in the future and we applied the same methodology we should arrive to better results.

Risk Adjusted Turn Rate Transformation Function

Another way to perform this conversion is by using the formula below:

$$Yhat = \mu - \alpha \times \sigma$$

Where *Yhat* is the unconditional estimate of the turn rate for configuration X, μ is the mean of all the records associated to configuration X, α is the importance factor given to the variability of the records, and σ is the standard deviation. This parameter has been referred to as the risk adjusted average turn rate by the Ford's Patent.

The privilege of using the risk adjusted turn rate, is the fact that it takes into account the variability of the turn rate estimates and generate an averaged value characterized by low variance. As we notice from the Standard Deviation versus mean scatter plot below, high average turn rates are associated with high σ, whereas low averages are associated with low σ. Therefore, the optimal turn rate value to be picked is somewhere in the middle of this range. The only decision variable in this context is the scaling factor α which should be optimized to generate the most accurate averaged turn rate.

In order to decide on the optimal value of *alfa*, we built an optimization model in excel whose objective function is maximizing the correlation factor between the *Rhat* estimates and the manually calculated historical turn rates. And its subject to boundary constraints that bounds *Rhat* between 0 and 1. Below is the exact formulation of the maximization problem:

Objective Function: Maximize

$$Correl(X,Y) = \frac{\sum (x-\bar{x})(y-\bar{y})}{\sqrt{\sum (x-\bar{x})^2 \sum (y-\bar{y})^2}}$$

Where $\bar{x}\ and\ \bar{y}$ are the sample mean for x and y

St.: Constraints: $0 \leq Yhati \leq 1$ for every I

Alfa is allowed to vary from -∞ $to + \infty$

The results are presented in the table below:

	Regression	Classifier
Alfa Value	-4.7292	-1.30622
Maximum Correlation	0.778726	0.854114
Average Total Error	0.369895	0.185302

Table 4 Risk Adjusted Turn Rate Alfa Optimization Results

$$\text{Average Total Error} = \sum_i \frac{|Ri-Rihat|}{\max(Ri,Rihat)}$$

Where *i* is the set of weeks going from 9 till 34.

The plot below visualize the correlation between the estimated turn rates and the real historical turn rates.

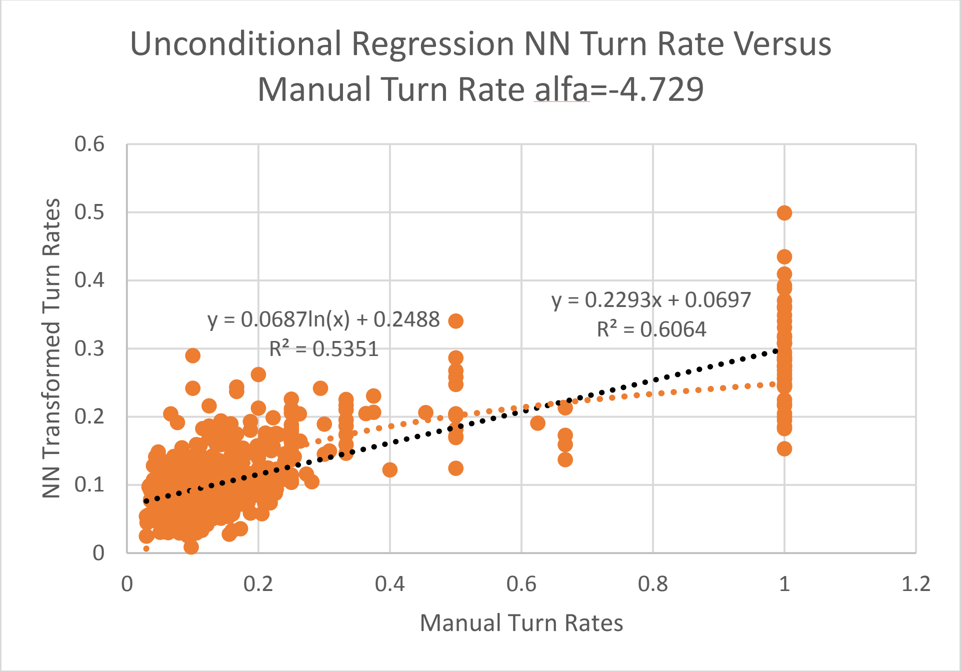


Figure 10Risk Adjusted Turn Rates versus Manual Historical Turn Rates for Regression Model

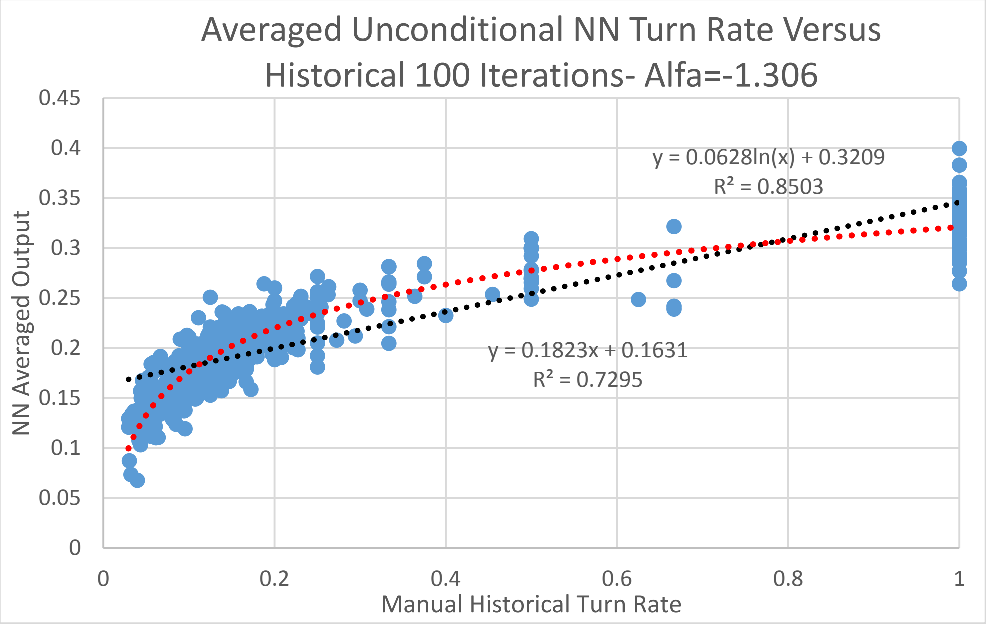


Figure 11Risk Adjusted Turn Rates versus Manual Historical Turn Rates for Classifier Balanced Model

Exponential Transformation Function

From the graphs above, we noticed that a logarithmic fit generated a higher R squared than the linear fit. Inspired by this result, we came up with another transformation function which takes the exponent of the risk adjusted turn rates.

$Yhat = e^{Yhat\ Risk\ Adjusted}$

The graph below shows the improvement in the correlation between the estimated unconditional turn rates and the manual historical averaged turn rates.

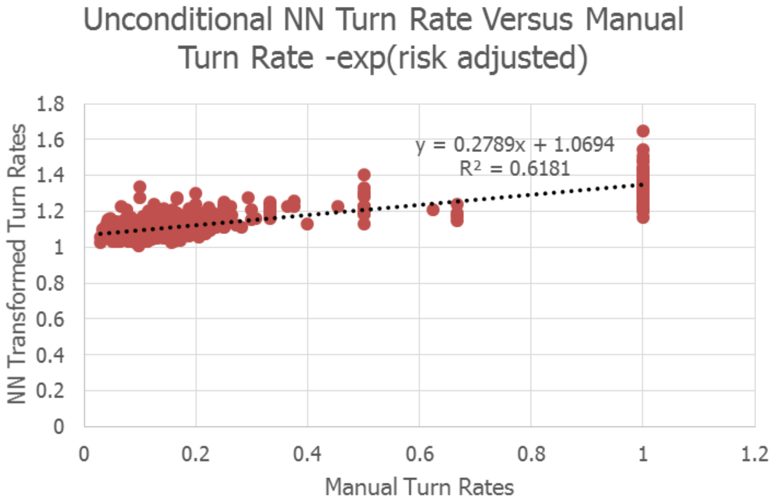


Figure 12 Exponent of Risk Adjusted Turn Rates Versus Manual Historical Turn Rates For Regression Model

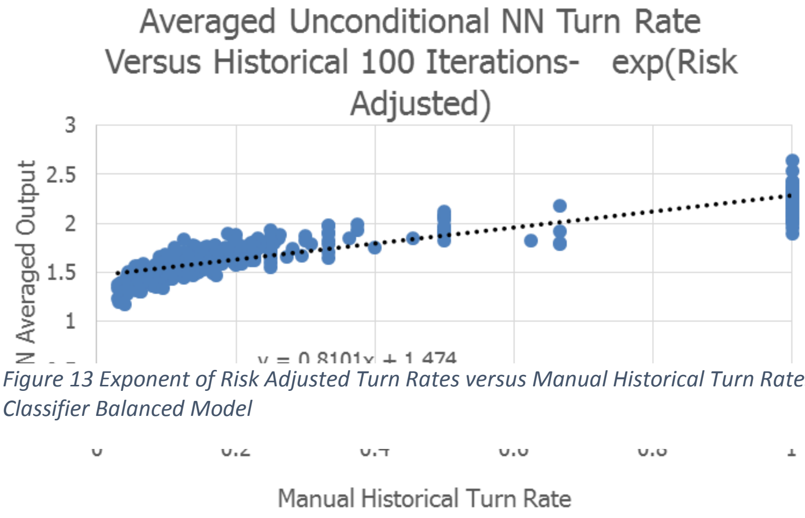


Figure 13 Exponent of Risk Adjusted Turn Rates versus Manual Historical Turn Rates for Classifier Balanced Model

Discussion

To summarize the results above, we noticed that factoring the variability of the turn rate records into the transformation equation, had raised the correlation between the estimated and the real historical turn rates. This proves that the risk adjusted value is in fact an effective representation. Moreover, the further increase in *R squared* resulting from the exponential transformation indicated that the rate of increase in the manual historical turn rate is faster than that of the neural network outputs which makes it more in line with the logarithm of the manual turn rates rather the manual turn rates themselves.

