Predicting whether Visitors Purchase after only one Click

Industrial Partner: Bluestem Brands

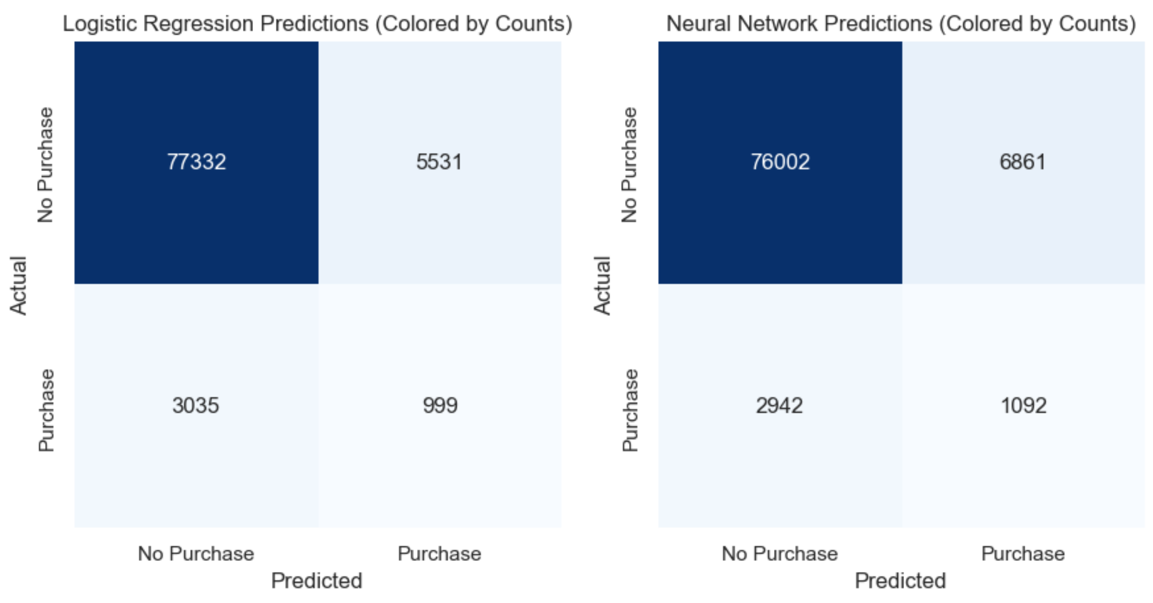
Contributors: Ayaan Gopalan, Tori Wang, Xi Wu, Ethan Young

Executive Summary

The primary question we wanted to answer was: given the first click of a user who just entered the site, can we predict if the user will make a purchase? We studied this question by implementing 2 models: logistic regression (because it is a simple model that served as a baseline) and neural networks (because they are easily implemented and flexible). Both models output a value between 0 and 1 that can be interpreted as the probability a user makes a purchase. If the value is greater than the threshold of 0.5, a user is predicted to purchase; if less than 0.5, a user is predicted to not make a purchase.

We utilized a multi-step feature selection procedure to narrow down the variables in the given dataset before using it in our models. We first extracted only the first server call of all unique visits and created a binary target variable to keep track if the visit resulted in a purchase. We then removed variables where at least 95% of their values were missing. Next, we removed variables where 99% of their values were similar.. Lastly, we used random forest feature importance to extract the top 30 variables to use in our models. We subset those 30 features from the initial dataset and used them to make predictions with our models. A key characteristic of our data was its imbalance: the number of users who did not purchase far outnumbered the number of users that did. We took measures to deal with this in our model building.

Our main results are summarized by the figures on the next page. It was clear that while the neural network performed slightly better, both models struggled with the data imbalance. The bottom row shows both models predicting most of the users who purchased as not purchasing.



We present 2 potential applications of our results. The first is plugging in data collected in real time to a pre-trained neural network to make predictions for all users visiting the site. This allows Fingerhut to gauge the intent of users based on their first click. If they are predicted to purchase, Fingerhut can optimize search results and offer promotional codes based on subsequent page visits to encourage them to make a purchase. If they are predicted to not make a purchase, Fingerhut can show them a more personalized homepage with targeted item recommendations to encourage them to continue exploring the site. We additionally propose to lower the 0.5 threshold, which helps the models predict users that purchase much more accurately. This is an approach that we hope can improve the performance of our models.

The second application is having a different neural network for every season. The motivation for this is that customer behaviors such as the use of promotional codes and categories of items that are frequently purchased likely change throughout the year. Making predictions with neural networks is particularly efficient and flexible because they can be saved and used at any time. The main difficulty is producing models that perform well because they require extensive testing to maximize their accuracy.

Table of Contents

1 Introduction

Bluestem Brands, the parent company of Fingerhut, asked us to work with a large dataset focused around user generated server events on the Fingerhut website. Fingerhut is a financial technology (FinTech) company that, through its own online retailing platform, offers flexible payment options to customers looking to establish, build, or rebuild credit. Bluestem Brands and Fingerhut were particularly interested in drawing insights from the data in order to discern successful advertising strategies and identify categories of buying customers. Our work over the quarter focused on answering the following question: based on the available attributes of a visitor after their first click on the website, what is the probability that visitor will make a purchase?

1.1 Problem Restatement

The main problem we were tasked with answering revolved around predicting user behavior on the Fingerhut website. Specifically, we attempted to identify visitors who are most likely to place an order on Fingerhut. In order to produce useful insights for Bluestem Brands, they provided us with a data set they do have access to: individual user activity on their site, including a range of metadata for each visitor. Mathematically, we can rephrase our problem as follows: for each user i with m features $x_{i1}, \ldots, x_{im}$, what is the probability $p(x_i)$, they will make a purchase? To answer this question, we implemented logistic regression and neural network models to perform binary classification.

2 Data Processing

2.1 The Dataset

The dataset given by Bluestem Brands consisted of 7 days of server calls from the Fingerhut website, taken from December 10th, 2022 to December 16th, 2022. There were

42,730,149 observations (server calls) in the entire dataset, which we assumed all corresponded to user clicks. For each server call, there were 283 variables including a visit ID, user ID, and IP address. Of the 283 variables in the data set, a significant portion of them returned primarily null values, making it difficult to extract useful information from them. Many of the variables also had an unclear explanation for how they were calculated or collected, which made it difficult to read more into them or make assumptions about them. For example, 'evar83' was the customer behavior score, but we did not know how it was computed.

There was also a large data imbalance between the visits that resulted in a purchase and the visits that did not result in a purchase in the dataset. Over 95% of visits to Fingerhut during the given week did not result in a purchase. This imbalanced distribution presented challenges in building a model that was not heavily biased toward users who did not make a purchase.

2.2 Data Cleaning

The first step in our data cleaning process was extracting the first server call observation from every visit in the dataset, and attaching a target variable to each one. To extract all of the first server calls, we selected observations where the variable 'visitpagenum,' which returned the page number of the visit, equaled 1. In order to ensure each of the observations we selected corresponded to a unique visit, we checked that the subsetting resulted in no duplicate visit IDs. After we acquired the dataset of first server call observations, we used the variable 'checkoutthankyouflag' to add a binary target variable to each observation that identified whether the visit resulted in a purchase, with 0 indicating a visit with no completed purchase and a 1 representing a visit with a completed purchase. Thus, 0 and 1 were the class labels.

