A blue and grey text on a black background

Description automatically generated

**COMP-7745-002 Machine Learning**

Optimizing Food Delivery Decisions: A comparative analysis of Random Forest and XGBoost for Food Ordering Prediction

**Submitted by**

|  |  |
| --- | --- |
| Neehanth Reddy Maramreddy | U00924209 |
| DuniSandeep Daddala | U00935835 |
| Rajesh Chinthalapudi | U00921971 |

**on**

**04/30/2024**

# Abstract

This study examines how well machine learning models analyze customer feedback and predict meal ordering behavior based on location. Using an online food order dataset, we apply feature engineering, data cleaning, and categorical variable encoding. We then split the data into training and testing sets, and built and trained three models: Random Forest, XGBoost, and Gradient Boosting Classifier. Performance of these models are evaluated by calculating the evaluation metrics such as Accuracy, Precision, F1-score, and Recall. To visualize performance, we employ ROC curves and confusion matrices. This study suggests that Random Forest, XGBoost, and Gradient Boosting Classifier can all be useful tools for optimizing food delivery decisions based on location and customer reviews.

# Introduction

The development of smartphones and the internet in the modern era has completely changed how people orders takeout. Launched in 1995, Worldwide Waiter (now Waiter.com) was the first online food ordering service, initially catering to a small region of northern California. A notable shift towards online ordering was seen by the establishment of mobile applications by major pizza chains by the late 2000s.

The restaurant is an essential part of modern-day life for several reasons. It is, first and foremost, an essential economic component. Restaurants provide a major contribution to the local, national, and international economies through creating jobs, generating large amounts of income, and encouraging economic progress. But in the early phases of the pandemic, lockdowns, and the need to keep a physical distance had a significant effect on the delivery industry, which led to the rise of the online meal delivery service trend. Although this service was offered in the past, it was not well-known or utilized. It turned out to be crucial for the struggling restaurant business [1].

These days, food delivery services use complex algorithms to anticipate user preferences and suggest restaurants. By investigating the possibilities of machine learning methods, particularly Random Forest and XGBoost, to further customize the user experience, this project expands on the current trend. Our goal is to create models that can precisely forecast decisions about ordering food by examining customer location and review opinions. For food delivery services, having this information can be revolutionary as it allows them to make more personalized recommendations and, in the end, give patrons a more enjoyable dining experience.

The ease of use and the ever-growing variety of restaurants and cuisines have led to the rapid expansion of the online meal ordering market in recent years. Food delivery platforms need to use advanced methods for predicting consumer tastes and customize their dining experience in order to stay competitive in this ever-changing market.

This project looks into how machine learning might be used to forecast meal choices based on review opinion and customer location. To construct predictive models that can take advantage of these factors and provide users with food recommendations, we concentrate on two well-liked machine learning algorithms: XGBoost and Random Forest.

We want to create models that can precisely predict a customer's likelihood of placing an order by examining customer data, such as location and review opinions. Food delivery services can utilize this data to customize their menus and advertising tactics, which will provide customers with a more customized and enjoyable dining experience.

The creation and assessment of the Random Forest and XGBoost food order prediction models are covered in detail in this report. We examine the phases involved in preprocessing the data, training the model, and contrasting the two algorithms' performances using a range of assessment metrics. The results will provide insight into how well these machine learning techniques work to optimize food delivery choices based on consumer feedback and geographic location.

# Methodology

**Data Acquisition and Preprocessing**

* **Data Source**: We have collected the data from Kaggle, "onlinefoods.csv", containing the data relevant to food orders, such as customer info details and their feedback.
* **Import required libraries:** In this step we import the required libraries like numpy, pandas, sea born, matplotlib time, label encoder and some other metrics libraries like accuracy score, precision score, f1 score etc .
* **Load the dataset:** In this step we load the dataset called onlinefood.csv file by using pandas.

* **Basic Checks:** Here we do basic checks like info of the data, if the data has any null values, describe the data to know mean, standard deviation to each column.

We dropped a column which is named 'Unnamed: 12’ in the data.