Simulator

For the sake of evaluating the accuracy of this conversion, we built a simulator that generates random customer choices at a given week, allocated based on the estimated unconditional turn rates, *Yhat* of the available configurations, then records the vehicle count for every configuration and compares it to the real historical counts. At this level we are going to simulate the outputs of the three different transformation functions listed above and verify which one is going to behave better under random choice simulation.

For example, if in the real life scenario we had 3 Black cars, 4 Red, and 6 Yellow at a given week, and 2 Blacks, 1 Red were sold we will be left with 1 Black, 3 Red and 6 Yellow at the end of the week. Now the simulator will take this initial inventory (3 Black cars, 4 Red, and 6 Yellow), and generate 3 random choices based on the estimated turn rates of each configuration, which may or may not match the actual sold vehicles. According to the laws of probability, the vehicle with the highest turn rate are more likely to be chosen by the simulator and thus sell faster. Now, if those turn rates were representative of the real life customer preferences, the simulated inventory should converge to the actual one. If for example the simulator suggested 1 Black, 1 Red, and 1 Yellow sold vehicles, we will end up with 2 Blacks, 3 Reds, and 5 Yellow. Now, the weighted average error is calculated as follows:

$$Error = \frac{3}{13} \times \frac{abs(1-2)}{\max(1{,}2)} + \frac{4}{13} \times \frac{abs(3-3)}{\max(3{,}3)} + \frac{6}{13} \times \frac{abs(6-5)}{\max(6{,}5)}$$

The above equation gives a higher weight to the relative errors of those popular vehicles by multiplying it with the ratio of the vehicle count of this given configuration to the total number of vehicles available on lot during that week.

Simulator Description

We only considered the sales data for Boston market for simulation purposes, starting from week 9 through week 43. The first step in the simulator calculates the vehicle counts in the historical dataset for every configuration in every week of the period extending from week 9 till 43 which will act as a benchmark to determine the deviation of the simulated inventories from the actual ones. Afterwards, a vector of weights is generated for every week, representing the ratios of vehicle count for each configuration to the total number of vehicles available on lot on a particular week. These weights will later be multiplied by the relative error of simulated and historical inventory as shown in the equation above.

Second step the actual simulating portion. To initialize the inventory at the beginning of every week, we equated the inventory level at the beginning of every week *t* to the real historical inventory at the end of week *t-1*. This will reset the inventory level at the beginning of every week to the real inventory that was available in the historical data in order to eliminate the propagation of simulation error from one week to another. After initializing the simulator will add the vehicles that arrived at the beginning of week *t* then it will count how many vehicles were sold on that week in order to know how many random numbers to generate. Accordingly, *N* random numbers between 0 and 1 will be generated representing *N* random customers walking into the dealer market and buying a vehicle.

Now to classify those random numbers into actual vehicle choices, we need to generate choice intervals based on the available set of configurations. This step is done by first identifying *M* which is the number of unique configurations available after adding the arrivals on week 10, then pulling the associated unconditional turn rates *Rhat* calculated in the previous step, and later normalizing those *Rhat's* to generate *M* choice sub intervals between 0 and 1 based on which the random numbers are assigned to configuration choices. The example below demonstrate the normalization and choice allocation procedure.

Take the same inventory mix mentioned previously, with the following assigned *Rhats*,

Configuration	Count	Rhat
Black	3	0.32
Red	4	0.2
Yellow	6	0.15

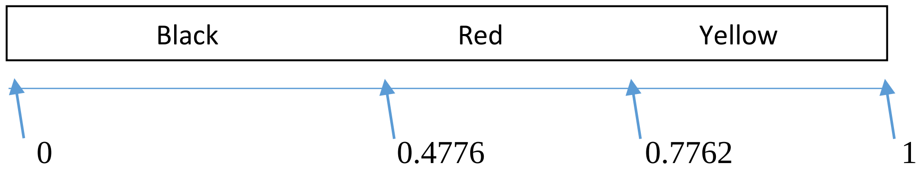
Table 5 Sample Simulator Example

Now to normalize the *Rhat's* we divide each by the sum of *Rhat's* associated with the available configurations,

$$Y1 = \frac{Rhat1}{Rhat1+Rhat2+Rhat3} = \frac{0.32}{0.32+0.2+0.15} = 0.4776;$$

$$Y2 = Y1 + \frac{Rhat2}{Rhat1+Rhat2+Rhat3} = 0.4476 + \frac{0.2}{0.32+0.2+0.15} = 0.7762;$$

$$Y3 = Y2 + \frac{Rhat3}{Rhat1+Rhat2+Rhat3} = 0.7762 + \frac{0.15}{0.32+0.2+0.15} = 1;$$



Now if the generated number lies between 0 and 0.4776 it will be assigned to the black configuration, if it was between 0.4776 and 0.7762 it will be assigned to the Red configuration, and Yellow otherwise. Notice that the biggest interval refers to the vehicle with the highest *Rhat*. Suppose that the generated random numbers were 0.2, 0.5, and 0.8, then the associated 3 random customers choices will be as follows: Black, Red, and Yellow.

After specifying these random choices, the inventory counts will be adjusted accordingly by subtracting the chosen vehicles from the current inventory to generate *I1*, which will be later inputted as an initial inventory to the next iteration at week 11. The same steps are then repeated for every week till week 43, and the vehicle counts are recorded at each week. In order to minimize the effect of randomness and noise, we run this simulator for 100, 500, 1000, 5000, and 10000 iterations and recorded the average of vehicle counts across all the iteration. Then the weighted relative errors between the averaged simulated vehicle counts and the historical counts, is measured and projected in order to evaluate the accuracy of *Rhat* estimates used. The flow diagram explains the steps followed for this purpose.

