Enhancing Restaurant Experience Through Machine Learning: A Predictive Approach to Consumer 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 consumer 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 consumer ratings for restaurants based on individual preferences and restaurant attributes holds significant implications for enhancing the overall dining experience. Consumers today are presented with an abundance of choices, ranging from culinary styles and ambiance to service quality. The challenge lies in deciphering the factors contributing to a consumer's satisfaction and their rating of a dining establishment.

This paper leverages existing machine learning techniques like gradient-based and tree-based learning in order to make predictions about consumer ratings based upon a set of consumer preferences and restaurant attributes. I will use different variations of regression in order to predict outcomes, and then use thresholds to match training and validation outcomes to the true empirical distribution of the data. In this way, the problem is transformed from regression to classification in order to obtain ratings on an ordinal scale.

The aim of this paper 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 consumer preferences and restaurant attributes, I hope to demonstrate the role machine learning can play in the future of the dining landscape. (OpenAI, 2023)

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 $X \in \mathbb{R}^{n\times d}$ as our feature matrix. Let $x, y, z \in A$ be rows of X. Then d is a metric if and only if

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

Common metrics are of the Minkowski 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 metric, which is the Minkowski metric 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 tuned for performance.

After cleaning the dataset and imputing all missing values, I then normalized the data by computing the mean μ_X and standard deviation σ_X of X:

$$X = \frac{X - \mu_X}{\sigma_X}.$$

Normalizing the feature matrix allows all features to be distributed on the same scale and can help with stability in training my models. Crucially, it's important to only scale feature matrices based upon the training mean and standard deviation to avoid overfitting.

After completing my data preprocessing, I began to use regression models in order to make predictions on restaurant ratings. Formally, I want to produce some prediction vector $\widehat{\boldsymbol{y}} \in \mathbb{R}^{n \times 1}$ by taking a linear combination of our feature matrix X and vector of learned parameters $\theta \in \mathbb{R}^{d \times 1}$: $\widehat{\boldsymbol{y}} = X\theta + \boldsymbol{\epsilon}$.

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

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

for some tunable 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: $L(\theta) = \frac{1}{n}(\boldsymbol{y} - \widehat{\boldsymbol{y}})^2$.

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.

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 $\mathcal{G}$ 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.

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 variance of models. The bias-variance tradeoff confirms this results since ensemble models have average predictions which 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 each of the m trees. Both m, k are hyperparameters tuned for performance with a validation set.

A *One-Versus-Rest* model is a model which trains k random forest binary classification models (1 for each outcome label). If a random forest model is trained on classifier k, then the model is trained as if it was a binary classification task: outputs are 1 if the label is k and 0 otherwise. Hence, three models are fit (one for each label in our dataset). In the prediction phase, a majority voting scheme is used for combining the predictions from each random forest. The model chooses the most-likely label for a given row based on the predictions of all three random forests.

In Section 6, I used a *fixed effects* model as an alternative approach the preference problem I study in this paper. Fixed effects is a type of regression used by economists which enables one to control for attributes of some entity that are fixed over time. In my problem, fixed effects are particularly insightful since both consumer preferences and restaurant attributes are unchanging in our dataset; hence, they are both fixed effects. In this research problem, I'll control for all things that are fixed with respect to consumers and restaurants—including unmeasured features in the model. My fixed effects model is

$$\widehat{\boldsymbol{y}} = \boldsymbol{X\theta} + \boldsymbol{U\alpha} + \boldsymbol{R\beta} + \boldsymbol{\epsilon},$$

where

> $\boldsymbol{U}$ is the fixed effects dummy matrix for consumers. Each column represents a unique consumer and each row is an observation from our dataset. The entry at the i^{th} row and j^{th} column is 1 if the row i is an observation from consumer j, and 0 otherwise,
>
> $\boldsymbol{\alpha}$ are learnable consumer fixed effects parameters,

