Designing a Machine Learning Model to Predict Food Items

A Random Forest Model with 89.7% Testing Accuracy

Franco Miguel Valencia
Jeremy Xie
Mykola Zhuk

CSC311

Table of Contents

Introduction

In this report, we describe our approach to developing a reliable yet efficient predictive model using machine learning. Using a dataset containing eight features pertaining to three different food items, our goal was to analyze and preprocess the data to build a model capable of accurately classifying responses into three categories: Pizza, Shawarma, and Sushi.

To achieve this, we developed scripts to preprocess the data and make it interpretable. We then evaluated the efficacy of four families of models in learning our dataset. Ultimately, we trained a Random Forest classifier on our cleaned dataset, which achieved a **5-fold cross-validation accuracy of 90.3%** with optimized hyperparameters and a **final testing accuracy of 89.7%**. This report outlines the steps taken to clean and transform the data, train the model, and ultimately generate predictions using a custom-built script.

With only three members, we had to put in substantially more work to compensate for one of our team members withdrawing from the course. We would greatly appreciate it if this were taken into consideration. We hope you enjoy our work!

Section 1. Data Processing and Feature Engineering

The combined dataset consisted of several features, including numerical, multiple-choice, and free-text. To make the dataset interpretable by machine learning models, we began by discussing how to select and convert all features to numerical representations, which is described in Section 1.1. We then conducted a thorough exploration of the data, which is also described in Section 1.1. Lastly, in Section 1.3, we describe how we split the data into various sets.

Section 1.1. Feature Selection, Data Representation, and Data Exploration

We wanted to consider **all available input features** when designing our model. As shown in this section, we determined that all available features did show some degree of separability, making them useful for classification in our final model.

Below, we outline our process for transforming all categorical and textual features into numerical representations, as well as our interpretations of our exploration of the data.

Section 1.1.1. Numerical Features

Question 1: Food Complexity

Q1, which measures the food's complexity on a scale of 1 to 5, was already written in the form of an integer, requiring no additional modifications. As shown in Fig.1, we visualized a boxplot in order to understand the range and distribution of each label.

Fig. 1. Boxplot of Q1 (Complexity) by Label. The median is shown in green.

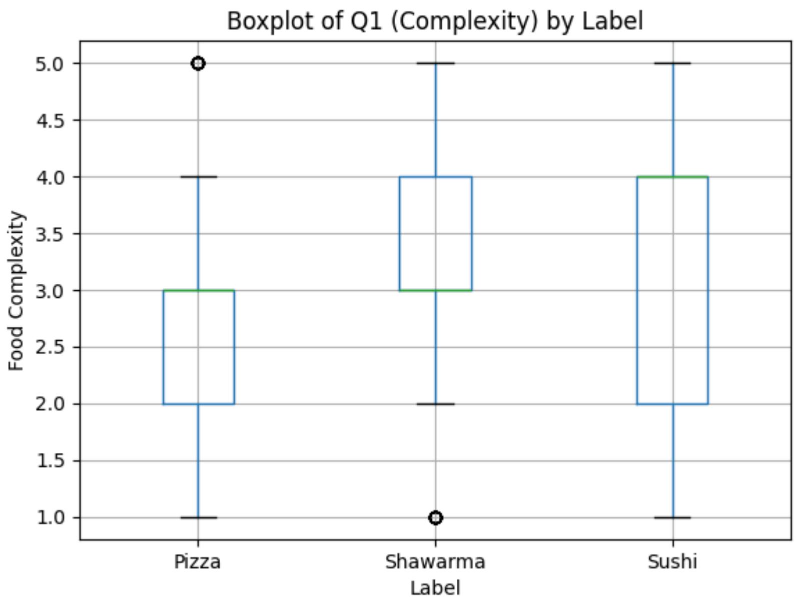
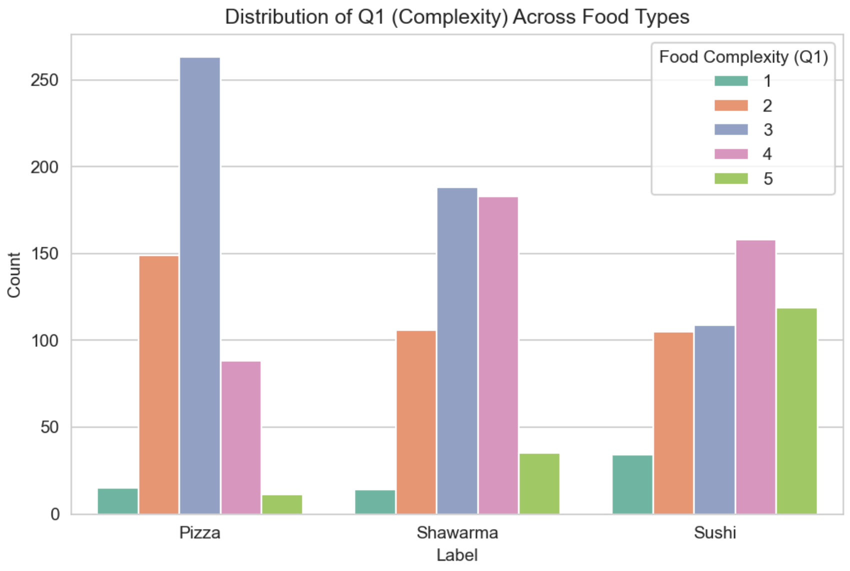