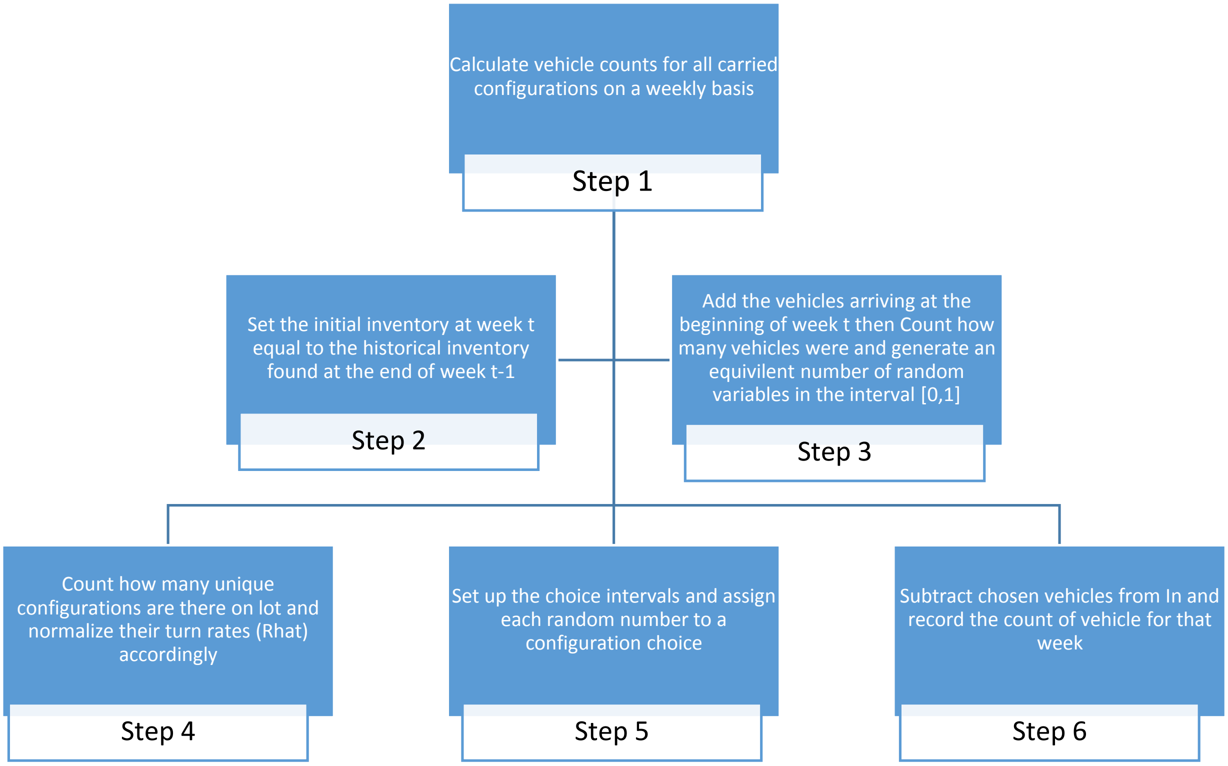

Calculate vehicle counts for all carried configurations on a weekly basis
Step 1
Set the initial inventory at week t equal to the historical inventory found at the end of week t-1
Step 2
Add the vehicles arriving at the beginning of week t then Count how many vehicles were and generate an equivilent number of random variables in the interval [0,1]
Step 3
Count how many unique configurations are there on lot and normalize their turn rates (Rhat) accordingly
Step 4
Set up the choice intervals and assign each random number to a configuration choice
Step 5
Subtract chosen vehicles from In and record the count of vehicle for that week
Step 6

Figure 14 Flow Chart for Count Simulator

Three different error functions were used to evaluate the output of the simulators:

- EWW:
  - ○ Weighted Relative Error=$\sum_i wi * \frac{|Ci-Chati|}{\max(Ci,Cihat)}$ where wi is the ratio of number of vehicles belonging to configuration I relative to the global number of vehicles availble in the market
- EWR
  - ○ Relative Error=$\sum_i \frac{|Ci-Chati|}{\max(Ci,Cihat)}$
- EWC:
  - ○ Error in Vehicle Count=$|Ci - Chati|$

Results Summary and Discussion

Classifier:

	EWW	EWR	EWC
Mean	0.45	0.388	2.812
Risk Adjusted	0.44	0.378	2.809
Exponential	0.43	0.376	2.806

Table 6 Count Simulator Results When Simulating The Classifier Turn Rates

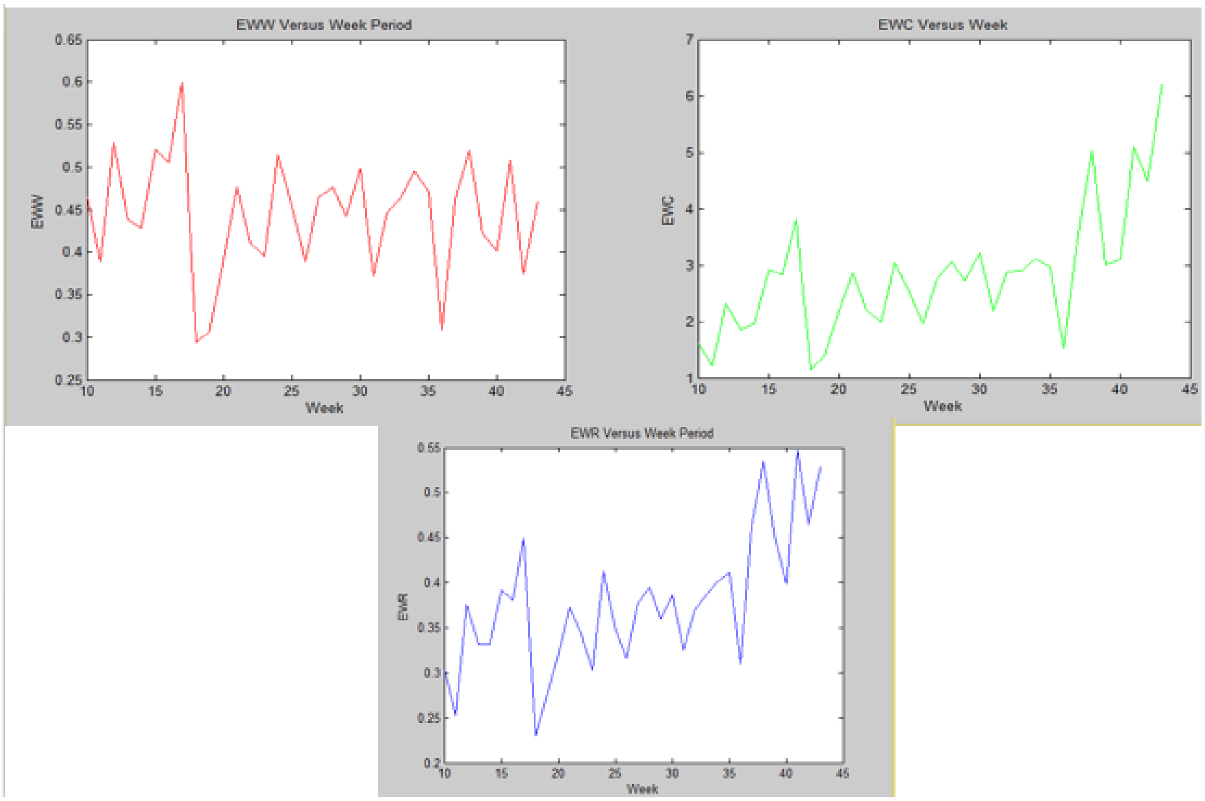


Figure 15 EWW, EWR, and EWC Variation as a Function of Week Period For Classifier Simulation

Regression:

	EWW	EWR	EWC
Mean	0.24	0.32	1.22
Risk Adjusted	0.23	0.30	1.21
Exponential	0.22	0.29	1.21

Table 7Count Simulator Results When Simulating The Regression Turn Rates

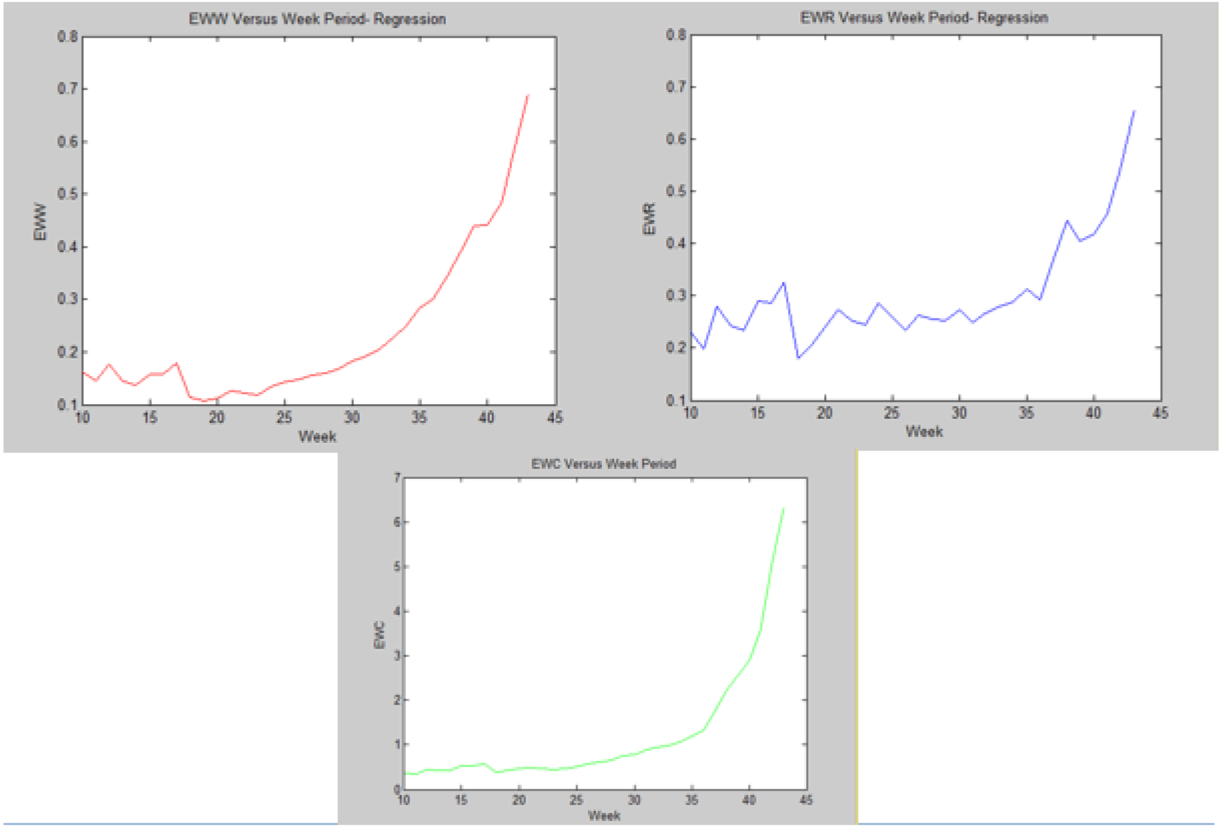


Figure 16EWW, EWR, and EWC Variation as a Function of Week Period For Regression Simulation

Sales-Based Simulator

In this model, our measuring criterion is the count of sold vehicles of each configuration on a given week, instead on inventory counts. In further details, we start the first step by recording how many vehicle were sold of each configuration on weeks 9 through 43 in real historical dataset which will act as a benchmark to determine the deviation of the simulated inventories from the actual ones. Afterwards, a vector of weights is generated for every week. This stage requires several steps, starting by summing up the total number of vehicles sold for every configuration over the whole time horizon Si. Then we will filter out the inventory of every week and identify the unique configurations available on that week. Finally, we calculate the week specific weights as follows:

$Wit = \frac{Si}{\sum Sj}$ Where i is one of the configurations available on lot, and the set of j's is the set of configuration available on week t.