* **Exploratory Data Analysis:** In this stage we plot graphs by using seaborn and matplotlib libraries and we compared columns how they behave.
* **Label Encoder:** Label encoder is used to convert the objective type data to numerical data. In this stage we change the objective data to numerical data by using replace function or label encoder library.
* **Splitting the data for training and testing data:** In this step we split the data for training and testing purpose with test size 0.25.
* **Model implementation:** Here we write the code for Random Forest and XG Boost, SVM Classifiers, and Gradient Boosting Classifier from scratch for the training and testing of the data. Provide arguments like n estimators, max depth, min sample split, learning rate etc.
* **Training and testing the data:** In this step we train the data in both the models by using fit() and also we test the data in all models. We have also found the time taken for training the data in models by importing time library.
* **Evaluating the models:** In this step we found accuracy score, precision score, recall score, f1 score for the testing data in XG Boost and Random Forest, SVM Classifiers, Gradient Boosting Classifier, Polynomial kernel function SVM and Radial Basis kernel function SVM models and we have done confusion matrix visualization for all models.

We plotted ROC Curve and Precision Recall curves.

**Modeling and Training**

1. **Random Forest Classifer:**

* The main decision tree algorithm is implemented by a custom DecisionTree class defined. The predict() and fit() methods which are defined in this class are used to make predictions and train the tree, respectively.
* By generating an ensemble of decision trees (n\_estimators) with hyperparameters like max\_depth (maximum depth of each tree) and min\_samples\_split (minimum samples required to split a node), the RandomForestModel class extends the DecisionTree class.
* Fitting each tree on a random sample of data points from the training data (X\_train and y\_train) is how the model is trained.
* By summing up the predicted outcomes from each tree in the ensemble, predictions are generated.

1. **XGBoost Model:**

* Defined a custom XGBoostModel class that implements a gradient boosting framework based on decision trees, much like the Random Forest model.
* The model receives stage-by-stage training. A fresh decision tree that focuses on fixing the mistakes of the preceding group is added to the group at each step.

1. **Gradient Boosting Classifier:**

* Establish the maximum tree depth, learning rate, and number of trees in the model setup. Create a new, empty list to hold the trees.
* Training: Apply decision trees in a step-by-step manner:

Based on the residuals (errors) of earlier trees, fit each tree.

Predictions should be updated by multiplying the learning rate by the predictions of the new tree.

* Prediction: To get the probability output, add up all of the scaled predictions from each tree and use a sigmoid function.

**Model Evaluation**

* **Training and Testing Split**: Using train\_test\_split from scikit-learn, the data is divided into training and testing sets (X\_train, X\_test, y\_train, y\_test). The models are fitted using the training data, and their performance is assessed using the testing data.
* **Evaluation Metrics**: For both models, the code computes several evaluation metrics, such as:
  + - Accuracy: The percentage of right forecasts.
    - Precision: Ratio of actual to predicted positive results.
    - Recall: True positive to actual positive ratio.
    - F1-score: The precision and recall harmonic mean.
* **Confusion Matrix**: To explain the distribution of accurate and inaccurate predictions for each class (whether in order or not).
* **ROC Curve**: The Receiver Operating Characteristic (ROC) curve is a graphical representation that the code uses to assess how well the models classify cases into positive and negative categories at different classification thresholds. AUC, or the area under the ROC curve, is a summary of the model's classification capabilities.
* **Precision-recall curve**: Plotting a precision-recall curve allows to determine how recall and precision compromise off for various classification thresholds.

# Model Descriptions

**Random Forest**

Random Forest is a machine learning algorithm based on decision trees, rather than a mathematical function in the traditional sense. However, there are mathematical principles and computations involved in the construction and operation of decision trees, which are the building blocks of Random Forest.

Here are some key mathematical concepts and computations involved in Random Forest:

* **Decision Trees:** Decision trees are constructed using algorithms such as CART (Classification and Regression Trees) or ID3 (Iterative Dichotomiser 3). These algorithms involve mathematical calculations to determine the best feature and split point at each node of the tree based on criteria such as Gini impurity (for classification) or variance reduction (for regression).
* **Ensemble Learning:** Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive performance. The predictions of individual trees are aggregated using methods such as majority voting (for classification) or averaging (for regression).
* **Random Subsampling:** In Random Forest, each decision tree is trained on a random subset of the training data, selected with replacement (bootstrap sampling). This involves random selection of samples, which follows mathematical principles of probability distributions.
* **Random Feature Selection:** At each node of a decision tree, a random subset of features is considered for splitting. This random selection helps to de-correlate the trees and reduce overfitting. The selection process can involve mathematical computations to determine the best subset of features.
* **Hyperparameter Tuning:** Random Forest has hyperparameters such as the number of trees, maximum tree depth, minimum samples per leaf, etc. Tuning these hyperparameters often involves mathematical optimization techniques such as grid search or randomized search to find the optimal set of hyperparameters that minimize a chosen objective function (e.g., cross-validation error).