Fig. 2. Distribution of Q1 (Complexity) Across Food Types



According to Fig. 1, most students rated Pizza as moderately complex to make, with a median of 2 and a boxplot range of 2 to 3 (centred around 2.5), indicating a slight tendency toward lower complexity. Shawarma also had a median of 3, but with a range of 3 to 4 (centred around 3.5), suggesting it was perceived as somewhat more complex than Pizza. Conversely, sushi had a broader range (2 to 4) with its median at the upper end, indicating that students generally considered it the most difficult to prepare.

We confirmed these results using a bar plot, as shown in Fig. 2. Students predominantly considered Pizza as having a difficulty level of around 3, while they perceived Shawarma as slightly more complex (around 3 to 4). The complexity level for Sushi exhibited a left-skewed distribution, indicating the highest complexity out of all three items.

Collectively, these trends suggest that Q1 has moderate predictive strength in determining the label. Pizza is generally perceived as the least complex, Shawarma as moderately complex, and Sushi as the most complex. Notably, Sushi had the most variability in responses, whereas Pizza and Shawarma exhibited comparable variation.

Question 8: Hot Sauce Level

Q8 measured hot sauce from “None” to “A lot.” Since this question captures hierarchical information on an ordinal scale, we converted string answers for this feature to an integer scale from 0 to 4:

- “None” → 0
- “I will have some of this food item with my hot sauce” → 1
- “A little (mild)” → 2
- “A moderate amount (medium)” → 3
- “A lot (hot)” → 4

Fig. 3. Boxplot of Q8 (Sauce Level) by Label. The median is shown in green.

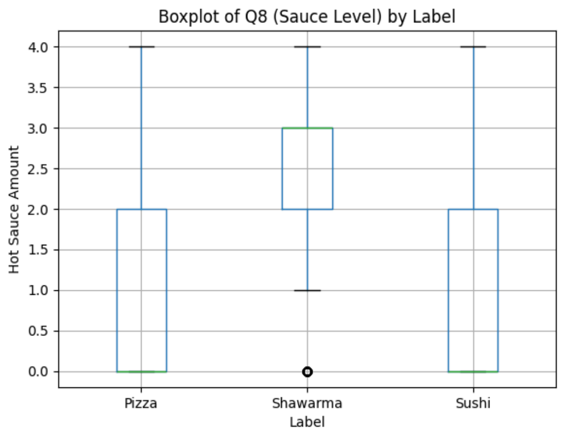
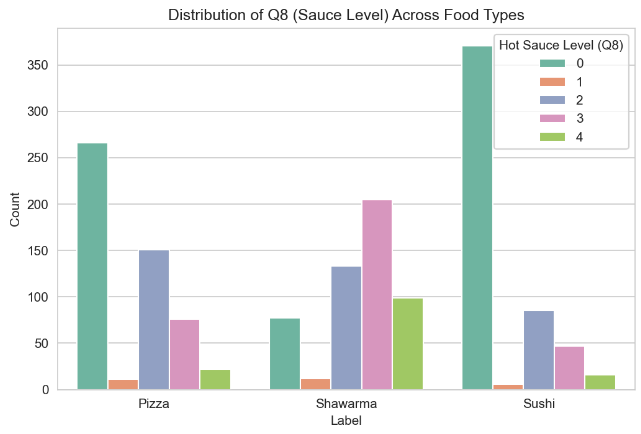