Once we appended a target variable to all the first server calls in our dataset, we removed 22 variables we believed would bias or negatively influence our model. We categorized these variables into 2 types: redundant variables, and dates or times. We removed redundant variables when the information was also contained in another variable. Two examples of this were the variables ‘visitoridhigh’ and ‘visitoridlow’ which contained the same information as the variable ‘visitID,’ and the variable ‘cookieID’ which contained the same information as the variable ‘ip.’ As for the removal of variables containing date or time information, because we were only given a week of data, we did not want our model to factor in the variation of the time or date of visits, as it may overfit to them and lose applicability to other parts of the year.

2.3 Feature Selection

Taking into account the large size of the dataset, the uncertainty of how some variables were collected or calculated, and the high frequency of missing values, we chose a high-level approach to data cleaning and feature selection as opposed to hand-picking features for our model. Specifically, we thought that manually selecting variables from the dataset would be arbitrary and could result in the exclusion of predictive variables.

One of the main drawbacks of high-level feature selection is that it can sacrifice the practical relevance of hand-picked variables. However, because our model inputs were limited to the first server call of visits to the Fingerhut website, our high-level feature selection and data cleaning process was already limited to variables that Bluestem Brands would have access to at the beginning of every website visit. We felt this helped address the limitations of high level feature selection and supported this approach. Another limitation of our feature selection process was that we only worked with the first day of data due to the size of the dataset. This biased our results because the data from one day may not be fully representative of the entire dataset.

Our first step in our feature selection process was analyzing the proportion of null values of each variable with the intent of removing variables that returned predominantly null values.

Because our primary goal was to identify visits that will result in a purchase, we calculated the proportion of missing values of variables in only the observations of purchase visits. Considering only the visits that resulted in a purchase was an attempt to address the large data imbalance and focus more on the minority class. After graphing the histogram of the proportion of values amongst variables in the data set (shown in Figure 1), we selected 95% missing values or greater as a reasonable cutoff for variables we should remove, which left us with 119 remaining variables.

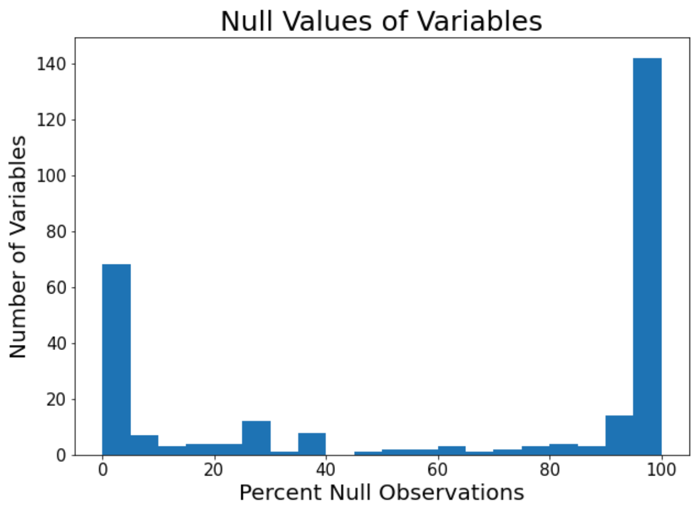


Figure 1: Histogram showing the proportion of null values amongst variables in purchase visit observations.

Following the removal of variables with 95% or more missing values in purchase visit observations, we analyzed the variance of the remaining variables to to remove variables that returned the same value in most of the first server calls, as we assume these variables would be less helpful in predicting the outcome of visits.[1] We first grouped the variables into either

[1] Paul, S. (2022, February 16). 'How to Use Variance Thresholding for Robust Feature Selection'. *Towards Data Science*. https://towardsdatascience.com/how-to-use-variance-thresholding-for-robust-feature-selection-a4503f2b5c3f. Accessed 6 Mar 2023.

categorical or numerical groups based on the variable descriptions provided by Bluestem Brands. With the categorical variables we converted the null values into a 'None' category in case the missing values provided any additional information. This 'None' category remained throughout the feature selection process and was also used in our model building. For the numerical variables, we excluded the null values when analyzing their variances. After accounting for the null values in both the categorical and numerical variables, we calculated the variance of the 119 remaining variables. We then removed all variables with a variance less than or equal to an arbitrarily chosen threshold of 0.01, which indicated that 99% or more of the variable's values were similar. This threshold removed 12 variables and left 107.

The last step in our feature selection process was training a random forest classifier and analyzing the feature importances. We chose to analyze random forest feature importances as they can accurately compare both categorical and numerical variables, and they can also estimate the predictivity of the variables towards a target variable, which in our case is whether the visit will result in a purchase. To preprocess the dataset for the random forest classifier, we encoded the categorical variables using leave-one-out (LOO) target encoding,[2] which outputted values in a 0 to 1 range. We discuss LOO encoding in the next paragraph. The numerical variables were also normalized to a 0 to 1 range in order to standardize the range of values for each feature in the data. This allowed the classifier to weigh the categorical and numerical variables evenly without significantly altering the information in the dataset. After rescaling, we filled any null values in the numeric variables with -999 so the classifier easily recognized these values.[3]

[2] Roy, B. (2019, July 16). All about Categorical Variable Encoding. Towards Data Science. https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02. Accessed 8 Mar 2023.
[3] Eddie_4072. (2021, May 19). 'Dealing with Missing Values in Python: A Complete Guide'. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide/. Last modified 15 Feb 2023. Accessed 15 Mar 2023.

LOO encoding is a Bayesian encoding technique that encodes the categorical variable with the mean value of the target variable for that category; the mean value calculation excludes the target value of the current row. To our knowledge, there is no peer-reviewed publication defining LOO encoding, but Hancock and Khoshgoftaar (2020) discussed this topic in the context of training neural networks. We chose this technique because it did not add extra columns to the dataset, which made model training quicker. In addition, the values assigned to the variables were not arbitrary like they would have been if we used other categorical encoding schemes such as label or ordinal encoding. However, two of the main drawbacks of LOO encoding are overfitting and response leakage, as the target variable itself is used in the encoding of the values. Leaving the current observation out of the target mean calculation helped to combat these issues.

