Enhancing Restaurant Experience Through Machine Learning: A Predictive Approach to Customer Ratings

Jonathon A. Carl jac16@williams.edu
Computer Science, Mathematics

1. Introduction

In recent years, the restaurant industry has witnessed a paradigm shift in the way consumers interact with dining establishments. The advent of online platforms, coupled with the explosion of data availability, has ushered in an era where understanding and predicting customer preferences play a pivotal role in the success of restaurants. In this context, the use of machine learning techniques has emerged as a powerful tool to unlock valuable insights from the vast troves of data generated by both consumers and restaurants. The ability to predict customer ratings for restaurants based on individual preferences and restaurant attributes holds significant implications for enhancing the overall dining experience. Customers today are presented with an abundance of choices, ranging from culinary styles and ambiance to service quality. The challenge lies in deciphering the intricate interplay of factors that contribute to a customer's satisfaction and their rating of a dining establishment.

Predicting customer ratings in the realm of restaurant and consumer data is a multifaceted problem. Customers bring a diverse set of preferences shaped by factors such as cuisine, dietary restrictions, ambiance preferences, and past dining experiences. Restaurants, on the other hand, vary in their offerings, service quality, and ambiance, creating a complex landscape for understanding customer satisfaction. Machine learning, with its capacity to discern patterns and relationships within vast datasets, offers a promising avenue for addressing this complexity. By leveraging advanced algorithms, machine learning models can analyze data on customer preferences and restaurant attributes, distilling actionable insights that can be employed to predict how certain customers will rank restaurants.

This paper explores the significance of utilizing machine learning methodologies to predict customer ratings in the context of restaurant and consumer data. By doing so, it seeks to contribute to the ongoing dialogue within the hospitality industry on how data-driven insights can be harnessed to tailor offerings, optimize service delivery, and ultimately elevate the overall dining experience. In delving into the exploration of predictive models for restaurant ratings, the aim is to shed light on the transformative potential of machine learning in catering to the ever-evolving expectations of modern consumers. Through a holistic examination of customer preferences and restaurant attributes, this paper seeks to demonstrate the instrumental role that machine learning can play in shaping the future of the dining landscape.

2. Preliminaries

In order to obtain a dataset capable of training a reliable machine learning model, I first needed to handle a missing data problem. *K-Nearest-Neighbors* (KNN) is an unsupervised learning technique which computes the distance between vectors in a d dimensional space $A \subseteq \mathbb{R}^d$ and locates the k nearest neighbors to a vector in A. I used KNN to replace missing values with the weighted average of its k nearest neighbors. Denote our X as our $n \times d$ feature matrix. Let $x, y, z \in A$ be rows of X. Then d is a metric on A if and only if d satisfies the following three properties:

1. $d(x, y) = 0 \iff x = y$,
2. $d(x, y) = d(y, x)$, and
3. $d(x, z) \leq d(x, y) + d(y, z)$.

Together (A, d) form a metric space. Say row x has missing data. Given a metric space (A, d), KNN uses d to compute the distance between x and y for all rows y such that $x \neq y$. Common metric functions are of the Minkowski distance form

$$d(\boldsymbol{x}, \boldsymbol{y}) = (\textstyle\sum_{i=1}^{d} |x_i - y_i|^p)^{1/p} \text{ with } p \in \mathbb{Z}_{>0}.$$

for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$. I use the Euclidean distance metric, which is the Minkowski distance evaluated at $p = 2$. KNN then selects the k nearest neighbors to x and computes a weighted average of those k nearest neighbors in order to impute x. Note that k is a parameter that can be tuned for performance.

After cleaning the dataset and imputing all missing values, I began to use regression models in order to make predictions on restaurant ratings. Formally, I've created a feature matrix $X \in \mathbb{R}^{n \times d}$, a vector of learned parameters $\theta \in \mathbb{R}^{d \times 9}$, and an outcome variable $\boldsymbol{y} \in \mathbb{R}^{n \times 9}$. The first crucial component in to understand in these models is linear regression, which produces predictions for $\boldsymbol{y}$ as a linear combination of X and θ: $\widehat{\boldsymbol{y}} = X\theta$.