**XGBoost Classifier**

XGBoost is an ensemble learning method that builds an additive model by sequentially adding weak learners (decision trees) to minimize a predefined loss function. The model combines the predictions of multiple weak learners using a technique called gradient boosting, where each new tree is trained to correct the errors of the previous trees.

Key Components:

* **Weak Learners (Decision Trees):** The base learners in XGBoost are decision trees, which are shallow trees with a small number of nodes. Each tree captures a simple relationship between input features and the target variable.
* **Gradient Boosting:** XGBoost applies gradient boosting to iteratively improve the model's predictive performance. At each iteration, a new tree is added to the ensemble to correct the residual errors of the previous trees.
* **Regularization:** XGBoost incorporates regularization techniques to prevent overfitting and improve generalization performance. It includes L1 (Lasso) and L2 (Ridge) regularization terms in the objective function to penalize large model coefficients.
* **Loss Function:** The loss function measures the discrepancy between the predicted and actual values and serves as the optimization objective during model training. XGBoost supports various loss functions depending on the task, such as regression, classification, and ranking.

Mathematical Equations: The XGBoost model can be described mathematically using the following equations:

* Objective Function: XGBoost minimizes the following regularized objective function:

A black symbols with a plus and a positive symbol

Description automatically generated with medium confidence

Where,

* = loss function measuring the difference between the true target and the predicted target
* is the regularization term for the 𝑘th tree, and 𝐾 is the total number of trees.
* **Loss Function:** The choice of loss function depends on the task:
* For regression: Mean Squared Error (MSE)
* For binary classification: Log Loss (binary cross-entropy)
* For multiclass classification: Softmax Loss (multiclass cross-entropy)
* **Gradient and Hessian:** During each iteration, the gradient and Hessian of the loss function with respect to the predicted scores are computed to update the model parameters.
* **Tree Construction:** Each new tree is built to minimize the objective function by recursively partitioning the feature space. The splitting criterion is chosen to maximize the information gain, which is computed based on the gradient and Hessian.

**Gradient Boosting Classifier**

The Gradient Boosting Classifier model is based on the principle of gradient descent optimization. It minimizes a loss function by iteratively fitting new weak learners to the negative gradient of the loss function with respect to the current ensemble's predictions. The final prediction is obtained by aggregating the predictions of all weak learners.

Key Features:

* **Gradient Descent Optimization:** The algorithm minimizes a differentiable loss function by descending along the negative gradient.
* **Sequential Learning:** Weak learners are added to the ensemble sequentially, with each new learner focusing on correcting the errors of the existing ensemble.
* **Shrinkage (Learning Rate):** A shrinkage parameter controls the contribution of each weak learner to the ensemble, preventing overfitting and improving generalization.
* **Tree-Based Learners:** Decision trees are commonly used as weak learners, but other base learners can also be used.
* **Regularization:** Regularization techniques such as tree depth restriction and subsampling can be employed to prevent overfitting.
* **Early Stopping:** Training can be stopped early based on performance on a validation set to prevent overfitting.

# Experiment and Results

**Demographic Analysis:**

A graph of age distribution

Description automatically generated A graph with red squares

Description automatically generated

A graph of a number of people

Description automatically generated A graph of a number of green bars

Description automatically generated

A graph with green dots

Description automatically generated A chart with orange bars

Description automatically generated with medium confidence

A diagram and pie chart

Description automatically generated A graph with blue and orange bars

Description automatically generated

**Evaluation Metrics:**

A screenshot of a computer

Description automatically generated

Fig 1: Evaluation Metrics

**Confusion Matrix:**

A blue squares with white text

Description automatically generated

Fig 2: Confusion Matrix Visualization

**ROC Curve:**

A graph of a function

Description automatically generated with medium confidence

Fig 3: ROC Curve

**Precision-Recall Curve:**

A graph of a line graph

Description automatically generated with medium confidence

Fig 4: Precision-Recall Curve

# Database

A screenshot of a computer

Description automatically generated

The dataset 'onlinefoods.csv' we are using for this study is being fetched from Kaggle. This dataset contains 388 rows and 13 features including the unnecessary columns, which are of type float64 (2), int64 (3), and object (8).