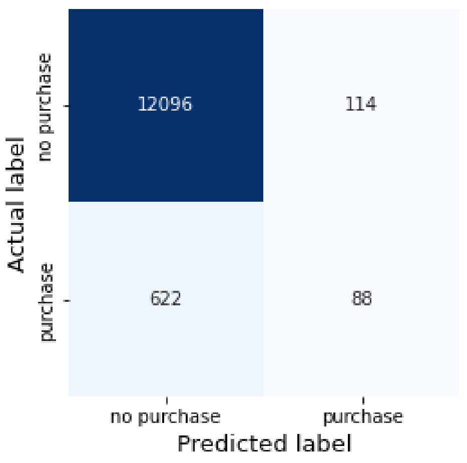
To train the random forest classifier, we generated a 70% training set and 30% testing set from the preprocessed dataset. We also utilized class weights ω, which penalized incorrect predictions of the minority class, when training the classifier to help account for the data imbalance between purchase visits and non-purchase visits. More details and descriptions of other methods addressing class imbalance are found in a survey article by Yanminsun et al. (2011). Let N be the total number of observations and N_c be the number of observations belonging to class c. Then, with 2 classes, the weight of class c, ω_c, is

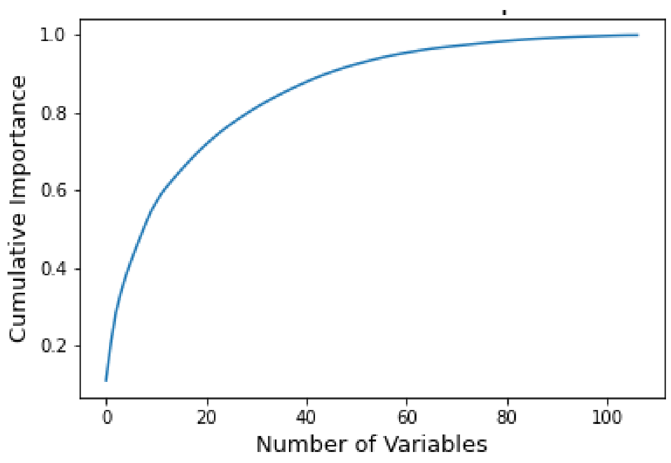
$$\omega_c = \frac{N}{2\,N_c}$$

We chose class weights because they help to prevent the model from classifying all visits as non-purchasing. This would result in high model accuracy, but would not be very useful as the features would be relevant only to users that do not purchase. While the model may still slightly prioritize classifying the majority ‘No Purchase’ class with class weights, training with the full dataset is more representative than undersampling from the majority class.

The random forest classifier produced a test set accuracy of roughly 94%. The confusion matrix is shown in Figure 2. We then calculated the feature importances from the classifier. Specifically, we computed the mean decrease in impurity—which is the total sum of the decrease in node impurity for a feature—weighted by the proportion of samples reaching each node and averaged over all trees of the ensemble. More details are found in chapters 2 and 4 of Breiman et al. (1984). To analyze the feature importances, we graphed the cumulative feature importance with respect to the number of variables (shown in Figure 2). Using the graph, we chose to extract the first 30 variables for our final feature set, which accounted for approximately 80% of the feature importance. Feature importance subsetting was the final step of feature selection.

The top 10 of the 30 features selected were: ‘service,’ ‘pageevent,’ ‘post_evar24,’ ‘post_evar23,’ ‘evar23,’ ‘prop38,’ ‘searchengine,’ ‘evar24,’ ‘ip,’ ‘geocountry,’ and ‘post_evar13.’ These variables contain user attributes and actions that can be observed from the first click. The variable ‘post_evar23’ is the visitor customer number, ‘geocountry’ is the physical location of the user, and ‘prop38’ captures the institution id of a user. These features and ‘ip’ (i.e., IP address) contain information on users that are instantly available when they access the website. Other variables such as ‘searchengine,’ ‘service,’ and ‘post_evar13,’ which flags a visit as BCP if on the new BCP platform, give information about the initial actions of a user.





a) Random forest confusion matrix

b) Random forest cumulative feature importance

Figure 2: Results from the random forest classifier. **Left**: Confusion matrix colored by counts (darker blue indicates higher counts). **Right**: The cumulative feature importance of the random forest with the number of variables on the x-axis.

2.4 Training and Testing Sets

We now detail how we generated the training and testing sets for our model. We first extracted the data frames of first server calls from each day. Next, using the list of the 30 most important features from the feature selection process, we subset the columns of those same features from each data frame. That subset of data from days 0, 1, 2, 4, 5, 6 constitute the training set, denoted by $\mathbf{X}_{\text{train}}$. $\mathbf{X}_{\text{train}}$ has corresponding labels $\mathbf{Y}_{\text{train}}$. Day 3 constitutes the testing set, denoted $\mathbf{X}_{\text{test}}$. $\mathbf{X}_{\text{test}}$ has corresponding labels $\mathbf{Y}_{\text{test}}$. The dimensions of the resulting data matrices are

- $\mathbf{X}_{\text{train}} \in \mathbb{R}^{408874 \times 30}$ and
- $\mathbf{X}_{\text{test}} \in \mathbb{R}^{86897 \times 30}$.

We then transformed $\mathbf{X}_{\text{train}}$ by fitting an encoding to the categorical variables in $\mathbf{X}_{\text{train}}$ using LOO encoding and normalized the numerical variables in $\mathbf{X}_{\text{train}}$ to 0 to 1 as described and motivated in section 2.3. We then transformed the categorical variables in $\mathbf{X}_{\text{test}}$ using the encoding fit on $\mathbf{X}_{\text{train}}$ and normalized the numerical variables in $\mathbf{X}_{\text{test}}$ to 0 to 1.

We had 18833 observations labeled 'Purchase' in $\mathbf{Y}_{\text{train}}$ and 4034 observations labeled 'Purchase' in $\mathbf{Y}_{\text{test}}$.We considered day 3 as our testing set because we wanted to capture user behavior on a weekday rather than a weekend, which might not be completely reflective of general user behavior. We use the terms 'testing set' and 'validation set' interchangeably.

3 Data Analysis

3.1 Overview

We implemented 2 classification models. The first, a logistic regression, was chosen as a baseline because it is an interpretable, simple model that is easily implemented and has very short computation time. The second was a shallow neural network, chosen because of its ease of implementation and, more importantly, its flexibility. By flexibility we mean that we have a great deal of control over the hyperparameters of the neural network, which can be intuitively thought of as the predetermined parameters of the neural network itself that affect how well it performs before training data are inputted. We discuss hyperparameters in greater detail in section 3.3.1. We addressed the class imbalance by utilizing class weights as described in section 2.3.

Both models output a prediction $\hat{y}_i$ for a user i that can be interpreted as a probability $p(x_i)$. If $p(x_i)$ is greater than 0.5, the user is classified as 'Purchase' and, if $p(x_i)$ is less than 0.5, the user is classified as 'No Purchase.' If $p(x_i)$ is exactly 0.5, then the model randomly decides.

Finally, we gauged model performance by using the accuracy and area under the receiver operating characteristic (AUC) metrics.[4] We used accuracy because it is a straightforward and

[4] Google, n.d. 'Classification: ROC Curve and AUC'. *Google Developers*. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Last modified 18 Jul 2022. Accessed 15 Mar 2023.

interpretable evaluation metric. However, this metric fails to consider differences in class distributions. This is what motivated us to use the AUC score, which helped us understand how well the model can distinguish between classes. This metric is computed with 2 parameters: true positive rate (TPR) and false positive rate (FPR). The TPR and FPR are defined as

- $TPR = \frac{TP}{TP+FN}$, and

- $FPR = \frac{FP}{FP+TN}$,

where *TP* stands for true positive, *FN* for false negative, *FP* for false positive, and *TN* for true negative. AUC can be interpreted as the probability of a model prioritizing a random positive example over a random negative one. Figure 3 shows different classifiers in the receiver operating curve (ROC) space defined by the FPR and TPR as the *x* and *y*-axes, respectively.