The parameters θ can be learned by using the gradient update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla L(\theta^{(t)})$$

for some learning rate $\alpha > 0$; larger learning rates imply "faster" learning, although there is a maximum value of α for which θ may be learned effectively. *Gradient descent* learns θ by minimizing a suitable loss function L. I use the mean squared error, which is defined for n samples of data as follows: $L(\theta) = \frac{1}{n}(\boldsymbol{y} - \widehat{\boldsymbol{y}})^2$ where $\boldsymbol{y} - \widehat{\boldsymbol{y}}$ is element wise subtraction in each column between the predicted and true outcomes. Note that mean squared error is computed independently on each of the nine columns in $\boldsymbol{y}$. That is, $L(\theta)$ is computed on the first column, and then subsequently computed on the remaining columns; each column in θ represent distinct predictions.

A loss function is *Lipschitz continuous* if there exists some real number $K > 0$ such that for any two points $\theta^{(1)}$ and $\theta^{(2)}$ we have that

$$||\nabla L(\theta)^{(1)} - \nabla L(\theta)^{(2)}|| \leq K||\theta^{(1)} - \theta^{(2)}||.$$

Given a twice differentiable and Lipschitz continuous loss function, $L(\theta^{(t+1)}) \leq L(\theta^{(t)})$ for any $0 < \alpha \leq 1/K$. Hence, gradient descent is *indeed* a descent algorithm with a suitable α and terminates once a maximum number of iterations is reached or the loss is strictly less than some predetermined tolerance. Note that α and the number of iterations in gradient descent are both hyperparameters that may be tuned for performance with a validation set.

A model is said to *overfit* the data when the model performs with high accuracy on testing data, but does not generalize well to validation or testing data. To avoid overfitting, it is common to apply a regularization penalty in conjunction with the loss: I use the L2 penalty. So, the overall function being minimized is then $L(\theta) + \lambda \sum_{j=1}^{d} \theta_d^2$, where λ is a hyperparameter that will be tuned using a validation set. Again, the L2 penalty is computed sequentially column-wise on θ: the loss and regularization penalty is first computed on the first column. This computation is repeated on remaining columns of θ. Thus, the loss function yields nine losses, one for each of the outcomes in θ. Gradient descent returns the optimal set of learned parameters that may be used for predictions on validation and testing data before being deployed.

Gradient descent is the foundation for the models I use in my analysis. To grasp the first model I'll be using, I'll need to provide some background on graphs. Let $\mathcal{G} = (V, E)$ be an undirected graph with vertex set V and edge set E. Then $\mathcal{G}$ is a tree if and only if

1. $\mathcal{G}$ is connected - there exists a path between any two vertices $u, v \in V$, and
2. $\mathcal{G}$ is connected - there is no path from u to u.

For ease of interpretation, I'll treat $\mathcal{G}$ as if the edges have been oriented from now on. The same principles hold from above, but edges are now directed in $\mathcal{G}$. If $\mathcal{G}$ has one designated root node with no parents and all other nodes have at most one parent, then $\mathcal{G}$ is an *arborescence*. The leaves of $\mathcal{G}$ have no children and are defined as *leaf nodes*.

A *regression tree* is a supervised learning technique that produces a continuous-valued output by shifting a data point from the root of an arborescence to a leaf node via a set of rules that compares a data point to threshold feature values (Breiman et al., 1984). If the data point is larger than the threshold values at $v \in V(\mathcal{G})$, it moves to the right child of v, otherwise it moves to left child of v.

Given an outcome $\boldsymbol{y} \in \mathbb{R}^{n \times 9}$, a matrix of input features $X \in \mathbb{R}^{n \times d}$, and a vector of parameters $\theta \in \mathbb{R}^{d \times 9}$, a regression tree is constructed via a recursive splitting algorithm. The procedure first determines all possible splits on features and possible values for those features. For continuous-valued features, the regression tree only seeks to split on a small subset of possible values; one approach is to select all unique values present for the feature in the dataset. After finding all possible feature splits, the regression tree calculates the mean squared error for each possible split. Calculating the mean squared error is defined as follows for a dataset $[D]$ and mean true outcome $\overline{y}$ associated with the rows of $[D]$:

$$\text{MSE}([D]) = \frac{1}{\text{len}([D])} \sum_{i=1}^{\text{len}([D])} (y_i - \overline{y})^2.$$