$\boldsymbol{R}$ is the fixed effects dummy matrix for restaurants. Each column represents a unique restaurant and each row is an observation from our dataset. The entry at the i^{th} row and j^{th} column is 1 if the row i is an observation from restaurant j, and 0 otherwise,

$\boldsymbol{\beta}$ are learnable restaurant fixed effects parameters, and

$\boldsymbol{\epsilon}$ are error terms,

In order to quantify the predictive power of my machine learning models, I use a few different metrics called *Precision*, *Recall*, *F1 Score*, and *Accuracy*. Define TP, FP as the number of predicted true and false positives, respectively; define TN, FN as the number of predicted true and false negatives, respectively. Then

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1 Score} := \frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}, \text{ and}$$

$$\text{Accuracy} := \frac{\sum_{i=1}^{n} \mathbb{I}(y_i = \widehat{y_i})}{n}.$$

Note that precision is particularly insightful when one cares about the false positive rate; a higher precision rate implies a lower false positive rate. Recall is useful when one cares about the false negative rate; a higher recall rate implies a lower false negative rate. However, there is often a tradeoff between precision and recall. In this way, the F1 score is a valuable metric that finds the balance between precision and recall. Accuracy is a crude metric which only provides binary answers about our predictions. In this paper, we'll try to maximize precision since we want to minimize the number of times we recommend a restaurant to a consumer that they *dislike*; it is much better to *not* recommend a restaurant that a consumer may like.

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 restaurant attributes, while another three CSVs consisted of consumer preference data; the last CSV was the final ratings of consumers 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 consumer). All features beginning with C are related to a consumer's preferences; features beginning with R relate to a restaurant's attributes. Note that Ccuisine is relabeled

to numerical values analogously to all other restaurant and consumer features in Table 2 and Table 3; Cpayment 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 consumer and restaurant, along with corresponding consumer preferences and restaurant features. Additionally, each row contains the final ratings provided by the consumer on the restaurant. Note that there may be many rows which contain the same consumer. In other words, there exist consumers in the dataframe who rate multiple restaurants. This implies that the *Independent and Identically Distributed* (IID) assumption of regression models is fundamentally violated: a consumer'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 consumer (which has the same set of preferences); a consumer 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 decided to choose a single restaurant per consumer. So, we have one set of restaurant attributes and one set of unique consumer preferences per row in the dataset. This simple choice yields IID data, yet also notably cuts down the number of observations in the dataset. I initially tried preserving all rankings of consumers by providing 3 distinct sets of restaurant attribute columns. While this layout of the dataset obeys the IID assumption, it proved difficult to have a matrix of outcomes, rather than a vector. Thus, I made the decision to simplify the problem by choosing only one rating per consumer, yielding my final feature matrix.

After organizing the dataset in a manner that satisfies the IID assumption, I noticed there were still a few missing values in the dataset. These missing values ranged from restaurant attributes to consumer preferences. I used KNN (Section 2) with $k = 2$ 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). The splits are 64% training, 16% validation, and 20% testing, respectively. Lastly, I normalized all of the feature matrices according to the mean and standard deviation of the training feature matrix in order to avoid overfitting.

4. Training And Validation Of Models

Note that each model trained in this paper is imported from scikit-learn (Pedregosa et al., 2011). Furthermore, the mechanisms of each model can be seen in more detail in Section 2.

The first model I used was a simple linear regression with default parameters. I then used a ridge regression, which employs L2 regularization to linear regression. I performed a grid search on the regularization penalty and solver. Grid search creates a model for all possible permutations of the hyperparameters. After creating and validating a model with every unique set of hyperparameters, grid search returns optimal hyperparameters that maximize validation accuracy. The hyperparameters for the ridge regression model are `'alpha' : 200.0, 'solver' : 'lsqr'`.

The next model I used was a regression tree. 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. These hyperparameters are
`'max_depth': None, 'max_features': 'log2'`.

I then used a random forest. Again, I performed a grid search to determine the optimal hyperparameters. These hyperparameters are: `'max_depth' = None, 'max_features': 'log2', 'n_estimators': 150`.