Fig. 4. Distribution of Q8 (Hot Sauce Level) Across Food Types



As shown in Fig. 3, most students indicated a preference for little-to-no hot sauce on Pizza (ranging from 0 to 2, centred at 1) and Sushi (also ranging from 0 to 2, centred at 1). However, many suggested using a higher amount of hot sauce with Shawarma (ranging from 3 to 4, centred at 3.5). This is supported by Fig. 4, as the vast majority of students chose not to put any hot sauce on either Sushi or Pizza while favouring a moderate to the maximum amount of hot sauce on Shawarma. This suggests that a high hot sauce level may have strong predictive power for identifying Shawarma, but not necessarily for Pizza or Sushi.

Section 1.1.2. String-Based Features

For open-text responses (Q2, Q4, Q6), we developed scripts to extract numerical values wherever possible. However, given that these responses also introduced the possibility of missing values, we applied specific strategies to handle them, as described in this section.

Question 2: Ingredient Count and Question 4: Expected Price

Q2 asked for the “number of ingredients” in the food item, while Q4 asked about the “price” of the food item. We took a step-by-step approach to identify any representations of quantity embedded in the responses:

1. Firstly, if the input was solely an integer, it was used directly.

2. Otherwise, if the input contained a float or integer surrounded by other characters, we used regular expressions to extract and round the first number in the string to the nearest integer.

3. If no numerical values were found, we searched for lexical representations of numbers (e.g., “one,” “three”) and converted them to integers in the same manner.

4. If no numerical values could be identified in the response, the input was treated as missing, and we assigned a synthetic value of -1. We experimented with other imputation methods, such as median imputation, but this reduced classification accuracy, likely because it introduced artificial similarities between missing and non-missing values. Assigning a value of -1 allows the model to treat missing data as an explicit placeholder without assuming a misleading statistical relationship.

In Fig. 5 and Fig. 6, we visualized the distributions of each question’s responses.

Fig. 5. Boxplot of Q2 (Number of Ingredients) by Label

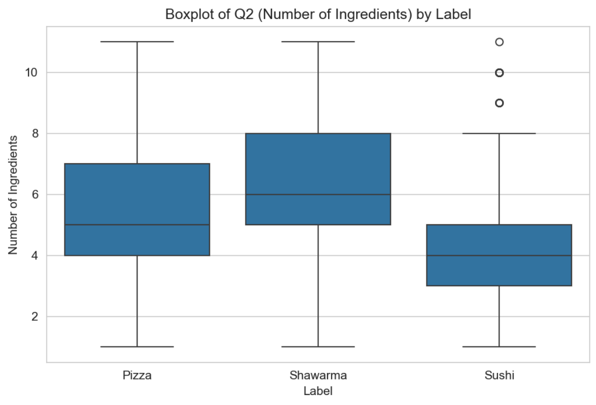
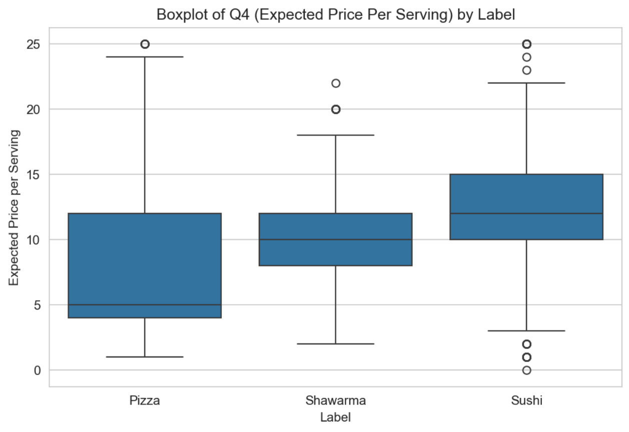