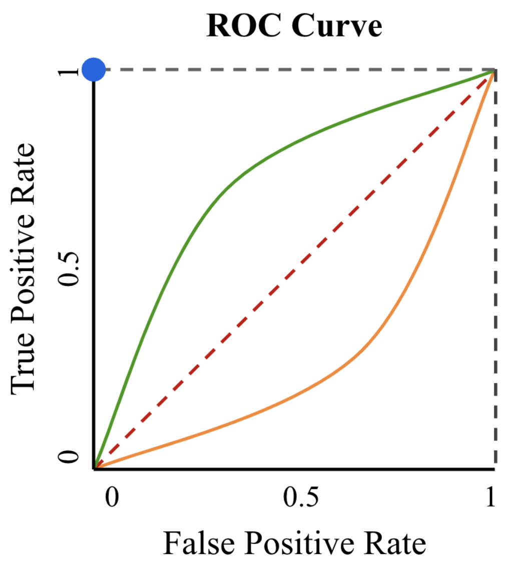


Figure 3: Graphic of the ROC space. The diagonal red dotted line represents a model that classifies randomly (AUC = 0.5). The blue dot in the top left corner represents a perfect classifier (AUC = 1). The green curve above the red dotted line represents a model that classifies better than random (AUC > 0.5). The orange curve below the red dotted line represents a model that classifies worse than random (AUC < 0.5).

3.2 Logistic Regression

We implemented logistic regression with default settings in scikit-learn.[5] Logistic regression involves estimating the coefficients $\beta_0, \beta_1, \ldots, \beta_{30}$ of a logistic model. For conciseness, we let p be $p(x_i \mid \beta_0, \beta_1, \ldots, \beta_{30})$, which denotes the classification probability $\hat{y}_i$ of user i. The model is constructed such that the log-odds $ln\,[p/(1-p)]$ is a linear function that depends on the features $\mathbf{X}_1, \ldots, \mathbf{X}_{30}$. The parameters $\beta_0, \beta_1, \ldots, \beta_{30}$ are determined by maximizing the logarithm of the likelihood (log-likelihood) function $L_{\text{likelihood}}$, and is written as

$$ln\, L_{\text{likelihood}}(\beta_0, \beta_1, \ldots, \beta_{30}) = \sum_{i=1}^{N} [y_i\, ln(\hat{y}_i) + (1-y_i)\, ln(1-\hat{y}_i)].$$

Mathematically, logistic regression is the logarithm of the odds (log-odds) of an event and is written as

$$ln(\frac{p}{1-p}) = \beta_0 + \beta_1\mathbf{X}_1 + \ldots + \beta_{30}\mathbf{X}_{30}.$$

3.3 Neural Network

The model we compared with the baseline logistic regression model was a shallow feedforward neural network implemented with Keras,[6] an easy-to-understand open-source Python package. This model is also known as a multilayer perceptron (MLP). In binary classification, a MLP sends data through layers of computation to output an inference ŷ that is a value between 0 and 1. In our project we conducted experiments with 2 hidden layers. We will also refer to this model as a '3-layer perceptron.' This was chosen as an arbitrary balance between the ability of the MLP to model the data and the computational cost associated with having more layers. The number of layers is a hyperparameter itself.

[5] scikit-learn, n.d. 'sklearn.linear_model.LogisticRegression'. *scikit-learn*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression. Accessed 2 Mar 2023.
[6] Keras, n.d. 'Models API'. *Keras*. https://keras.io/api/models/. Accessed 18 Feb 2023.

Each layer involves a matrix multiplication followed by an activation function *f* and is composed of nodes with connections between them called weights that have randomly chosen values that are initialized before training. The weights between layers compose a weight matrix **W** that is multiplied with the input data **X**. We also included trainable biases **B** in each layer that shift the input of the activation function by pushing classifications more toward 0 or 1. The number of weights and the activation function in each layer are also hyperparameters.

Mathematically, a 3-layer perceptron is written as

$$\hat{y} = f(f(f(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{J}\mathbf{B}^{(1)})\mathbf{W}^{(2)} + \mathbf{J}\mathbf{B}^{(2)})\mathbf{W}^{(3)} + \mathbf{J}\mathbf{B}^{(3)}),$$

where **J** is an $N\times1$ vector that ensures the biases are added element-wise to each matrix after matrix multiplication in each layer. The superscripts (ℓ) denote the ℓ-th layer. $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$ are the hidden layers, while $\mathbf{W}^{(3)}$ is the output layer. We chose 256 nodes in both $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$. While these values were found empirically to perform well, much more work is needed to know what numbers of nodes in each layer give the best model performance. The total number of trainable parameters (i.e., weights and biases) is 73985 (73472 weights and 513 biases). Figure 4 gives an overview of the model.

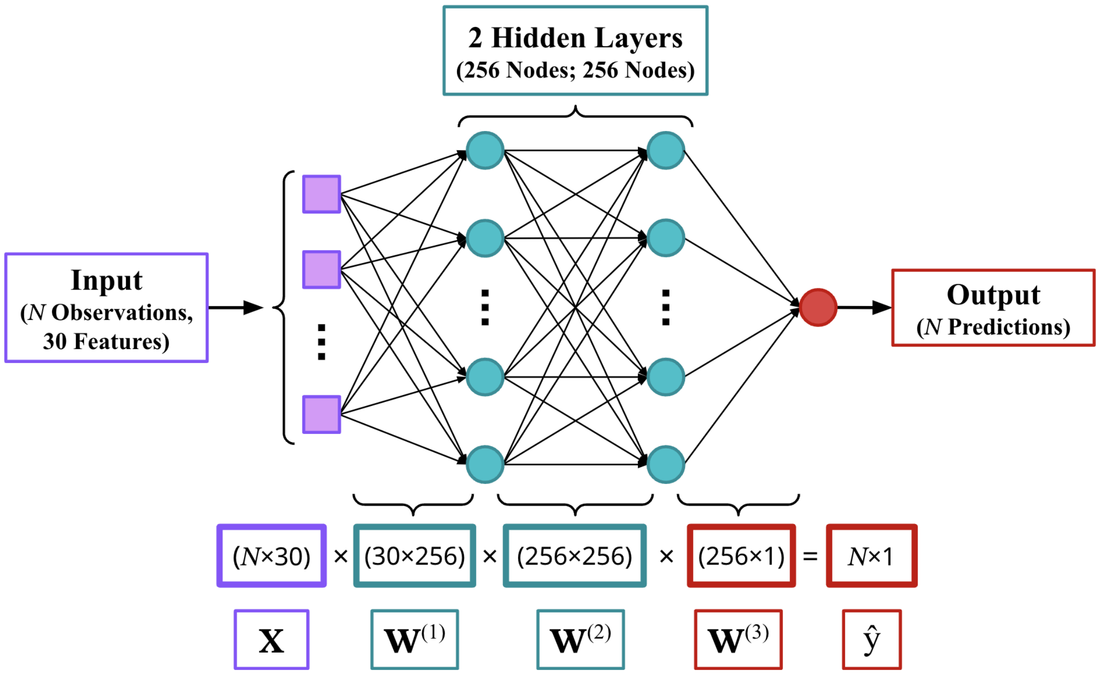


Figure 4: Diagram of the 3-layer perceptron used to generate the results in this report. The dimensions of $\mathbf{W}^{(1)}$, $\mathbf{W}^{(2)}$, and $\mathbf{W}^{(3)}$ are at the bottom. Biases are not included.