I also used a One-Versus-Rest (OvR) model with a random forest regressor for each binary classification model. I used grid search to determine the optimal hyperparameters for the OvR model. These hyperparameters are
`'max_depth' : None, 'max_features' : 'log2', 'n_estimators' : 100`.

Lastly, I trained a fixed effects model which will be used in Section 6.

Each of the model's validation metrics can be seen in Table 4. The regression forest and OvR perform best with a validation accuracy of 0.5; they also have poor metrics for `Rating=0`. The L2 model performs next best with a validation accuracy of 0.45. All of the models perform better than both linear regression (validation accuracy of 0.22) and a naive classifier (validation accuracy of 0.33). In choosing a final model, I would proceed with the regression forest which has the joint-best validation accuracy and the best precision scores.

5. Results

After testing all of the trained models with optimal hyperparameters, the testing results in Table 1 show that providing recommendations to consumers based on preferences is a non-trivial task. The regression forest was the best performing model, but still does not perform well in predicting restaurants that consumers *will not* enjoy (i.e. precision for the outcome 0 *is* 0). A further breakdown of the regression forest predictions can be seen in Figure 1. In Section 1, I discussed that this was of the utmost importance for any model used in creating restaurant recommendations; providing *bad* recommendations to consumers can be dangerous as it provides a direct cost to the consumer adhering to the model's advice.

The regression forest performs as if the problem has been reduced to a binary classification task that identifies *how much* a consumer would enjoy a restaurant (and completely disregards if a consumer may dislike the restaurant). Although 60% accuracy is better than a naive baseline's predictive accuracy of 33% and a linear regression's predictive accuracy of 25%, the regression forest is not incredibly useful for providing valuable restaurant recommendations to consumers. There is work to be done in developing a model capable of performing well on this consumer preference data.

The validation and testing accuracies were not as high as I had hoped. In a deeper dive into the data, I discovered that Ccuisine is not uniformly distributed. Over 80% of consumers 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. In this way, I fear my models will not generalize to different datasets well; there will be a huge distribution shift problem. One way to have stability to a distribution shift is to sample more representative data so models learn from a wider array of preferences.

Distribution shift is the quintessential problem plaguing machine learning models used to provide recommendations: preferences are often related to geographical location. Hence,

any model trained on homogeneous preferences is likely to generalize poorly on recommendation tasks. Although I did not have access to other data, I would be interested to test my models on a different dataset; I suspect my hypothesis would hold true since different datasets would present distribution shifts to my models.

	Metrics for Ratings			**Overall Accuracy**
	Precision	Recall	F1 Score	
Linear Regression				0.250
`Rating=0`	0	0	0	
`Rating=1`	0.538	0.875	0.667	
`Rating=2`	0	0	0	
Linear Regression (L2)				0.500
`Rating=0`	0	0	0	
`Rating=1`	0.538	0.875	0.666	
`Rating=2`	0	0	0	
Regression Tree				0.357
`Rating=0`	0	0	0	
`Rating=1`	0.533	0.500	0.516	
`Rating=2`	0.200	0.333	0.250	
Regression Forest				0.607
`Rating=0`	0	0	0	
`Rating=1`	0.615	1	0.761	
`Rating=2`	1	0.167	0.286	
OvR				0.536
`Rating=0`	0	0	0	
`Rating=1`	0.600	0.750	0.666	
`Rating=2`	0.375	0.500	0.429	
Fixed Effects				0.382
`Rating=0`	0.203	0.267	0.506	
`Rating=1`	0.385	0.417	0.404	
`Rating=2`	0.352	0.400	0.449	

Table 1: Testing Metrics for Overall Rating

6. Ablation Study

The consumer-producer preference problem I'm studying in this project is a popular task performed in many other fields, especially economics. One alternate approach to the problem is a fixed effects model (Section 2).