Fig. 6 Boxplot of Q4 (Expected Price Per Serving) by Label



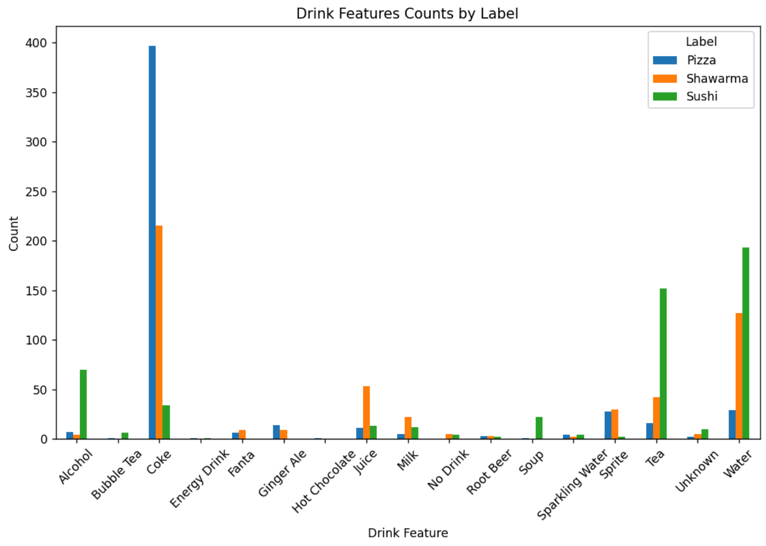
As shown in both figures, there appear to be clear differences in the expected number of ingredients and price between all three food types. For Q2, students expected a moderate number of ingredients for Pizza (median = 5), slightly more for Shawarma (median = 6), and fewer for Sushi (median = 4). For Q4, students expected to pay relatively little for Pizza (median = \$5), somewhat higher for Shawarma (median = \$10), and the highest for sushi (median = \$12). Collectively, these results suggest that Q2 and Q4 are both useful for characterizing food type.

Question 6: Paired Drink

Q6 contained free-text drink names, which could have drastically increased the number of features in the dataset if we employed a traditional Bag of Words approach. To limit the number of possible categories while maintaining interpretability, we applied a text-mapping approach to group similar terms under a single category. For example, “Diet Coke,” “Coca-Cola,” and “Pepsi” were all mapped to the category “Coke,” reducing redundancy and preventing excessive categorical information.

Importantly, some students typed the word “None,” which was automatically recognized as a missing value (NaN) rather than a string literal. To address this, we mapped such cases—as well as inputs that did not map to a common drink category—to a one-hot vector corresponding to “Unknown” drinks. Fortunately, such cases were rare. As shown in Fig. 7, we generated a grouped bar chart to illustrate the frequency of drink pairings (e.g., Coke, Water, Tea) with each food label in the cleaned dataset.

Fig. 7. Grouped Bar Chart of Q6 (Associated Drinks) by Label



Pizza is predominantly linked with Coke and other soft drinks (e.g., Fanta, Ginger Ale), aligning with typical fast-food combinations. On the other hand, Shawarma was almost exclusively associated with juice and milk, while Sushi was primarily associated with alcohol, tea, bubble tea, and soup. These differences reflect cultural dining norms, not only revealing how participants conceptualize food-and-drink pairings but also suggesting that certain drinks serve as strong predictors for specific food categories.

Section 1.1.3. Multiple-Choice Features

Question 3: Meal Settings and Question 7: Related People

For features where respondents could select multiple answers, we converted responses to one-hot vectors. This approach was ideal because:

a) It preserved all relevant information from multiple-choice selections; and
b) It ensured that our trained models could examine each choice without assuming any hierarchical (ordinal) structure.

The results of our one-hot encoding are shown in Fig. 8 and Fig. 9.