The algorithm selects the feature and threshold combination which minimizes the weighted average of the mean squared error of $[D]$ being split into two datasets $[D]^{s1}$ and $[D]^{s2}$:

$$\text{MSE}([D]^{s1}, [D]^{s2}) = \frac{\text{len}([D]^{s1})MSE([D]^{s1}) + \text{len}([D]^{s2})MSE([D]^{s2})}{\text{len}([D]^{s1}) + \text{len}([D]^{s2})}.$$

All samples in X whose data points for the given feature(s) are greater than the threshold responsible for the split lie in $[D]^{s2}$, while all other samples lie in $[D]^{s1}$. The algorithm recursively splits these datasets until a dataset can be split no further or the maximum depth specified for $\mathcal{G}$ has been reached. When a node can split no further, the node becomes a leaf in the decision tree. Predictions are made at leaves by finding the mean prediction among the outcomes associated with the dataset at a given leaf. Note that the maximum depth of $\mathcal{G}$ is a hyperparameter than can be tuned for performance.

A *random forest* is a collection of *bagged* regression trees. Formally, *bagging* is a technique which reduces the variances of models. The bias-variance tradeoff confirms this results since a collection of ensemble models whose averaged predictions approximately approach the true mean prediction. Given a dataset $[D] \in \mathbb{R}^{n \times d}$, the random forest algorithm samples m datasets $[D]_1, \ldots, [D]_m \in \mathbb{R}^{n \times d}$ from $[D]$ with replacement. Then, for each $[D]_i$, a decision tree of unlimited depth is trained. When training the bootstrapped datasets, a random subsample $k \leq d$ of features is selected; these are the only features considered for the split at a given node. The final aggregated model is the mean predictions from the m individual trees. Note that both m, k are hyperparameters that can be tuned for performance with a validation set.

A *neural network* is ...

3. Data

I obtained the data via the UCI Machine Learning Repository (Medelln and Serna, 2012). The data was structured into nine distinct CSVs. Five CSVs consisted of data on particular restaurants, while another three CSVs consisted of customer preference data; the last CSV was the final ratings of customers on restaurants–overall, service, and food ratings. I used Pandas–a Python data manipulation library–for data preprocessing (Wes McKinney, 2010).

All of the data in the CSVs was categorical aside from the final ratings. Hence, a large amount of the preprocessing effort was spent in relabeling categorical values into numerical values. For clarity, I've renamed all variables beginning with U (for user) to begin with C (for customer). All features beginning with C are related to a customer's preferences; features beginning with R relate to a restaurant's attributes. Note that Rcuisine $\subseteq$ Ccuisine and that Ccuisine is relabeled to numerical values analogously to all other restaurant and customer features in Table 2 and Table 3. Similarly, Cpayment $\subseteq$ Rpayment and Rpayment is mapped to numerical values in the same manner as Ccuisine.

After relabeling the data, I merged all nine CSVs into a single Pandas dataframe. Each row contains a single customer and restaurant, along with corresponding customer preferences and restaurant features. Additionally, each row contains the final ratings provided by the customer on the restaurant. Note that there may be many rows which contain the same customer. In other words, there exist customers in the dataframe who rank multiple restaurants; conversely, restaurants may be ranked by multiple customers. This implies

that the *Independent and Identically Distributed* (IID) assumption of regression models is fundamentally violated: a customer's preferences propagates through to distinct restaurant ratings. That is, predicted ratings are not necessarily independent since two ratings may be produced by the same customer (which has the same set of preferences); a customer may choose to rank two restaurants differently based on their underlying preferences.

In order to ensure that the data used in this analysis obeys the IID assumption for regression tasks, I chose the minimum number of ratings (3) that all customers in the dataframe made on a number of restaurants. Choosing this minimum number ensures that each customer in the dataframe is equally represented in the sense that every customer makes the same number of ratings. After obtaining an equal number of ratings for each customer, I altered the dataframe by assigning a single row to a customer. In order to retain all three ratings for each customer, I created three copies of restaurant features as columns in the dataframe. That is, each row has columns for the preferences of the customer index, columns for the restaurant features and ratings for restaurant one, followed by columns for the features and ratings for restaurants two and three, respectively. With this manipulation of the dataframe, every observation in the dataset is independent since no row contains a duplicate customer. The preferences of any customer is only observed in the row where the customer is indexed.