3.3.1 Model Training and Hyperparameters

Neural network training involves successive iterations of backpropagation, done with an optimization algorithm. This is the process where the neural network optimizes the weights of a neural network to maximize model accuracy. Each iteration is called an epoch, wherein all batches of data are used to update the weights. The batch size is the number of training data that are used in each step of optimization. We found empirically that most batch sizes produced similar results. In our project, we used the Adam optimizer, a computationally efficient first-order stochastic gradient method that was easily implemented and scaled well with large datasets. Adam uses adaptive estimates of the first moment (i.e., the mean) and the second moment (i.e., the variance) to update the parameters. The full details are specified in the paper by Kingma and Ba (2015). The primary advantage of using Adam is having low memory requirements, which made it a particularly appealing optimizer for this project. An additional hyperparameter in relation with this optimizer is the step size, which dictates how large of a gradient step is taken. We found empirically that a step size of 0.0005 performed well.

The activation function we used in all layers is the sigmoid function S. The formula for this function for a given input z is

$$S(z) = \frac{1}{1+e^{-z}}.$$

We found empirically that using the sigmoid function in all layers performed well and was more stable (i.e., the training accuracy did not fluctuate wildly during training).

The loss function L_{loss} is binary cross-entropy,[7] which is the negative weighted version of the aforementioned log-likelihood function. Cross-entropy, or log-loss, is preferred to other loss functions such as mean squared error because it is interpretable in that it considers classes and the probabilities of their predictions. Binary cross-entropy is written mathematically as

$$L_{\text{loss}} = -(1 / N) \sum_{i=1}^{N} [y_i \, ln(\hat{y}_i) + (1-y_i) \, ln(1-\hat{y}_i)].$$

To prevent overfitting, we utilized dropout, which is a regularization technique that drops nodes in layer ℓ and their connections during training with a pre-specified probability $p^{(\ell)}_{\text{drop}}$ ranging from 0 to 1. During testing, the entries of the weight matrices $\mathbf{W}^{(\ell)}$ are scaled by $p^{(\ell)}_{\text{drop}}$ (i.e., $\mathbf{W}^{(\ell)}$ becomes $p^{(\ell)}_{\text{drop}}\mathbf{W}^{(\ell)}$). The goal with dropout is to prevent an effect called co-adaptation, in which a neural network is too reliant on particular weights during the training process. The full details are found in the paper by Srivastava et al. (2014). We use dropout values of 0.9 in

[7] Godoy, D. (2018, November 21). 'Understanding Binary Cross-Entropy / Log Loss: A Visual Explanation'. *Towards Data Science*. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. Accessed 8 Mar 2023.

both hidden layers. We found empirically that dropout values ranging from 0.5 to 0.9 generated similar results for the MLP shown in Figure 4. Smaller values led the model to overfit more harshly. We concluded that while the effects of larger values of dropout were subtle in this project, further tuning was needed to see which values heavily impacted model performance. Figure 5 is an example of what dropout looks like in a MLP.

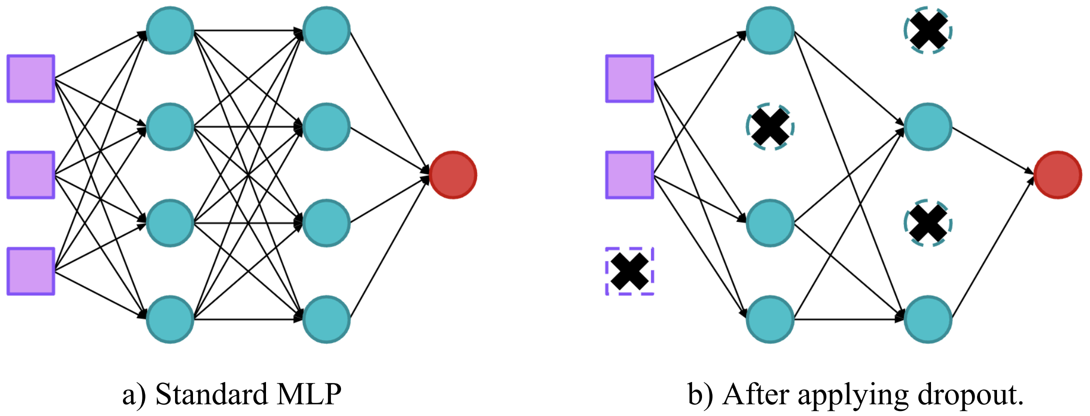


Figure 5: Dropout in MLPs. **Left**: Standard MLP with 2 hidden layers. **Right**: MLP after applying dropout to the network. Crossed units are dropped.

Table 1 shows the hyperparameters of the model that generated the results in this report. While the optimizer, activation function in all layers, and loss function remained the same throughout our experiments, others were tuned more carefully. The number of epochs, for example, was chosen such that accuracy, AUC, and loss stabilized and changed slowly. We found that if the number of epochs was too high (e.g., 500), the model overfitted on the training set and generalized poorly. However, we have not performed thorough hyperparameter optimization, so our results are not comprehensive. More details are provided in section 4.3.

# Epochs	100
Batch Size	1024
Optimizer (Step Size)	Adam (0.0005)
# Nodes in $\mathbf{W}^{(1)}$	256
Dropout Value in $\mathbf{W}^{(1)}$	0.9
# Nodes in $\mathbf{W}^{(2)}$	256
Dropout Value in $\mathbf{W}^{(2)}$	0.9
Activation Function in all Layers	Sigmoid
Loss Function	Binary Cross-Entropy

Table 1: Model hyperparameters that produced the results in this report.

3.4 Comparison and Discussion of Results

The main results of both models (i.e., the evaluation metrics) are summarized in Table 2. It is clear that classification accuracy, while high, was insufficient to gauge the performance of our models. On the other hand, the AUC gave more insight into how our models performed. It was clear both of our models struggled to generalize to the testing set. The poor performance on the testing set suggested that either we needed much more data in general or that we needed a larger testing set in order for the LOO encoding method to work well. If it turns out that having a larger testing set produces similar results to the ones we have, then it may be that LOO encoding is not a method that Fingerhut should pursue. We discuss this topic further in section 4.3.