Fig. 8. Grouped Bar Chart of Q3 (Meal Settings) by Label

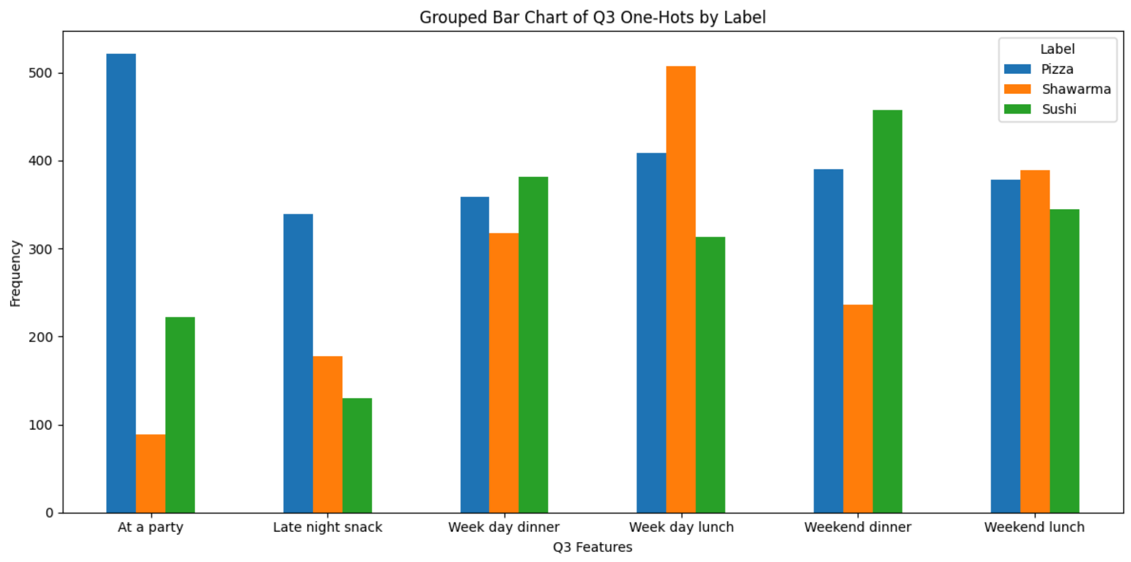
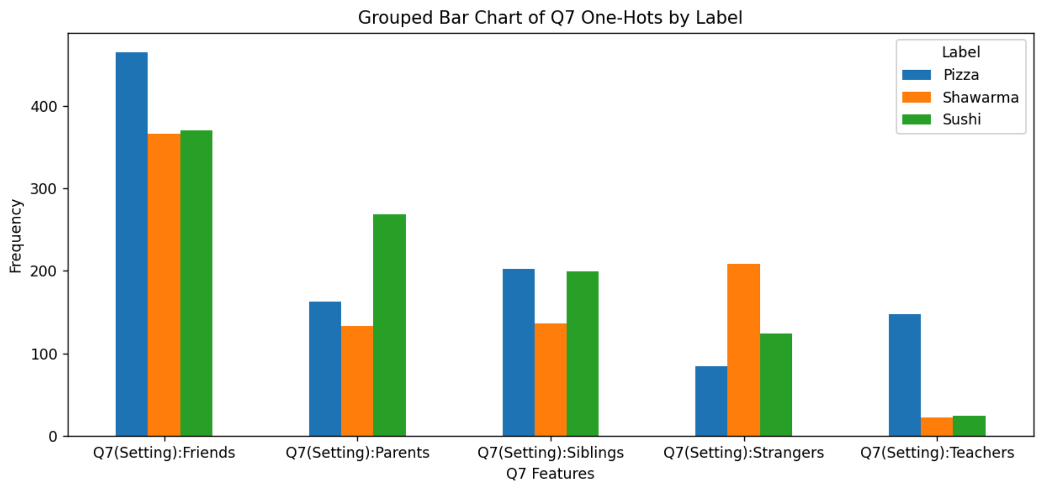


Fig. 9. Grouped Bar Chart of Q7 (Social Groups)



The frequency of responses as shown in the one-hot encoded vectors offers insights into the contexts in which students associate with each food item. As shown in Fig. 8, Pizza is most likely to be served at a party, Shawarma is most likely to be served at a weekend lunch, and Sushi is most likely to be served at a weekend dinner. Contrarily, Shawarma is very unlikely to be served at a party, while Sushi is unlikely to be served at a late-night snack or party.

Fig. 9 further highlights these differences, illustrating how participants associate each food label with social groups. For example, Pizza is strongly linked to friends and teachers, suggesting communal or school-related settings. Sushi is more often associated with parents, hinting at family dining traditions, while Shawarma is linked to strangers, implying it is more often eaten in casual or public settings.

These results suggest that Q3 and Q7 both serve as moderate predictors of the food label, capturing the situational and social contexts in which each food item is typically consumed.