These weights will later be multiplied by the relative error of simulated and historical inventory to attain a single scalar value representing the deviation at a given week.

The third step of the simulator, is the actual customer choice simulation step which is very similar to the simulation step explained in the model above. To count the number of sold vehicles, we define a vector Vcount of zero values assigned to each vehicle-week record from week 9 till 43. Then every time a choice is randomly picked we increment the Vcount element corresponding to that particular vehicle-week record by one. Finally, the model is evaluated via several error functions which are listed below:

1- |Si-Sihat|/∑Sij
2- Wij*|Si-Sihat|/∑Sij
3- Wij*|Si-Sihat|/max (Si, Sihat)

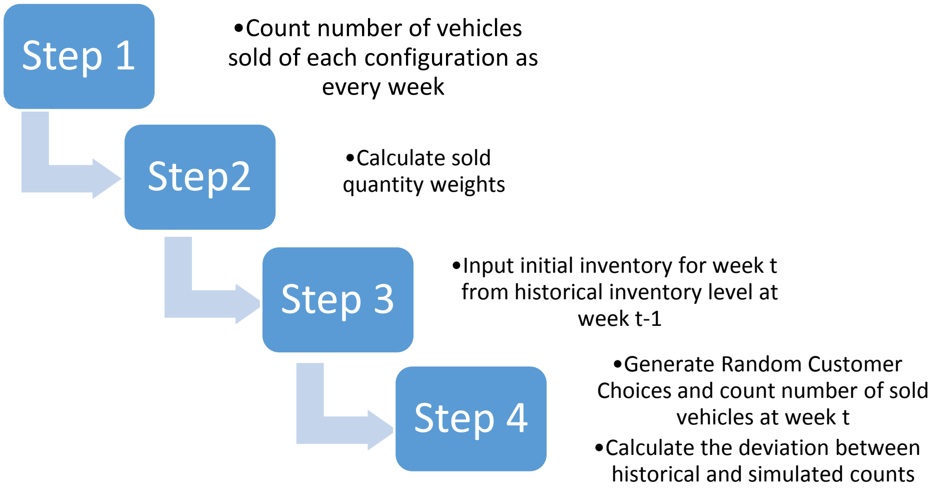

Step 1
•Count number of vehicles sold of each configuration as every week
Step2
•Calculate sold quantity weights
Step 3
•Input initial inventory for week t from historical inventory level at week t-1
Step 4
•Generate Random Customer Choices and count number of sold vehicles at week t
•Calculate the deviation between historical and simulated counts

Figure 17 Flow Chart for Sales Simulator

Results of this simulator turned out to be highly in accurate with an average error of 87% for both the classifier and regression regardless to the transformation method used. The reason behind this extreme results lies behind the high level of randomness in predicting the customer choices. As we noticed the selling incidence of a given vehicle is an extremely low probability occurrence, where the majority of the configuration has a turn rate below 0.07. Counting the vehicle available on lot at the end of the week rather than the number of vehicles sold during the week for each configuration had yield to lower levels of errors since |*Ci*-*Cihat*| is relatively small compared to *Ci* or *Cihat*, whereas |*Si*-*Sihat*| is very close to both *Si* and *Sihat*. Therefore, we ignored the results of the later.

Discussion

Comparing the three error estimates generated by the classifier and regression models in the count simulator, we noticed that the Regression was superior to the classifier when simulated under a random customer choice behavior. *EWW* was 23% in the regression as compared to 45% in the classifier. As for *EWR* and *EWC* they were 0.32 and 1.22 in the regression as compared to 0.38 and 2.88 in the later. For that reason, we believe that data balancing that has been done in the classifier model has helped in improving the correlation between the estimated turn rates and manual historical turn rates, and improving the precision of these estimates by reducing their variance. However, it led to a deviation in the accuracy of the output which explains why the regression model behaved better in the simulator.

CONCLUSION

Applying the mix rate modulated transformation enabled us to translate the inventory information from the configuration level to the feature level, which is the key to studying the feature effect on the probability of sales of a given configuration. Even though both of our models, regression and classifier haven’t exceeded a 70% accuracy in predicting probability of sales, they were directionally correct in terms of better predicting more popular configurations, and giving higher turn rates for those configurations who actually have fast selling rates. Similarly, giving low turn rate values for slow selling configurations. Another good point about both models, is the relatively low coefficient of variation, which means that the neural network is capable of correlating vehicles belonging to the same configurations regardless to their mix rate modulated transformation, by assigning them turn rates that falls into a small confidence interval. With a better dataset, that is less sparse and extends to a larger period, the neural network is expected to reach higher levels of accuracy. In addition to optimizing the processing, training, and performance functions used in the neural network might also help boosting the accuracy of the model.

The simulator on the other hand, provides an insight about the accuracy of the different transformation methods applied to both models’ outputs. And it proved that even though the classifier model yielded a better correlation with historical values, the regression model generated more accurate outputs which yielded to relatively low error value when simulated. More effort should be put into improving the conditional turn rates’ estimation methodology either through optimizing the neural network models or through applying different estimation methods. Moreover, the transformation functions should be further improved by creating a close loop feedback optimization model that minimizes the error of the simulator by varying the transformation function.

Our future goals extend to the strategic planning level by creating a tool to help the company decide on the optimal complexity of its assortments. This objective is attained by determining the utility associated with every standalone variant of a feature, and eliminating the low utility variants after studying the customer substitution behavior in the absence of those variants. Moreover, knowing those individual feature utilities will give the company the privilege of predicting the probability of sales of any configuration based on its feature constituents even if it was never offered in the assortment before. Branching out from this paper, those utility factors can be computed by equating the utility score of a given configuration to its unconditional turn rate calculated above. As a definition, the utility score of a given configuration is the linear or nonlinear combination of the dot products of a set of weights, or utilities, corresponding to the explanatory variables, features, constituting the given configuration. For example, if an MNL model is to be utilized, the corresponding score for configuration i is $(e^{\sum \beta i X i})i$. Then an error minimization optimization model is to be formulated to compute the values of the β's. Below is a possible formulation of this optimization model:

$$\text{Objective Function: Minimize } \sum_i \frac{(e^{\sum \beta i X i})1}{\sum_j (e^{\sum \beta i X i})l+U0} \; - Rhati$$

Where i is any configuration in any given assortment and j is the set of all configurations offered in that assortment

Constraint: $\beta i's$ are non-negative

Xi vector for each configuration ` is given

$U0$ is the walk away probability.

As we mentioned in the introduction above, the primary purpose of this thesis is to feed in the value of the β parameters in a more sophisticated assortment planning model that will optimize the level of complexity in a way that maximizes the company’s profits.

APPENDIX A

The Table Below show the list of regions considered in our model.

Location Number	Location Name
1	Boston
2	New York
3	Philadelphia
4	Pittsburgh
5	Memphis
6	Orlando
7	Atlanta
8	Washington
9	Chicago
10	Twin Cities
11	Detroit
12	Cincinnati
13	California
14	Denver
15	Northwest
16	Kansas City
17	Southwest

Figure 18 List of Markets Considered

APPENDIX B

Data Processing and Coding Procedure

Data Preprocessing

The Sequence for Running Matlab codes:

We extract the following columns from any data set received

Arrival Date	Sales Date	Dealer Location	Set of Features being considered

We first convert the dates from days to weekly periods. In other words, if our dataset starts from January 1st, we count this to be our origin in the timeline. Then if a vehicle was received on March 1st, we can say the vehicle was received on week 9 of the time horizon. Similarly, for the sales dates.

Second, we assign a number indicator for each location to be able to input it to Matlab as an integer.