One concern I still I have about the data is the Identically Distributed assumption. The dataset contains many possible values for Ccuisine and Rcuisine. However, there may be preferences in cuisine that are much more rare than others. This may lead to outliers in the dataset that a machine learning model may struggle to correctly predict. One idea I have to deal with the violation of the identically distributed assumption is to stratify the training and testing data by cuisine type. Stratification of the data ensures an even distribution of cuisine types across customers and restaurants in the training and testing data used in the machine learning models. After successfully stratifying the data by cuisine, I believe the data will be IID. Of course, the data is only IID under the assumption that customers were randomly sampled and had no impact on another customer's ratings or preferences. The data was originally used for research purposes at the National Center for Research and Technological Development (CENIDET) in Mexico.

After organizing the dataset in a manner that satisfies the IID assumption, I noticed there still remained a few missing values in the dataset. These missing values ranged from restaurant attributes to customer preferences. I used KNN (Section 2) in order to impute these missing values. I then split my dataset into training, validation, and testing data using scikit-learn (Pedregosa et al., 2011).

4. Training And Validation Of Models

The first model I used was a regression tree from scikit-learn (Pedregosa et al., 2011). I performed a grid search on the maximum depth and maximum number of features considered for a split in order to obtain the optimal hyperparameters for the model. The grid search enumerates all possible permutations of the hyperparameters and returns optimal hyperparameter values that maximize validation accuracy. These hyperparameters are `'max_depth': None, 'max_features': 'log2'`.

I then used a random forest from scikit-learn (Breiman, 2001). Again, I performed a grid search to determine the optimal hyperparameters. These hyperparameters are: `'max_depth' = None, 'max_features': 'log2', 'n_estimators': 150`.

The regression tree model's predictive accuracy can be seen in Figure 1 and Table 1. Since I have three output classes, it performs marginally better than a naive classifier (which would predict every outcome as the same number). The performance by the random forest was at par with (marginally worse than) the single regression tree (results in Section 8, Figure 2). The concerning statistics here are that R-squared is negative in both the training and validation sets.

In a deeper dive, I discovered that Ccuisine is not normally distributed. Over 80% of customers preferred Mexican cuisine to any other cuisine (the data was sampled in Mexico). I first tried binarizing Ccuisine to "Mexican" and "Other"; I also tried excluding it from the model completely. Both of these approaches only helped increase training accuracy marginally. I plan to study the distribution of my feature variables closer to understand which features may be less valuable to include in my future models.

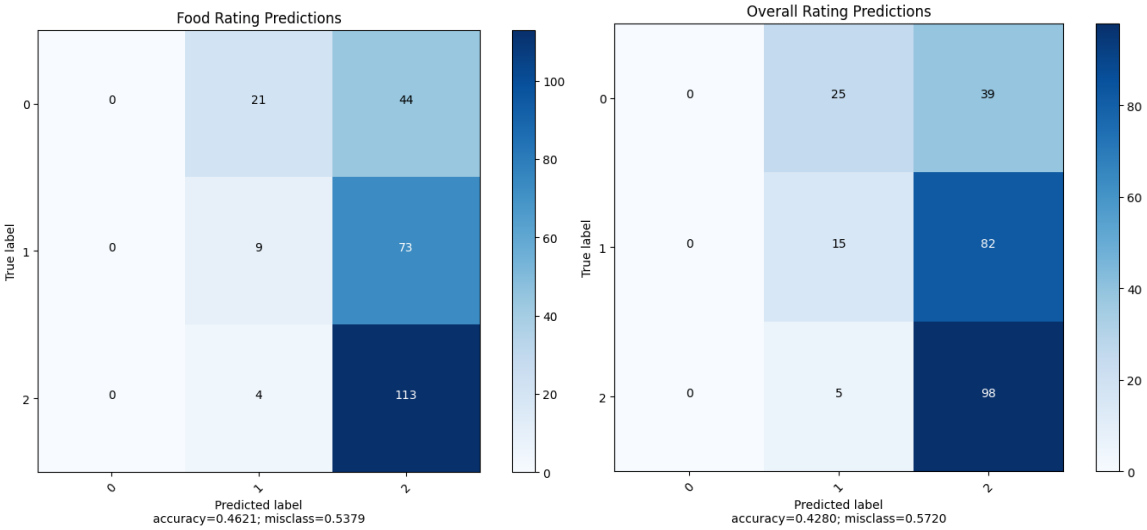
	Training MSE	Training R-squared	Validation MSE	Validation R-squared
Regression Tree	1.0846	-0.8019	1.0253	-0.6998
Random Forest	1.3333	-1.2076	1.1465	-0.9010