In order to use a fixed effects model, I had to retransform my dataset into a panel structure which was indexed by consumers and restaurants; note that there are now multiple rows containing the same set of user preferences. Performing this transformation jeopardizes the IID assumption; however, the fixed effects model I use will *somewhat* counteract these dependencies within the dataset. All dependencies are related to restaurant attributes and consumer preferences, which I'm controlling for in the model.

I used Stata (StataCorp, 2023) and `reghdfe` (Correia, 2016) in order to perform my fixed effects analysis. Once I obtained the coefficient estimates from Stata, I transferred the estimates to Python and manually computed predictions given training and validation features. The validation and testing results can be seen in Table 4 and Table 1. The fixed-effects model performs better than the regression tree, but worse than all other models

One possible explanation for the differences in performance is that random forests are non-parametric models, while the fixed effects model is parametric. This means that random forests may become arbitrarily complex. Since random forests are bagged regression trees, they are less prone to overfitting–even though they are complex. In comparison, the fixed effects model is probably too simple for predicting consumer rankings. The fixed effects model may perform better if I had access to more features that varied over time; most of the features in my dataset were captured in the fixed effects term, which means that the fixed effects model did not have many non-fixed predictors to learn from. Still, the models used in this paper were able to generalize better than a two-way fixed effects model.

7. Conclusion

Despite an underwhelming performance from my models, I am satisfied with my analysis on predicting restaurant ratings based on consumer preferences. This is an area of research that has much room for growth. In the future, I would be interested in using an unsupervised approach to the problem. In particular, I feel that using a clustering algorithm may prove useful in predicting ratings for restaurants. At a high level, if two restaurants are *similar*, then the ratings of those restaurants provided by a consumer should also be *similar*.

Additionally, I also feel that using stable matching (Gale and Shapley, 1962) would provide a different approach to the problem that may provide better results; with stable matching, recommendations would only be provided if a consumer is guaranteed to "like" a restaurant. In some scenarios, no stable matching would exist, and hence no recommendation would be provided. This approach aligns exactly with my desire to avoid *bad* recommendations to consumers.

There are many directions of future work that may be taken to investigate this complex relationship between preferences and ratings more closely. This paper illustrates that the recommendation problem at hand is non-trivial. Any future work will require careful analysis and decision making in order to make meaningful progress at developing a model that remains stable amid distribution shift from consumer preferences.

References

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

Sergio Correia. Linear models with high-dimensional fixed effects: An efficient and feasible estimator. Technical report, 2016. Working Paper.

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962. ISSN 00029890, 19300972. URL `http://www.jstor.org/stable/2312726`.

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

OpenAI. ChatGPT (Feb 13 version). `https://chat.openai.com`, 2023. Large language model.

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.

StataCorp. *Stata Statistical Software: Release 18*. StataCorp LLC, College Station, TX, 2023.

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.

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
	high → 2
Rambience	quiet → 0

Continued on next page

Table 2 – *Continued from previous page*

Variable	**Relabeled Values**
	familiar → 1
Rfranchise	false → 0
	true → 1
Rarea	closed → 0
	open → 1
Rotherservices	none → 0
	internet → 1
	Internet → 1
	variety → 2

Table 3: Relabeled Values for consumer 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
	public → 1
	car owner → 2

Continued on next page

Table 3 – *Continued from previous page*

Variable	**Relabeled Values**
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 yellow → 6