Third, we calculate week on lot for each vehicle by subtracting Sales Week from Arrival Week

Finally, we modify the feature vector by adding all possible variants of a given feature, even if one of the variants was a no-feature option. For example, the "Heated Seats" feature can come in the following varients:

1- Front Heated Seats
2- Front and Rear Heated Seats
3- Part of the Climate Package
4- No heated Seats

Thus we must make sure that all of those variants are mentioned as separate columns in the configuration definition of the dataset in order to cover all possible configurations in the assortment. Refer to excel file "Simplified Configurations" for better understanding.

Now we will attain the following columns which will be inputted into the first Matlab Code:

Arrival Week	Sales Week	Weeks on Lot	Dealer Location Indicator	Set of Modified Features being considered

Code 1: Generate the Mix Rate Modulated Weekly Inventories

The code below will take the above variables and will output the mix modulated feature vector for every week's inventory

```
clc
clear all

myfile='simplifiedconfigurations.xls';
sheet=2;
xlRange='T3:AT31104';% range of features
xlRange1='E3:E31104';% range for arrival week
xlRange2='G3:G31104';% range for sold week
xlRange3='Q3:Q31104';% range for location indicator
INV=xlsread(myfile, sheet, xlRange); % inventory data with features
A=xlsread(myfile, sheet, xlRange1);% arrival week vector
S=xlsread(myfile, sheet, xlRange2);%sold week vector
L=xlsread(myfile, sheet, xlRange3);%Location Indicator vector
X=size(INV,1); % no of records
period=1;% this is the week increments at which we are grouping the market’s inventories
Global=[];% Matrix containing all sales records for all regions
Config=unique(INV,'rows');
%set of available configurations this vector will assign an indicator for each configuration available in the assortment in order for us to track how this configuration behaves differently as it appears in different inventory scenarios
Configuration=[]% Configuration pointer
for G=1:X
[~,indx]=ismember(INV(G,:),Config,'rows');
Configuration(G)=indx;
End

INV1=[L,Configuration',A,S,INV];
% We Add the Configuration indicator to the matrix.

for i=1:17 % Different Locations
temp=[];
INV2=[];% The Matrix that contain sales record for a specific region
for x=1:X
if(INV1(x,1)==i)
INV2=[INV2;INV1(x,:)];
end
end
Lamda=size(INV2,1);% Size of the regional INVENTORY
Vehicle=[1:Lamda];% Vehicle Identification Number to track each vehicle as it moves from one week to another
INV2=[Vehicle',INV2];
Totalweeks=max(INV2(:,4));
% Last week being studied in the Market region i
Y=size(INV2,2); % number of columns
% Now we need to categorize the inventory as of what is available on lot on each week starting week 9
```

```
for n=9:period:(Totalweeks)
Vector=[]; %V
SoldStatus=[];% This vector is a binary indicator =1 if the vehicle got sold on that week or =0 if
not sold
CurrentWeek=[];
WeeksonLot=[];%Average days on Lot
M=[];% This matrix lists all the vehicles available on week n
D=1;% Index increments for Soldstatus

for j=1:Lamda
if (INV2(j,4)<n+period) && ((INV2(j,5)>=n) || (INV2(j,5)==0)) % if the vehicle has arrived on
or before this week and sold on or later than this week (i.e. if vehicle is/was on the lot this week)
M=[M;INV2(j,:)];

if (INV2(j,5)<n+period) && (INV2(j,5)>=n) % if vehicle is sold in the current period/week
SoldStatus(D)=1;
else
SoldStatus(D)=0;
end
CurrentWeek(D)=n;
WeeksonLot(D)=abs(INV2(j,4)-n)/period;
D=D+1;

else
continue
end

end
% The steps below are applying the mix rate modulated technique explained in the thesis
SizeM=size(M,1);
SumM=sum(M); % total feature inventory
Mixrate=SumM/SizeM; % average of each feature in the inventory
InputData=M;% The matrix that contains the sales record over all the weeks in a given location
for k=6:Y
InputData(:,k)=InputData(:,k)-Mixrate(k);
end
InputMatrix=horzcat(CurrentWeek',WeeksonLot',InputData,SoldStatus');
temp=[temp;InputMatrix];
end

[GlobalRow,GlobalCol]=size(temp);
Recursive=1; % to determine which row of the Inputtemp matrix is to be evaluated

% The Steps below are to remove the censored data
```

```
temp(find(temp(:,7)==0),:)=[];% filter out all rows that have a sold week =0 which means they weren’t sold within the period of study
temp(find(temp(:,7)>(Totalweeks)),:)=[];
Sizetemp=size(temp,1); % determine how many records are left after removing the sensored data

Global=[Global;temp]; % Compile weekly inventories for every week in the time horizon
End

% The Steps Below are only applied if data balancing is required
SizeGlobal=size(Global,1); %number of rows in Global Matrix
for L=1:SizeGlobal
if(Global(L,end)==1)
Replicate=repmat(Global(L,:),6,1);
Global = [Global;Replicate];
else
continue
end
end
SizeGlobal=size(Global,1); %number of rows in Global Matrix
filename = 'Neural Network Input Global.xlsx';
xlswrite(filename,Global,1)
Input=[Global(:,2),Global(:,[8:end-1])]; % This matrix is the input to train the neural network
Output=Global(:,end); % This is the set of target values needed for training the neural network
Location=Global(:,4);
Configset=Global(:,5);
AvgweeksonLot=Global(:,2);
V=Global(:,3);
save('Records.mat','Location','Global','INV1','Input','Output','Configset')
```

Code2: Weight Calculation

For the reasons specified in the thesis, we need to input weights into the penalty function of the neural network training code in order to give more importance to more popular configurations and less importance to the rare ones.

```
clc
clear all
load('Records.mat','Global','Configset')

 % this will load the Global matrix generated by the previous code and the row listing all the configurations available on every week.

AvgweeksonLot=Global(:,2);
V=Global(:,3); % Vehicle Indicator variable
Location=Global(:,4);
Configset=Global(:,5);
Output=Global(:,end);% Sold Status
```

```
Matrix=[Configset, V, Output,AvgweeksonLot,Location]; % this matrix contain the associated configuraion number and vehicle number and sold status for each record in the dataset
E=size(Matrix,1);
TotalConfig= max(Matrix(:,1)); % Total number of configurations available in the dataset
Popularity=zeros(TotalConfig,1); % Number of vehicles available from each configuration
for f=1:TotalConfig % Chooses a given configuration
    Group=[]; % The matrix that will group all records of the same configuration together

    for g=1:E
        if Matrix(g,1)==f
            Group=[Group;Matrix(g,:)];
        end
    end
    H=size(Group,1);% Number of vehicles available of each configuration
    Popularity(f,1)=H;
end
for i=1:size(Popularity,1)
    if Popularity(i)==0
        Popularity(i)=Popularity(i);
    else
        Popularity(i)=1/Popularity(i);
    end
end

Con=[1:TotalConfig];
Popularity=[Con',Popularity];

ew=zeros(size(Configset,1),1);
for j=1:size(Configset,1)
    ew(j,1)=Popularity(find(Popularity(:,1)==Configset(j)),2);
end

save('weights.mat','Popularity', 'ew')
```