	Accuracy (Training)	**Accuracy** (Testing)	**AUC** (Training)	**AUC** (Testing)
Logistic Regression	0.94	0.9	0.91	0.59
3-Layer Perceptron	0.91	0.89	0.96	0.73

Table 2: Evaluation metrics of both models for training and testing sets.

Figures 6 and 7 show the confusion matrix and histogram of the outputs, respectively, of the testing set for logistic regression. The histogram suggested that some threshold on the classification probability may be a reasonable approach. In particular, we conjecture that a low probability threshold may produce better results from this model due to the heavy right-skew of the distribution of testing set classification probabilities.

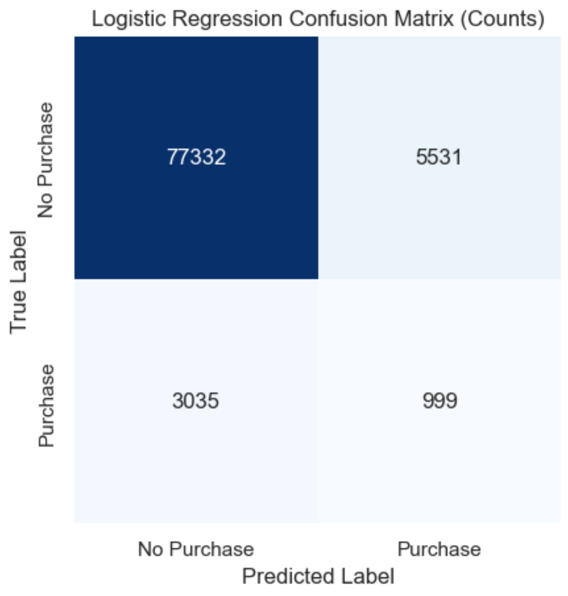


Figure 6: Logistic regression confusion matrix. Darker blue indicates higher counts.

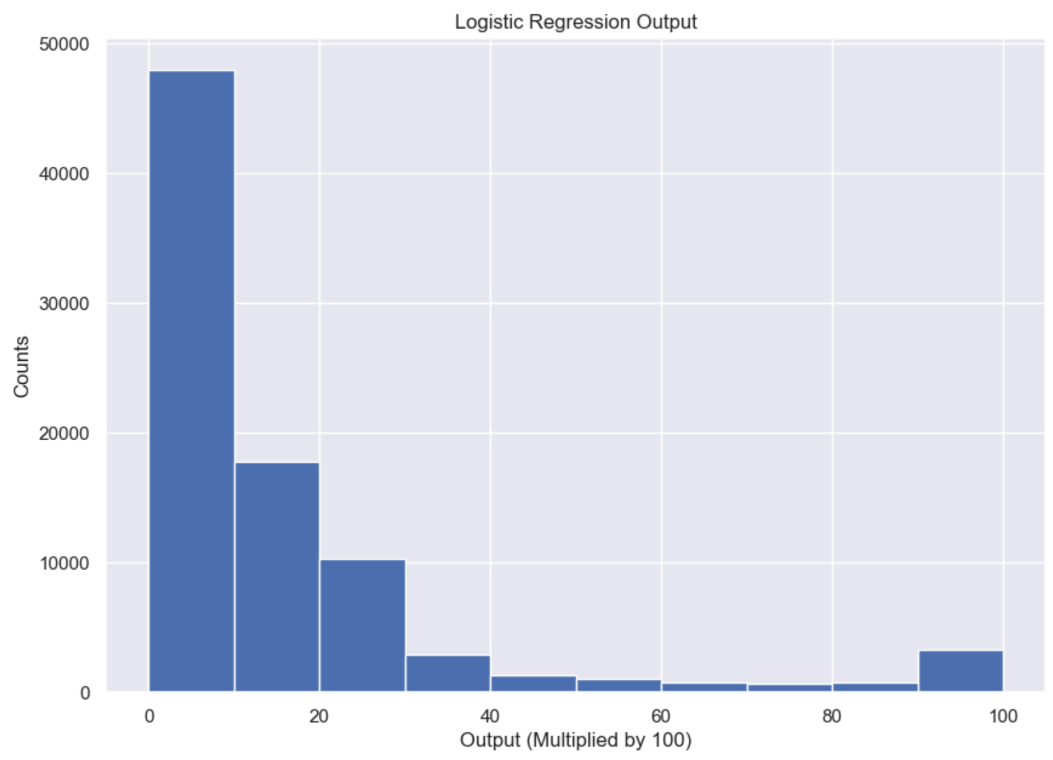
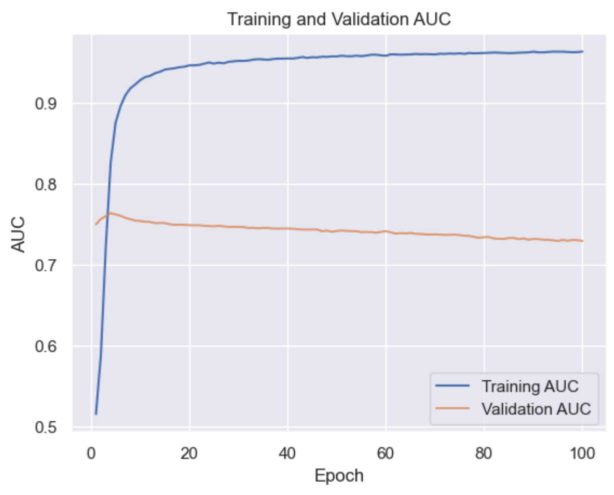
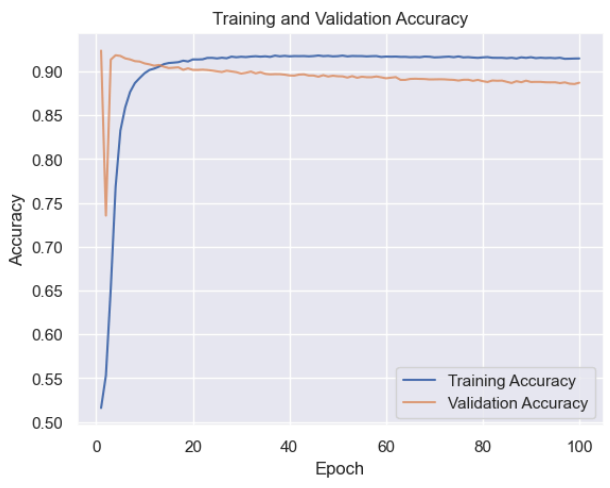


Figure 7: Histogram of logistic regression outputs (multiplied by 100).