# Training and testing logs

**Computational Times:**

A screen shot of a computer

Description automatically generated

Fig 5: Computational times for Random Forest Classifier

A screenshot of a computer

Description automatically generated

Fig 6: Computational times for XGBoost model

A screen shot of a computer

Description automatically generated

Fig 7: Computational times for Gradient Boosting Classifier

# Discussion and comparison

**Computational times:**

* Random Forest: Depending on the particular problem, a good balance between testing and training speed, but not the most accurate model.
* XGBoost: If most of the work involves developing predictions on fresh data, this is a good option. It is slower to train than Random Forest but much faster for testing.
* Among the three, the Gradient Boosting Classifier has the slowest testing and training times. Given that it is generally simpler to understand than XGBoost, it could be a wise option if interpretability is a top concern.

**Evaluation Metrics:**

* XGBoost: The evaluation metrics indicate that XGBoost is the top performer. As proven by the F1-score, it gains the maximum accuracy and strikes a good balance between recall and precision.
* Gradient Boosting Classifier: Although it has the highest precision, it may miss a larger percentage of real positive cases due to its lower recall and F1-score when compared to XGBoost.
* Random Forest: Out of the three, it has the lowest accuracy and F1-score but is the fastest to train.

**Confusion Matrix Visualization:**

Random Forest:

* Out of 152 positive cases, Random Forest correctly classified 113 of them (True Positives).
* 11 out of 152 positive cases were mistakenly categorized as negative (False Negatives).
* 16 out of 152 negative cases were mistakenly categorized as positive (False Positives).

XGBoost:

* 113 out of 152 positive cases (True Positives) were correctly classified.
* 11 out of 152 positive cases were mistakenly categorized as negative (False Negatives).
* 16 out of 152 negative cases were mistakenly categorized as positive (False Positives).

Gradient Boosting:

* 113 out of 152 positive cases (True Positives) were correctly classified.
* 11 out of 152 positive cases were mistakenly categorized as negative (False Negatives).
* 16 out of 152 negative cases were mistakenly categorized as positive (False Positives).

With the highest recall and F1-score - two important metrics for imbalanced datasets - Gradient Boosting seems to be the best model overall for this dataset.

# Conclusion

In this study, we investigated how well machine learning models could forecast restaurant ordering patterns based on review opinion and customer location. After performing feature engineering and data cleaning on a food order dataset, we used Random Forest, XGBoost, and Gradient Boosting Classifier. While all three models performed well, XGBoost showed a tiny advantage in terms of accuracy and F1-score. XGBoost performed exceptionally well in testing speed, while Random Forest provided the fastest training times. Even though it was slower, the gradient boosting classifier offered a model that might be easier to understand. All three of the models may be useful resources for meal delivery services, according to our evaluation metrics and confusion matrices. By reviewing customer feedback and personalizing recommendations based on their location, they can optimize delivery decisions and ultimately improve the dining experience for their customers.

# Limitations

* Predictions in real-world scenarios may be biased or inaccurate due to limitations or biases in the data used.
* The list excludes additional variables like user demographics, the time of day, promotions, and weather that may affect food ordering behavior. Increasing the number of features in the model may help it perform better.
* It is possible that the models require retraining or fine-tuning for varying customer demographics or geographic locations, and that the dataset utilized does not accurately represent the entire population.
* Gradient Boosting Classifier's performance metrics were slightly lower than XGBoost's, despite providing improved interpretability.

# References

[1] R. D. Abhang, B. D. Bailurkar, S. S. Save, P. D. Ingale, and M. Y. Patekar, “Zomato Review Analysis Using Machine Learning,” May 2023, doi: 10.1109/iconscept57958.2023.10170538.

[2] [https://www.kaggle.com/code/yinn94/food-visualization-classification-acc-0-91/](https://www.kaggle.com/code/yinn94/food-visualization-classification-acc-0-91/notebook)

[3] <https://www.kaggle.com/code/jelenadimitric/online-food-analysis-with-coordinates>

[4]<https://www.researchgate.net/publication/348631934_Application_of_Random_Forest_Algorithm_for_the_Prediction_of_Online_Food_Delivery_Service_Delay>

[5] <https://www.mdpi.com/2076-3417/13/4/2299>

[6] <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>

[7] <https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d>