Code3: Neural Network Script

This is the generic Newff script with some adjustment

```
clc
clear all
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created Wed Apr 08 22:40:15 EDT 2015
%
% This script assumes these variables are defined:
%
```

```
%   Input - input data.
%   Output - target data.
load('Records.mat','Input','Output')
load('weights.mat','Popularity','ew');
OutputVector1=zeros(size(Input,1),30);% output as probability
OutputVector2=zeros(size(Input,1),30);% output as a categorical variable
%for i=1:40
train_inp= Input(:,[2:end]);
% %standardise the data to mean=0 and standard deviation=1
% %inputs
% mu_inp = mean(train_inp);
% sigma_inp = std(train_inp);
% for i=1:size(train_inp,2)
% train_inp(:,i) = (train_inp(:,i) - mu_inp(1,i) )/ sigma_inp(1,i);
%end
x = train_inp';
t = Output';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. NFTOOL falls back to this in low memory situations.
%trainFcn = 'trainlm';  % Levenberg-Marquardt

for i=1:100
%% NEW CODE
net=newff(minmax(x), [15,1],{'tansig','purelin'},'trainlm');

%net = init(net); % For Repeating Initialization - Note that newff Performs Initialization as well!
The transig, purelin and trainlm are the training functions of inputs and outputs
% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.inputs{1}.processFcns = {'removeconstantrows','mapstd'};
% The mapstd will normalize the inputs and outputs such that their mean=0 and std=1
net.outputs{2}.processFcns = {'removeconstantrows','mapstd'};
net.trainParam.show = 50;
net.trainParam.lr = 0.001;
net.trainParam.epochs = 30;
net.trainParam.goal = 1e-5;

%% END OF NEW CODE

% Create a Fitting Network
% hiddenLayerSize = 20;
% net = fitnet(hiddenLayerSize,trainFcn);
% % Choose Input and Output Pre/Post-Processing Functions
% % For a list of all processing functions type: help nnprocess
% net.input.processFcns = {'removeconstantrows','mapminmax'};
% net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean squared error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
  'plotregression', 'plotfit'};

% Now for the classifier model we need to convert the continuous variable into a classified
variable based on a threshold which is specified to be 0.5 for a balanced dataset and 0.14 for an
unbalanced dataset.

T=0.5;
% Train the Network
[net,tr] = train(net,x,t,[],[],ew');
%[net,tr]=train(net,ptr,ttr,[],[],val,test);
% Test the Network
y = net(x);
ytst = round(y-T+0.5);% This is the classified version of the outputs
e = gsubtract(t,ytst);
performance = perform(net,t,ytst); % the performance function here will minimize the difference
between ytst which is the classified output and the target values which are also classifier
variables

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t  .* tr.valMask{1};
testTargets = t  .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)
```

```
% View the Network
%view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, plotfit(net,x,t)
%figure, plotregression(t,y)
%figure, ploterrhist(e)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
if (false)
  % Generate MATLAB function for neural network for application deployment
  % in MATLAB scripts or with MATLAB Compiler and Builder tools, or simply
  % to examine the calculations your trained neural network performs.
  genFunction(net,'myNeuralNetworkFunction');
  y = myNeuralNetworkFunction(x);
end
if (false)
  % Generate a matrix-only MATLAB function for neural network code
  % generation with MATLAB Coder tools.
  genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
  y = myNeuralNetworkFunction(x);
end
if (false)
  % Generate a Simulink diagram for simulation or deployment with.
  % Simulink Coder tools.
  gensim(net);
end
 OutputVector1(:,i)=y;
 OutputVector2(:,i)=ytst;
 i
end
Mean=mean(OutputVector1');

save('Classifier1.mat','OutputVector1','OutputVector2','Mean')
```

Code 4: Unconditional Turn Rates

This code is used to convert the Neural Network’s output to unconditional by applying any the risk adjusted Turn Rate transformation function mentioned in the thesis

```
clc
clear all

% We first input the Global Matrix Outputted from the code1
load('Recordsunbalanced.mat','Global')

% Then we input the turn rates generated by the neural network Either classifier or regression
code3

load('Classifier1','Mean'); % conditional turn rates outputed from NN
Location=Global(:,4);
Configset=Global(:,5);
TR=[Location,Configset, Mean];% A matrix showing the location, configuration, and turn rate
estimate for each record
E=size(TR,1);% Number of rows in matrix TR
TotalConfig= max(TR(:,2)); % Total number of configurations available in the dataset
alfa=2;% Penalty for variability
Set=[]; % Unconditional turn rate estimates for every configuration
for f=1:TotalConfig
   Group=[]; % The matrix that will group all records of the same configuration together
   for g=1:E
      if TR(g,2)==f
         Group=[Group;TR(g,:)];
      end
   end
   if size(Group,1)==0 % this will capture all the configurations that didn’t appear in the final
matrix after removing censored data
      Nu=0;
      Sigma=0;
   else
   Nu= mean(Group(:,3)); % Average all the turn rates for all the vehicles belonging to
configuration f and calculate their standard deviation
   Sigma=std(Group(:,3));
   end
   Rh=Nu-alfa*Sigma; % This is the risk adjusted turn rate
   Set=[Set;[f,Nu,Sigma,Rh]]; % Matrix showing config number, turn rate, mean and std for
every configuration
End
% Below are vectors that combine all the averaged turn rates, standard deviations, and Rhats for
all configurations
mean=Set(:,2); % The average of all turn rate estimates for a given configuration
Std=Set(:,3);% The standard deviation for all turn rate estimates for a given configuration
Configurationset=Set(:,1);% List of configurations
Rhat=Set(:,4);% List of unconditional turnrates
```

```
save('Rhatfileclassifier1.mat','mean','Std','Configurationset','Rhat')
scatter(Set(:,2),Set(:,3))
size(Set,1)

Ans=mean-0.2*Std;
```

Code5: Simulator

Since we defined two criteria to evaluate the simulator results, we have two simulator codes, one that track the count of inventory for each week and calculate the error as: |CountHisotrical-CountSimulator|/max (CountHistorical, CountSimulator). Whereas the second, keeps track of how many vehicles were sold of each configuration is recorded every week and the error is then calculated as: |SoldHisotrical-SoldSimulator|/max (SoldHistorical, SoldSimulator)

A- Simulator Inventory Count based

```
clc
clear all
load('Records.mat','INV1','Global')
Cweek=Global(:,1); % current week
Location=Global(:,4);
Configset=Global(:,5);
Numberofconfig=max(Configset);% gives total number of available configurations
History=[Cweek,Configset,Location];% Real historical inventory present each week
History=History(find(History(:,3)==1),:);% Filter those for location 1=Boston
TotalReal=[];% The matrix that include the counts of all configurations on every week period
weight=[];
for d=10:43 % This for loop will filter for every week and count how many vehicles of each configuration there is
   Real=[];% Matrix that group the records at week d
   Real=History(find(History(:,1)==d),:);
   CountReal=unique(Real(:,2));% find out how many different configurations are present on week d
   SizeCountReal=size(CountReal,1);
   weeksReal=d*ones(SizeCountReal,1); % assign current week
   CountReal=[weeksReal,CountReal];
   Vcountreal=[]; % the vector for vehicle counts
   for c=1:SizeCountReal % count how many vehicles of configuration c are present on week d
      Vcountreal=[Vcountreal;sum(Real(:,2)==CountReal(c,2))];
   end
   CountReal=[CountReal,Vcountreal]; % Now this matrix shows the current week, configuration number, count of this configuration in this given week
   for CC=1:Numberofconfig % this for loop will add all missing configurations and will associated a number zero for the count
      Find=CountReal(find(CountReal(:,2)==CC),:);
```

```
        if size(Find,1)==0
            CountReal=[CountReal;[d CC 0]];
        end
    end
    [values, order] = sort(CountReal(:,2));
    sortedCountReal = CountReal(order,:); % Sort the matrix in increasing order of configuration
number

% Now we wont to define a weight vector that will give higher importance for more popular
vehicles when it comes to calculating the average weighted error in week d.
     sumcount=sum(sortedCountReal(:,3));% sum of vehicles available on week d
    weightweek=sortedCountReal(:,3)/sumcount;% fraction of vehicles of configuration CC out of
the total number of vehicles available
    weight=[weight;weightweek];
    TotalReal=[TotalReal;sortedCountReal];
end
TotalReal=[TotalReal,weight];


% INV1 is the matrix that contains all records for all regions over the
% whole time horizon without classifying them into weekly inventories as in Global Matrix. In
other words, each vehicle is only mentioned once in INV1 regardless to how many weeks it
stayed on lot %INV1=[[L,Configuration',A,S,INV];
load('Rhatfileclassifier1.mat','Configurationset','Rhat');% Here we loaded all available
configurations with their associated unconditional turn rates Rhat
X=size(INV1,1);
Standard=[Configurationset,Rhat];% A matrix that shows all the configuration with their
associated unconditional turn rate
INV2=[];% The Matrix that contain sales record for region 1= Boston
    for x=1:X
        if(INV1(x,1)==1)
            INV2=[INV2;INV1(x,:)];
        end
    end
Lamda=size(INV2,1);% Size of the regional INVENTORY
VehicleCount=TotalReal(:,2); % save vehicle count for every iteration with the first column
listing the configuration number

% Now the actual simulation starts and its embedded in a loop of 100 iterations to average its
output over all the iterations
for sim=1:1000 % Repeat Simulator loop
    SimulatedINV=[]; % This matrix is to track the simulated inventory changes at each week
over the whole time horizon
%Simulation starts here
 for n=10:43
    Initial=[];
```

```
   Currentweek=[];
   for j=1:Lamda % loop that generates the initial inventory on week n
      if (INV2(j,3)<=n) && ((INV2(j,4)>=n) || (INV2(j,4)==0)) % if the vehicle has arrived on or
before this week and sold on or later than this week (i.e. if vehicle is/was on the lot this week)
         Initial=[Initial;INV2(j,:)];
         Currentweek=[Currentweek;n];
      end
   end
Initial=[Currentweek,Initial];

Sold=sum(INV2(:,4)==n);% count how many vehicles were sold on week 10 in a given region
% Now we want to generate random sales equivalent to the number of vehicles which were
actually sold in the historical dataset.
Rand=[];
Choice=[]; % Vector of randomly chosen configurations
for y=1:Sold
Rand=[Rand,rand(1,1)];
Cf=unique(Initial(:,3));% array that has all the configuration present at week n
Rh=[];
SizeCf=size(Cf,1);
In=1;% Increment for index in Rh vector
for k=1:SizeCf % Assign Rhat to available configurations
  index= find(Standard(:,1)==Cf(k));
  Rh(In)=Standard(index,2);
  In=In+1;
End
% Here we want to normalize the Rhat for the available configurations in order to specify the
choice interval which was defined in the thesis

Sum=sum(Rh);% sum of Rh accross available configurations
Fhat=[];% Vector of normalized Rhats associated with available configurations on week n
Fhat(1,1)=Rh(1,1)/Sum; % Initialization of the normalization process
for m=2:SizeCf % Loop to normalize Rhat
   Fhat(1,m)=Fhat(1,m-1)+(Rh(1,m)/Sum);
end
Fhat=[0,Fhat];

   for x=1:SizeCf % this for loop is to assign choices to randomly chose probabilities
   if Rand(1,y)>Fhat(1,x) && (Rand(1,y)<=Fhat(1,x+1))
      Choice=[Choice,Cf(x,1)];
      break
   end
   end
   I=size(Initial,1); % number of rows in "Initial" Matrix
   for z=1:I % Deduct randomly chosen sales from the Initial Inventory
      if Initial(z,3)==Choice(y)
```

```
            Initial(z,:)=[];
            break
        end
    end
end
SimulatedINV=[SimulatedINV;Initial];
end


% Counting how many vehicles of each configuration were available that week after deducting the simulated random choices, similar to the way we did it to the actual historical dataset above.
Total=[];
for a=10:43
    Group3=[];%Matrix that group the records at week a
    Group3=SimulatedINV(find(SimulatedINV(:,1)==a),:);
    Count=unique(Group3(:,3));% find out how many different configurations are present on week a
    SizeCount=size(Count,1);
    week=a*ones(SizeCount,1); % assign current week a
    Count=[week,Count];
    Vcount=[];
    for b=1:SizeCount  % count how many vehicles of configuration b are present on week a
        Vcount=[Vcount;sum(Group3(:,3)==Count(b,2))];
    end
    Count=[Count,Vcount]; % Now this matrix shows the current week, configuration number, count of this configuration in this given week
      for CC=1:Numberofconfig % this for loop will add all missing configurations and will associated a number zero for the count
        Find1=Count(find(Count(:,2)==CC),:);
        if size(Find1,1)==0
            Count=[Count;[a CC 0]];
            [values, order] = sort(Count(:,2));% Sort the matrix in increasing order of configuration number
            sortedCount = Count(order,:);
        end
    end
    Total=[Total;sortedCount];
end
VehicleCount=[VehicleCount,Total(:,3)];
sim
end

% Now we want to average the simulated counts across the 100 iterations of the simulator
AverageNumberofVehicles=mean(VehicleCount(:,[2,end])');
HistoricalCountofVehicles=TotalReal(:,3);
```

```
Compare=[HistoricalCountofVehicles,AverageNumberofVehicles',weight]; % This matrix lists
both the historical counts, averaged simulated counts, and the weights associated with each
vehicleweek record in our dataset
SizeCompare=size(Compare,1);
ErrorWeek=zeros(SizeCompare,1); % This Vector include the weighted relative error for every
week
for f=1:SizeCompare
   if max(Compare(f,1),Compare(f,2))==0
   ErrorWeek(f,1)=0;
   else
   E=(Compare(f,3)*abs(Compare(f,1)-Compare(f,2)))/max(Compare(f,1),Compare(f,2));
   ErrorWeek(f,1)=E;
   end
end
FinalMatrix=[TotalReal(:,1),TotalReal(:,2),Compare,ErrorWeek];
```