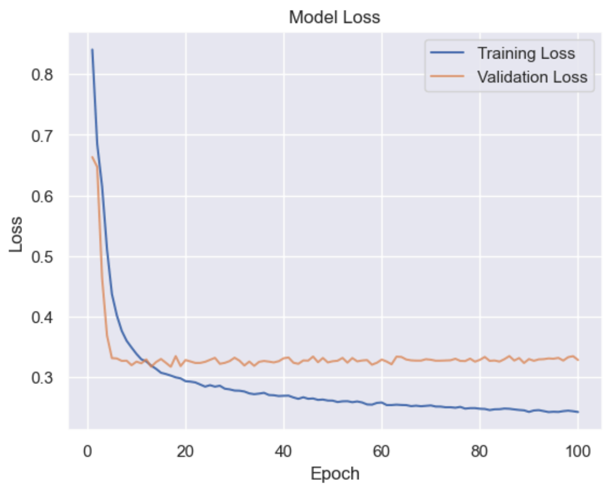
Figure 8 shows the learning curves of neural network training. Note that the testing data performed well for the first few epochs. As the model continued training, the testing AUC and accuracy gradually decreased and the testing loss gradually increased. These results suggested that there was some overfitting, despite using high dropout values. We also observed that the choice of step size had a noticeable effect on how much the testing set accuracy and AUC decreased. Specifically, step sizes larger than 0.00075 (we tested both this and 0.01) resulted in much steeper decreases in AUC (e.g., testing set AUC was as low as 0.62 in some experiments). We conjecture that smaller step sizes stabilized training and reduced the degree to which the model overfit on the training set.



a) Training and validation AUC.



b) Training and validation accuracy.



c) Training and validation loss.

Figure 8: Neural network learning curves.

Figures 9 and 10 show the confusion matrix and histogram of the outputs, respectively, of the testing set for the neural network. Like with the logistic regression results, we observed a heavy right-skew in the distribution of testing set classification probabilities for the neural network, with even more observations in the 0% to 10% range. These results supported our consideration of a low classification probability threshold being needed for the testing set.

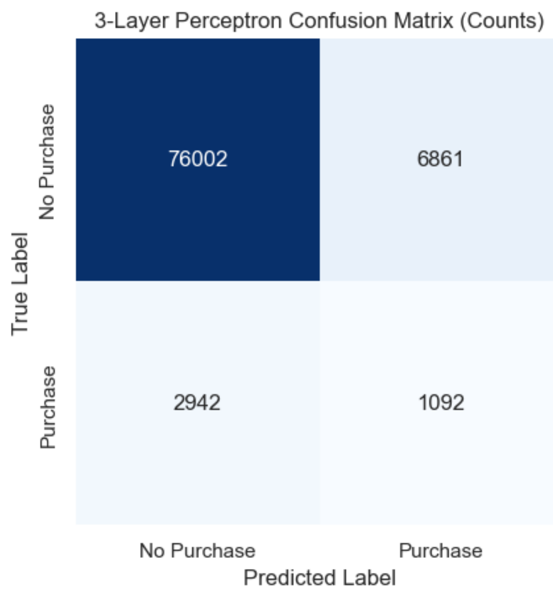


Figure 9: 3-layer perceptron confusion matrix. Darker blue indicates higher counts.

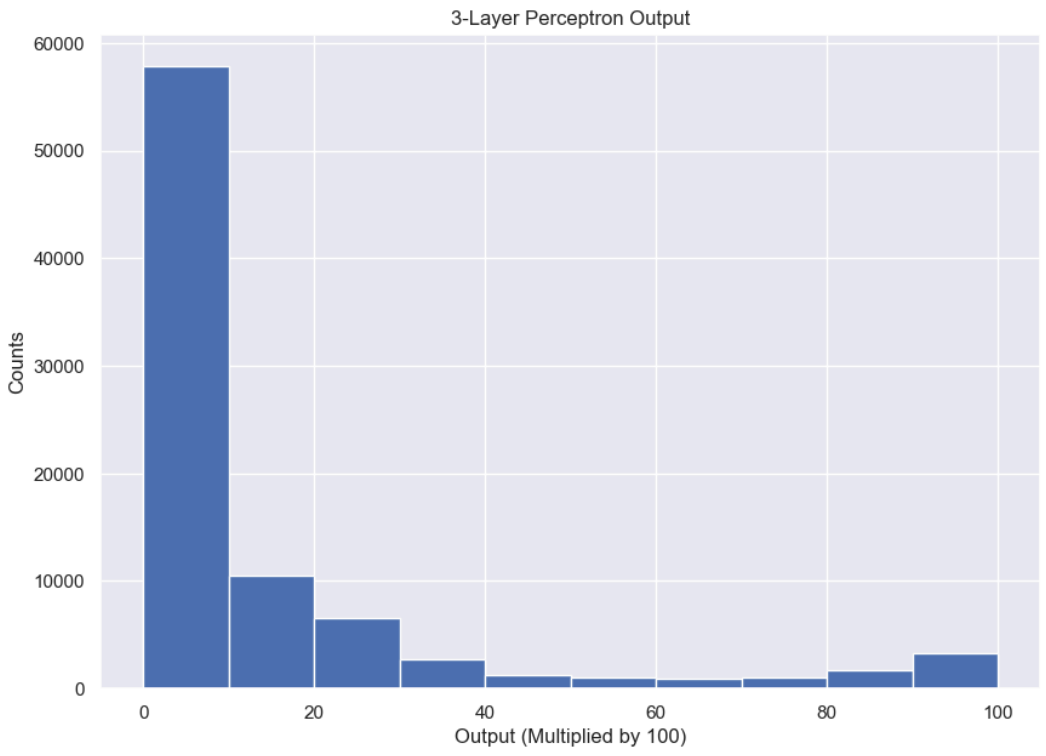


Figure 10: Histogram of 3-layer perceptron outputs (multiplied by 100).

It was clear that both models struggled with the imbalanced data. Despite using class weights, both models still classified most ‘Purchase’ users as ‘No Purchase.’ While the neural network performed more favorably for the ‘Purchase’ class, the improvement in comparison to logistic regression was minor. Furthermore, the histograms of testing set classification probabilities suggest that low probability thresholds might be interesting to pursue. We present an example for the results in this report with Figure 11. Our criterion was a threshold that resulted in roughly 0.5 model accuracy. We chose this value with the assumption (and supported by observation) that most of the classification probabilities for the ‘Purchase’ class were higher than that of the ‘No Purchase’ class. We found a testing set classification threshold of 0.05 was the closest value that met that criterion (due to time constraints, we only incremented the threshold by 0.01). Figure 11 shows how the 0.05 threshold captured most users that made a purchase. We discuss these results, their applications, and their limitations in more detail throughout section 4.

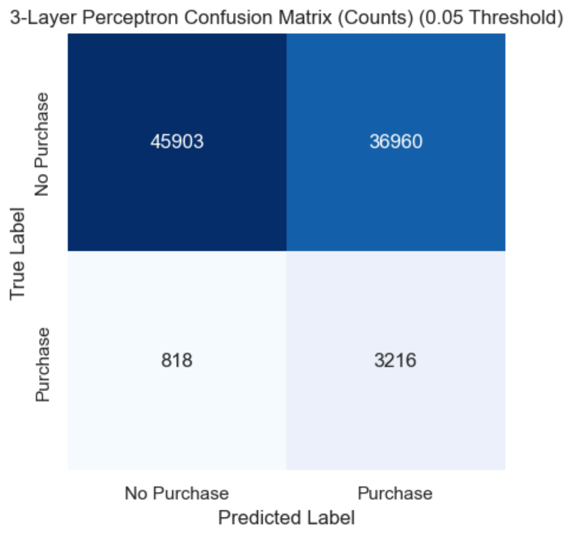


Figure 11: 3-layer perceptron confusion matrix with 0.05 testing set classification probability threshold. Darker blue indicates higher counts.