Table 1: Metrics on Regression Tree and Random Forest models

The last model I'll use is a neural network from Tensorflow (Abadi et al., 2015). The input layer will contain all the features from the feature matrix. The output layer will contain three sub-layers, each with three neurons. Each sub-layer represents one of the outcome variables (overall, food, and service rating); there are exactly three neurons in each sublayer, corresponding to the three restaurant rankings provided by each user per row in the dataframe. I plan to do hyperparameter optimzation via grid search in order to determine the optimal number of hidden layers, sizes of layers, activation functions, etc.

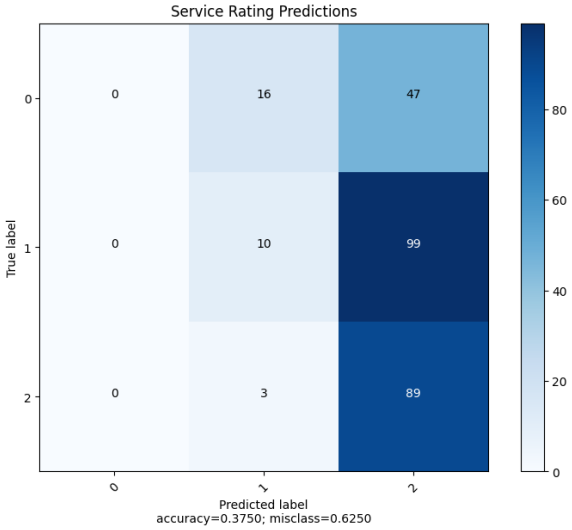
5. Results

I have yet to test my data on the testing set. I'm still working on maximizing testing and validation accuracy for my current models.



(a) Food

(b) Overall



(c) Service

Figure 1: Food, Overall, and Service Accuracy (Decision Tree)

6. Ablation Study

7. Discussion and Conclusion

8. Appendix

Table 2: Relabeled Values for Restaurant Features

Variable	**Relabeled Values**
Rparking	none → 0
	free → 1
	street → 1
	public → 1
	yes → 1
	valet parking → 2
	validated parking → 2
	fee → 2
Rcuisine	...
Rpayment	...
Ralcohol	No_Alcohol_Served → 0
	Wine-Beer → 1
	Full_Bar → 2
Rsmoking	none → 0
	not permitted → 0
	only at bar → 1
	section → 1
	permitted → 1
Rdress	informal → 0
	casual → 1
	formal → 2
Raccessibility	no_accessibility → 0
	partially → 1
	completely → 2
Rprice	low → 0
	medium → 1

Continued on next page

Table 2 – *Continued from previous page*

Variable	**Relabeled Values**
	high → 2
Rambience	quiet → 0
	familiar → 1
Rfranchise	f → 0
	t → 1
Rarea	closed → 0
	open → 1
Rotherservices	none → 0
	internet → 1
	Internet → 1
	variety → 2

Table 3: Relabeled Values for Customer Features

Variable	**Relabeled Values**
Cpayment	...
Ccuisine	...
Csmoker	false → 0
	true → 1
Cdrink	abstemious → 0
	social drinker → 1
	casual drinker → 2
Cdress	no preference → 0
	informal → 1
	formal → 2
	elegant → 3
Cambience	solitary → 0
	family → 1
	friends → 2
Ctransport	on foot → 0