B- Simulator Inventory Sales based

```
clc
clear all
load('Recordsunbalanced.mat','INV1','Global')
Cweek=Global(:,1); % current week
Location=Global(:,4);
Configset=Global(:,5);
Numberofconfig=max(Configset);% gives total number of available configurations
% INV1 is the matrix that contains all records for all regions over the
% whole time horizon %INV1=[[L,Configuration',A,S,INV];
Arrival=Global(:,6); % Arrival week for a given vehicle
Sold=Global(:,7); % The week in which a vehicle was sold
History=[Cweek,Location,Configset,Arrival,Sold];% Real historical inventory present each week
stating the arrival and sold week of each vehicle History=History(find(History(:,2)==1),:);%
Filter those for location 1=Boston
TotalReal=[];% The matrix that include the count of sold vehicles from each configurations for
every week period
for d=10:43
   Real=[];% Matrix that group the records at week d
   Real=History(find(History(:,1)==d),:);
   UniqueConfig=unique(Real(:,3));% find out how many different configurations are present on
week d
   SizeUniqueConfig=size(UniqueConfig,1);
   weeksReal=d*ones(SizeUniqueConfig,1); % assign current week
  SizeReal=size(Real,1);% How many vehicles available in week d
  SoldVector=[];%States the configurations being sold on week d
```

```
% In this loop, if the sold week of a given record is equal to current week d then this recorded is
counted as sold
  for i=1:SizeReal
      if Real(i,5)==d
        SoldVector=[SoldVector,Real(i,3)];% This vector will list the configuration numbers of
the vehicle being sold on week d
     end
  end
  Countsold=[]; % the vector that will list the number of vehicles of each configuration sold on
week d by counting how many times is configuration j repeated in the ‘SoldVector’
  for j=1:SizeUniqueConfig
     S=sum(SoldVector==UniqueConfig(j));% Count how many vehicles were sold of each
configuration
     Countsold=[Countsold,S];
  end
   Summary=[weeksReal,UniqueConfig,Countsold']; % This matrix will summarize what
configurations are available on week d and how many of each where sold
   for CC=1:Numberofconfig % this for loop will add all the configurations which aren’t
available on week d and will associated a number zero for the count
      Find=Summary(find(Summary(:,2)==CC),:);
      if size(Find,1)==0
         Summary=[Summary;[d CC 0]];
      end
   end
[values, order] = sort(Summary(:,2));
   sortedSummary = Summary(order,:); % Sort the matrix in increasing order of configuration
number
   TotalReal=[TotalReal;sortedSummary];
end
 % Calculating Global Sold Vehicles quantities
   for O=1:579
   Sales(1,O)=sum(TotalReal(find(TotalReal(:,2)==O),3)); % This vector will sum the numbers
of vehicles sold on configuration O over the whole time horizon
   end
List=[1:579]; % List all possible configurations available in the assortment
Sales=[List',Sales'];
% Calculating Global Sold Weights
SizeHistory=size(History,1);
weight=[];% Combines all the weeks in one matrix
 for d=10:43
   FilterHistory=[];% Returns all vehicles available on week d
   w=[];% Returns the weights for each configuration in every week
   conf=[];% This vector will list all the configurations that were available on week p
   Beta=[];% This vector will return the global number of sold vehicles of the configurations
which are available on a particular week
```

```
    for U=1:SizeHistory % This for loop will filter for weekly inventories
        Filterhistory=History(find(History(:,1)==d),:);
    end
    conf=unique(Filterhistory(:,3));
    sizeconf=size(conf,1);
    for q=1:sizeconf % This loop will populate the global quantity of vehicles sold of
configuration q over the time horizon from the Sales matrix
        Beta(q,1)=Sales(find(Sales(:,1)==conf(q)),2);
    end
    for r=1:sizeconf
        w(r,1)= Beta(r,1)/sum(Beta);
    end
    P=d*ones(sizeconf,1); % week indicator
    w=[P,conf,w];
    for s=1:579
        if size(w(find(w(:,2)==s),:),1)==0
            w=[w;[d s 0]];
        end
    end
    [values, order] = sort(w(:,2));
    sortedw = w(order,:); % Sort the matrix in increasing order of configuration number
    weight=[weight;sortedw];
 end
% From here on the code is very similar to the count simulator code in terms of counting how
many vehicles got sold on a given week and generating equivalent numbers of random choices.
With 2 differences: 1- Initial inventory at every week is taken from the real historical dataset.
2- The error is calculated based on sold counts rather than inventory counts
3- Generating a sold vehicle count vector is generated within the for loop of each week.

load('Rhatfileclassifier1.mat','Configurationset','Rhat');% Here we load all available
configurations with their associated Rhat
X=size(INV1,1);
Standard=[Configurationset,Rhat];% A matrix that shows all the configuration with their
associated unconditional turn rate
INV2=INV1(find(INV1(:,1)==1),[1:4]);% The Matrix that contain sales record for region 1=
Boston
Lamda=size(INV2,1);% Size of the regional

%Now we need to generate a matrix similar to TotalReal to track the number of simulated sold
vehicles, so we first define the first 2 columns to be equivalent to TotalReal and then we modify
the values of all records of the third column of TotalReal

Total=TotalReal; % In those 2 lines I want to define a matrix that lists all possible configuration
at every week with its associated week, and a zero count of sales
SizeTotal=size(Total,1);
```

```
VehicleCount=[Total(:,1),Total(:,2)]; % This matrix will track all the sales counts generated by every iteration of the simulator
for sim=1:500 % Repeat Simulator loop
Vcount=zeros(size(TotalReal,1),1); % This vector will track the count of sold vehicles according to the simulator results
Incr=1; % this is an index increment used to fill up the values for Vcount vector
   S=[];% Vector that will track total number of sales for all configurations each week
SimulatedINV=[]; % This matrix is to track the simulated inventory changes at each week over the whole time horizon
%Simulation starts here
for n=10:43
   Initial=[];
   Currentweek=[];
   for j=1:Lamda % loop that generates the initial inventory on week on each week
      if (INV2(j,3)<=n) && ((INV2(j,4)>=n) || (INV2(j,4)==0)) % if the vehicle has arrived on or before this week and sold on or later than this week (i.e. if vehicle is/was on the lot this week)
         Initial=[Initial;INV2(j,:)];
         Currentweek=[Currentweek;n];
      end
   end
Initial=[Currentweek,Initial];
SizeInitial=size(Initial,1);
Sold=sum(INV2(:,4)==n);% count how many vehicles where sold on week 10 in a given region
S=[S;Sold]; % Vector S tracks how many vehicles were sold each week
Rand=[];
Choice=[]; % Vector of randomly chosen configurations
for y=1:Sold
Rand=[Rand,rand(1,1)];
Cf=unique(Initial(:,3));% array that has all the configuration present at week n
Rh=[];
SizeCf=size(Cf,1);
In=1;% Increment for index in Rh vector
for k=1:SizeCf % Assign Rhat to available configurations
  index= find(Standard(:,1)==Cf(k));
  Rh(In)=Standard(index,2);
  In=In+1;
end
Sum=sum(Rh);% sum of Rh accross available configurations
Fhat=[];% Vector of normalized Rhats associated with available configurations on week n
Fhat(1,1)=Rh(1,1)/Sum; % Initialization of the normalization process
for m=2:SizeCf % Loop to normalize Rhat
   Fhat(1,m)=Fhat(1,m-1)+(Rh(1,m)/Sum);
end
Fhat=[0,Fhat];
```

```
  for x=1:SizeCf % this for loop is to assign choices to randomly chose probabilities it will
check in which interval of Fhat does the random number fall in and assign the choice based on
that
  if Rand(1,y)>Fhat(1,x) && (Rand(1,y)<=Fhat(1,x+1))
    Choice=[Choice,Cf(x,1)];
    break
  end
  end
  % Counting how many vehicles of each configuration were sold that week
  for i=Incr:Incr+578 % This for loop will increment the number of sales vehicles of the
randomly chosen configuration on week n by 1
    if Total(i,2)==Choice(y)
      Vcount(i,1)=Vcount(i,1)+1;
    end
  end
end
Incr=Incr+579;% we are adding 579 so that at the next iteration, representing the next week, the
for loop will start incrementing the values of Vcount associated with the current week n. Given
that at each week we are counting the sales for all possible 579 configurations. Even if they
aren’t available, they will be assigned a value of 0.
end
VehicleCount=[VehicleCount,Vcount];
sim
end
Iteration=sum(VehicleCount(:,[3:end])');
AverageNumberofVehicles=Iteration/size(VehicleCount(:,[3:end])',1);% Average Simulated
Number of vehicles sold for every config in every week
HistoricalCountofVehicles=TotalReal(:,3); % Real historical sold vehicles
Compare=[Total(:,1),HistoricalCountofVehicles,AverageNumberofVehicles',weight(:,3)];
SizeCompare=size(Compare,1);
ErrorWeek=[]; % This is the vector that will list the averaged errors for every week
ErrorWeekPrime=[]; % The weighted relative error
CW=10; %starting week to compute error
for m=1:34
  Group4=Compare(find(Compare(:,1)==CW),:);% Group the weekly inventory
  Size4=size(Group4,1);
  Error=[];% This vector will list the error associated with each configuration on week m
  ErrorPrime=[];
  for f=1:Size4
    if max(Group4(f,2),Group4(f,3))==0
      E=0;
      EPrime=0;
    else
     EPrime=Group4(f,4)*abs(Group4(f,2)-Group4(f,3))/max(Group4(f,2),Group4(f,3));
     E=abs(Group4(f,2)-Group4(f,3));%/max(Group4(f,2),Group4(f,3));% This is the absolute
difference between historical and simulated sold counts
```

```
        end
   Error=[Error,E];
   ErrorPrime=[ErrorPrime,EPrime];
    end
    GlobalWeekError= sum(Error)/S(m); % This will calculate the averaged error for week m by dividing the sum of the error by the total number of sold vehicles on week m
    GlobalWeekErrorPrime=sum(ErrorPrime); % This is the weighted relative error
    ErrorWeek=[ErrorWeek,GlobalWeekError];
    ErrorWeekPrime=[ErrorWeekPrime,GlobalWeekErrorPrime];
    CW=CW+1;
end
```