4 Conclusion

4.1 Recommendations

We present 2 recommendations to Fingerhut. The first is that Fingerhut can plug in data collected in real time to a pre-trained neural network to make predictions for all users visiting the site at any given time. We propose intuitive strategies depending on the prediction of these users.

1. If predicted to purchase, Fingerhut can optimize search results and offer promotional codes based on subsequent page visits to encourage these users to make a purchase.
2. If predicted to not purchase, Fingerhut can show these users a more personalized homepage with targeted item recommendations to encourage them to continue exploring the site.

Since the dataset is highly imbalanced and the distribution of testing set classification probabilities are heavily right-skewed, we suggest using a lower threshold when making predictions. Indeed, as demonstrated by Figure 11, a large portion of users who do not make a purchase will be classified as 'Purchase' with their first click using this approach. However, this approach does classify a majority of users who made a purchase correctly. This is a clear tradeoff and relegates roughly half of all users to being predicted to make a purchase, which may not be ideal for Fingerhut. That being said, actively plugging in data to make predictions in real time is a valuable way to gauge the intent of a user, and Fingerhut can tailor different strategies depending on the prediction. Since Keras allows for neural networks to be saved, Fingerhut can load a model that is already trained with 1 month of data, for example, from the previous year and make predictions for the current year. This is a particularly efficient approach because it saves on computational cost by taking advantage of previous iterations of the model.

The second recommendation is retraining neural networks for each season. The motivation for this is that user behaviors such as the use of promotional codes and categories of items that are frequently purchased likely change from season to season. We intuit that the feature selection procedure can reveal variables that indicate customer behaviors that change throughout the year, which in turn can inform marketing decisions that change with trends in customer behavior. An example is during the spring and summer when people often look for and buy houses. Fingerhut can address this change in behavior by emphasizing ads on social media platforms that offer deals on home and furniture goods. Having a different pre-trained model for each season is a natural extension of the first recommendation, so Fingerhut can capitalize on season-specific user behaviors and make stronger predictions of users visiting the site.

4.2 Limitations

The limitations in our project can be focused into 2 main categories: data and models. The first set of limitations concerned the datasets and the feature selection procedure. Firstly, having only 1 week of data limited the applicability and generalizability of our results. Any potential seasonal variations in customer behavior may not be captured in 1 week. In addition, since our data is drawn from a time of active holiday shopping, variables such as item categories (e.g., toys), promotional codes used, and frequency of purchases may not reflect general user behavior in the rest of the year. Secondly, there were tradeoffs with performing high-level feature selection compared with hand picked features. Specifically, while the features selected may be predictive, they may not be of practical interest to Fingerhut. Hand-picked features may be practically relevant but may also not be predictive in the context of data analysis.

The second set of limitations related to the use of class weights and our models. While the neural network approach offered flexibility, it lacked interpretability. Firstly, using balanced class weights allowed us to consider the data in its entirety, but resulted in our models classifying most of the users who did purchase as not purchasing. An alternative we briefly pursued that may be of interest is under sampling from users that did not purchase. However, addressing this bias was too computationally expensive, as we would have needed to set many random seeds and gauge the sensitivity of our models to different subsets of the under-sampled data. Secondly, neural networks are black-box methods (i.e., the layer weights are basically meaningless). If interpretability is of importance to Fingerhut, then we suggest for the data analysis teams there to focus on other methods (e.g., decision trees).

4.3 Future Work

To address the limitation of using only the first server call for our analysis, we plan to incorporate additional server calls to gather more information as visitors spend more time on the website. We hope that utilizing subsequent server calls can capture more information about visitor behavior and preferences, which can be used to better understand their likelihood of making a purchase. Furthermore, we hope to combine our aforementioned idea of probability thresholding with data consisting of more server calls. It would be interesting to see if using more server calls increases the probability threshold. If it did, then that would support our thinking that subsequent server calls contain more information about user behavior.

We plan to address the model limitations by 1) optimizing the neural network hyperparameters and 2) examining classification probability thresholding. The first task involves conducting experiments on all combinations of hyperparameters (as outlined by Table 3) and finding the one that gives the best performance. Due to time and computing resource constraints, we were unable to carry this out. We also would like to increase the number of hidden layers of the neural network and study its performance. Seeing if increasing the computational cost corresponds to an improvement in model performance is relevant to continuing this project. If, for example, a deeper (i.e., more hidden layers) network performs worse, then that gives some empirical evidence that simpler models are preferable to model this data than more complicated ones. Ultimately, we hope that optimizing the neural network can improve the quality of our analysis and enable us to formulate more informed suggestions based on the insights we gain.

# Epochs	100 250 500 1000
Batch Size	32 64 128 256 512 1024 2048
Step Size	0.0025 0.001 0.0005
# Nodes in $\mathbf{W}^{(1)}$	128 256 512 1024
Dropout Value in $\mathbf{W}^{(1)}$	0.1 0.25 0.5 0.75 0.9
# Nodes in $\mathbf{W}^{(2)}$	128 256 512 1024
Dropout Value in $\mathbf{W}^{(2)}$	0.1 0.25 0.5 0.75 0.9

Table 3: Hyperparameter search range. The values that produced the results in this report are highlighted. Powers of 2 for batch size and number of nodes are general heuristics.

The second task is much more interesting in the context of better understanding how the data transformed with LOO encoding performs. We intend to test different thresholds in combination with larger testing sets in order to study the relationship between probability thresholds, size of the testing set, and LOO encoding. This, however, is a computationally expensive task and requires extensive use of computing resources.

Acknowledgements

We thank Ben Thompson for his helpful clarifications, our professor for his advice and guidance, and our colleagues for their useful suggestions and feedback.

References

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees* (1st ed.). Chapman & Hall.

Hancock, J. T. and Khoshgoftaar, T. M. (2020). ‘Survey on categorical data for neural networks’. *Journal of Big Data*, 7(1), 28. https://doi.org/10.1186/s40537-020-00305-w. Accessed 15 Mar 2023.

Kingma, D. P., and Ba, J. (2015). ‘Adam: a method for stochastic optimization’. *3rd International Conference on Learning Representations* (ICLR). https://doi.org/10.48550/arXiv.1412.6980. Accessed 8 Mar 2023.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). ‘Dropout: A simple way to prevent neural networks from overfitting’. *Journal of Machine Learning Research*, 15(1), 1929-1958. https://jmlr.org/papers/v15/srivastava14a.html. Accessed 12 Mar 2023.

Yanminsun, Wong, A., and Kamel, M. S. (2011). ‘Classification of imbalanced data: a review’. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(4), 687-719. https://doi.org/10.1142/S0218001409007326. Accessed 12 Mar 2023.