Continued on next page

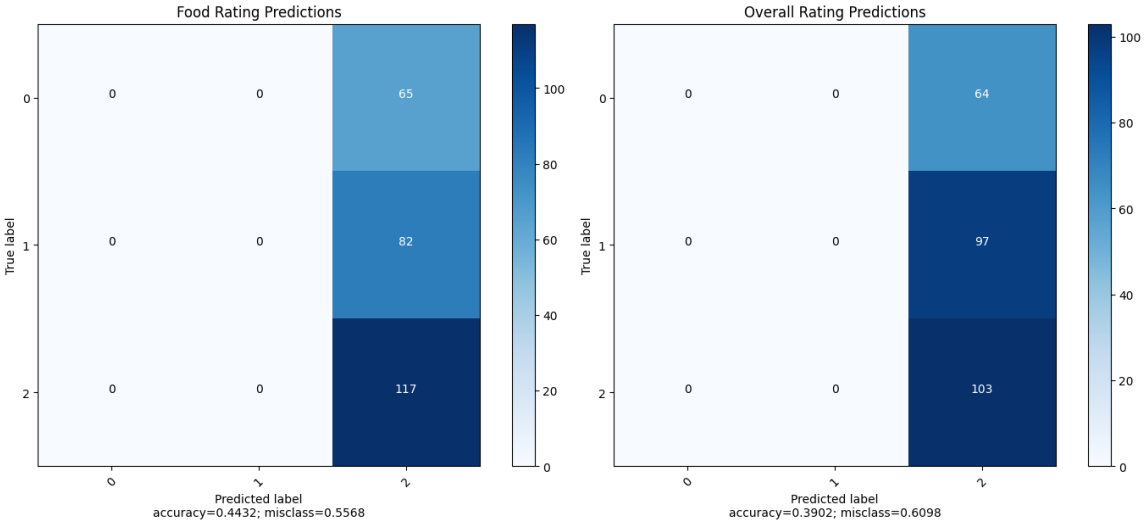
Table 3 – *Continued from previous page*

Variable	**Relabeled Values**
	public → 1
	car owner → 2
Cmaritalstatus	single → 0
	married → 1
	widow → 2
Cchildren	dependent → 0
	independent → 1
	kids → 2
Cinterest	none → 0
	retro → 1
	technology → 2
	eco-friendly → 3
	variety → 4
Cpersonality	conformist → 0
	thrifty-protector → 1
	hard-worker → 2
	hunter-ostentatious → 3
Creligion	none → 0
	Catholic → 1
	Christian → 2
	Mormon → 3
	Jewish → 4
Cactivity	unemployed → 0
	student → 1
	working-class → 2
	professional → 3
Ccolor	black → 0
	red → 1
	blue → 2
	green → 3
	purple → 4
	orange → 5

Continued on next page

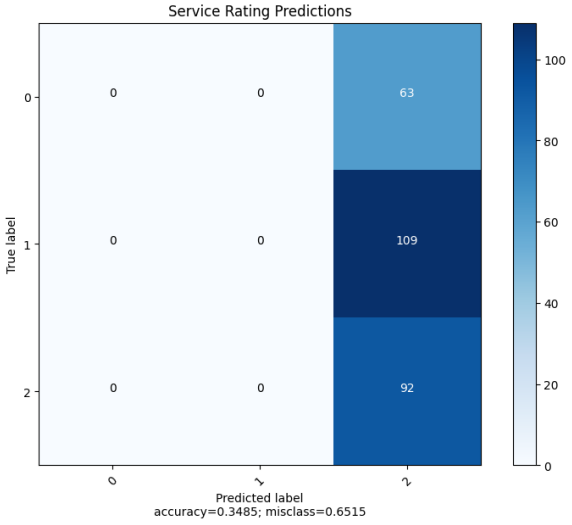
Table 3 – *Continued from previous page*

Variable	**Relabeled Values**
	yellow → 6
	white → 7
Cbudget	low → 0
	medium → 1
	high → 2



(a) Food

(b) Overall



(c) Service

Figure 2: Food, Overall, and Service Accuracy (Random Tree)

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and Regression Trees*. Routledge, 1984.

Rafael Medelln and Juan Serna. Restaurant consumer data. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C5DP41.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.