REFERENCES

Mahajan, Siddharth, and Garrett van Ryzin. "**Stocking retail assortments under dynamic consumer substitution."** *Operations Research* 49.3 (2001): 334-351.

Mahajan, Siddharth, and Garrett Van Ryzin. **"Inventory competition under dynamic consumer choice."** *Operations Research* 49.5 (2001): 646-657.

Kok, A. G., and M. L. Fisher. "**Demand Estimation and Assortment Optimization Under Substitution: Methodology and Application**." *Operations Research* (2007): 1001-021. Web. 26 Oct. 2014.

Yücel, Eda, Fikri Karaesmen, F. Sibel Salman, and Metin Türkay. "**Optimizing Product Assortment under Customer-driven Demand Substitution**." *European Journal of Operational Research* (2009): 759-68. Web. 26 Oct. 2014.

Karabati, Selçuk, Bariş Tan, and Ömer Cem Öztürk. "**A Method for Estimating Stock-out-based Substitution Rates by Using Point-of-sale Data**." *IIE Transactions* (2009): 408-20. Web. 26 Oct. 2014.

Rodríguez, Betzabé, and Göker Aydın. "**Assortment Selection and Pricing for Configurable Products under Demand Uncertainty**." *European Journal of Operational Research* (2010): 635-46. Web. 26 Oct. 2014.

Puskorius, Gintaras V., and Brian R. Goodman. **Turn Rate Calculation**. Ford Motor Company, assignee. Patent US 8214313 B1. 3 July 2013. Print

ABSTRACT

A CUSTOMER CHOICE MODELING FRAMEWORK FOR THE ASSORTMENT PLANNING OF CONFIGURABLE PRODUCTS IN THE AUTOMOTIVE INDUSTRY

by

FARAH DUBAISI

May 2015

Advisor: Alper Murat, PhD

Major: Manufacturing Engineering

Degree: Master of Science

Due to the increased competition in the auto industry, proliferation of the vehicle models and increased customer need for choice and customization, it has become more critical than ever to offer a variety of features and customization flexibility while at the same time restraining and, even better, cutting down the costs. Product complexity, in the automotive industry, can be measured by the size of the assortment offered, i.e., set of vehicle configurations a customer can choose from (e.g., for a given model of a brand). While complexity fosters growth with increased alignment of product characteristics and customer needs, it results in decreased revenue (e.g., cannibalization) and profitability (e.g., increased total supply chain costs). Companies that manage complexity by improving their products' true profitability have seen savings of 10 percent to 15 percent on their cost of goods sold.

In order to determine the optimal complexity that should be offered, the company must first understand its customers buying behavior, and their response at the instances where their primary vehicle configuration choice is not offered or is stocked out. In this thesis, we develop a customer choice modeling framework that predicts the likelihood of an average customer to buy a specific vehicle configuration in a given assortment offering. Our modeling approach utilizes neural networks to predict, based on the historical dealership level sales and inventory data, how likely a given configuration will sell when it's offered along with a set of configurations. These

configuration level sale probability estimates are then used to estimate the attraction factor for each feature included in the vehicle configuration. The attraction factor of each feature represents feature's individual contribution to the probability of sale of the configuration as a whole. With this feature level estimation, the probability of sales for any feature combination or vehicle configuration can be estimated (including those configurations not yet built or offered). We report on the performances of several modeling and neural network based estimation approaches using historical dataset from a major US automotive OEM. Our models are parametric and thus can be used within an assortment planning model to determine the optimal product assortment that optimizes complexity by considering true profitability of the configurations in the assortment.

AUTOBIOGRAPHICAL STATEMENT

Farah Dubaisi is a graduate Manufacturing Engineering student at Wayne State University. Farah received her Bachelor degree in Mechanical Engineering at American University of Beirut in spring 2013. Throughout her education, she worked hard to maintain a high GPA, 3.9, on one hand, and attain practical industrial experience, on the other hand. She have succeeded in completing two internships at TRW Automotive and NYX Incorporated, in the capacity of project management. Farah developed her expertise in Excel and Matlab modeling during her work towards this thesis.