Continued on next page

Table 3 – *Continued from previous page*

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

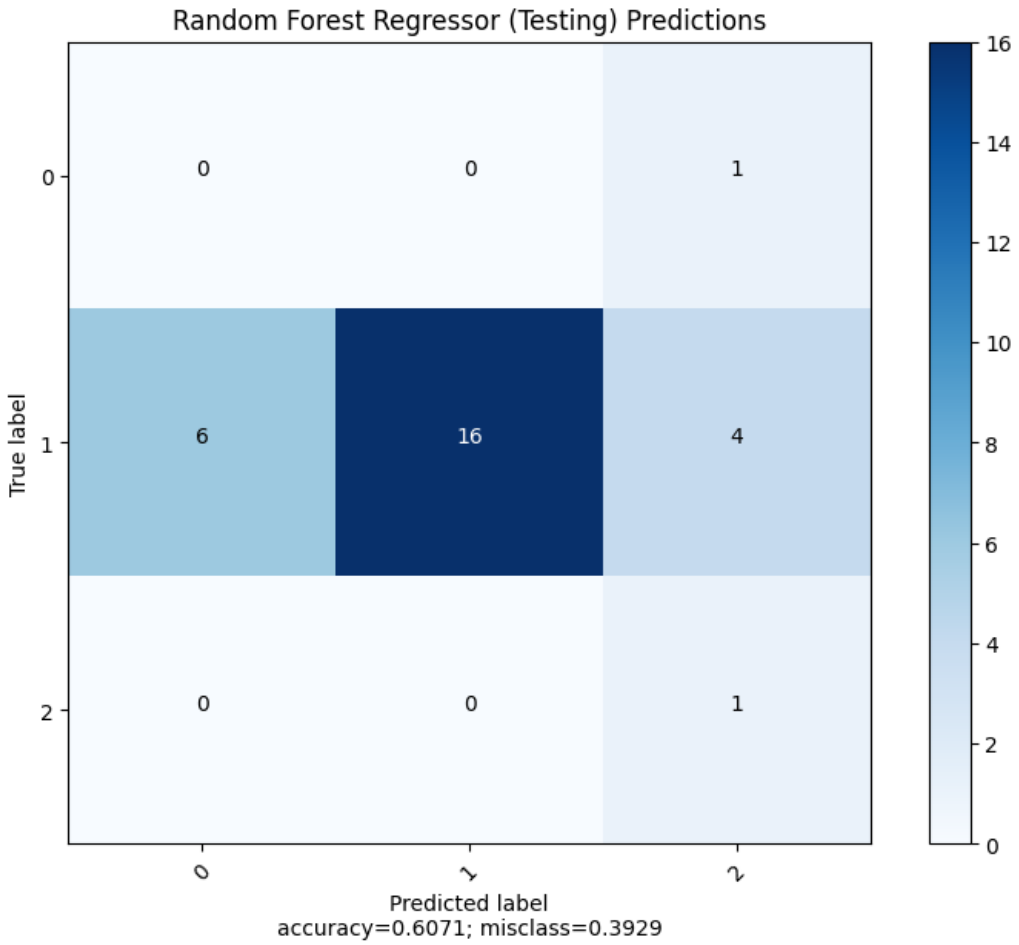

Figure 1: Testing Accuracy for Regression Forest

	Metrics for Ratings			**Overall Accuracy**
	Precision	Recall	F1 Score	
Linear Regression				0.227
`Rating=0`	0.167	0.250	0.200	
`Rating=1`	0.125	0.111	0.333	
`Rating=2`	0.200	0.118	0.353	
Linear Regression (L2)				0.454
`Rating=0`	0	0	0	
`Rating=1`	0.429	1	0.600	
`Rating=2`	1	0.111	0.200	
Regression Tree				0.409
`Rating=0`	0.167	0.250	0.200	
`Rating=1`	0.444	0.444	0.444	
`Rating=2`	0.571	0.444	0.500	
Regression Forest				0.500
`Rating=0`	0	0	0	
`Rating=1`	0.450	1.000	0.621	
`Rating=2`	1.000	0.222	0.363	
OvR				0.500
`Rating=0`	0	0	0	
`Rating=1`	0.435	0.777	0.560	
`Rating=2`	0.666	0.444	0.533	
Fixed Effects				0.366
`Rating=0`	0.333	0.372	0.408	
`Rating=1`	0.372	0.362	0.365	
`Rating=2`	0.352	0.352	0.385	

Table 4: Validation Metrics for Overall Rating