Section 1.1.4. Handling Q5 (Movie Names)

Question 5 presented the most significant preprocessing challenge, as it allowed for the most open-ended responses. Unlike Q6, where drink names were relatively structured, many Q5 responses contained spelling mistakes, inconsistent punctuation, and full sentences rather than just movie titles. If taken directly, this would have resulted in nearly 1000 unique values, drastically increasing our dataset size and feature count.

Thus, we began by considering several different approaches to handling Q5, as outlined in Table 1:

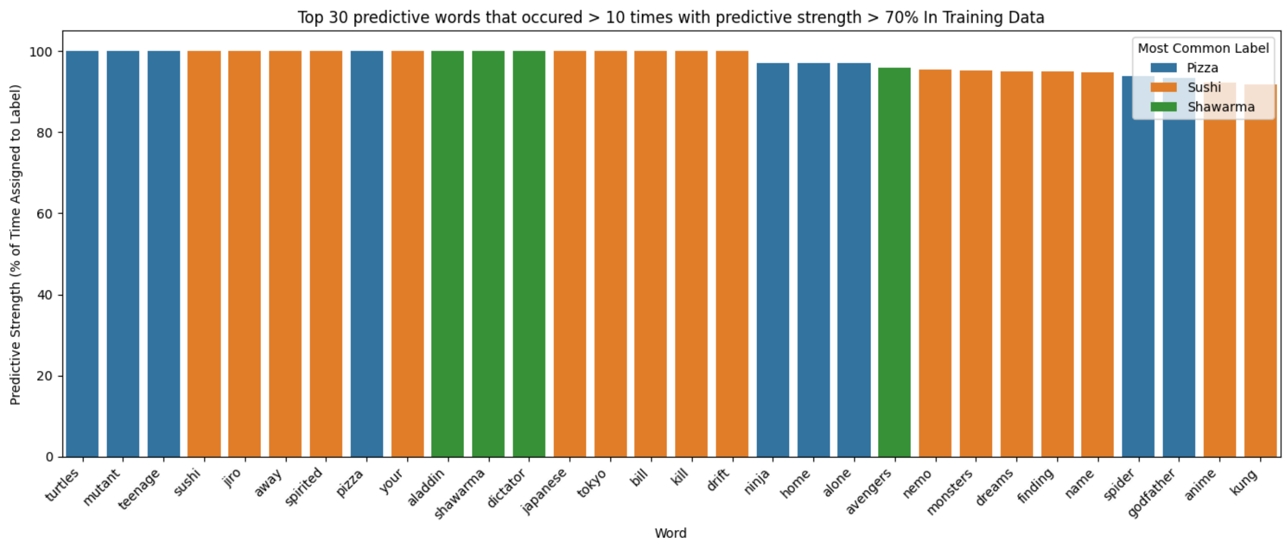
Table 1. Possible Approaches to Handling Q5

Approach	Pros	Cons
Bag of words (No Preprocessing)	Captures word relationships to food labels in title names	High-dimensional data (time and space-intensive)
One-hot Lookup table (Curating a vocabulary of titles either by GPT or manual, which	Can be paired with autocorrect to reduce noise Results highly interpretable	The curating step is highly error-prone and time-intensive High-dimensional data (time and space-intensive)
Bag of words (Most predictive, occurring more than 10 times in training, correlated with its most common label more than 70% of the time)	Words with low predictive power are dropped Reduced Dimensionality (~1000 to ~50) Results highly interpretable	Niche movie titles that may be strong indicators are lost, albeit these are rarer. Words with low frequency (e.g. spelling mistakes) are dropped Some words are redundant (e.g. same movie title)

Ultimately, we adopted the filtered Bag of Words approach, as it provided the best balance between interpretability, predictive power, and computational efficiency. Unlike the raw Bag of Words approach, which preserved all words but resulted in excessively high dimensionality, the filtered version allowed us to focus on words that were frequent enough to be meaningful while predictive enough to contribute to classification.

Thus, we proceeded by modelling the most "predictive" words; that is, words that most often correctly predicted a label in the training data.

Fig. 10. Top 30 Most Predictive Words From Q5 Answers



We ended up with around 50 words with a predictive strength of over 70%, but only the top 30 words shown in Fig. 10 are displayed due to size constraints. These results were highly promising and intuitive, aligning with common cultural associations. To give some examples:

- Titles correlated with Pizza (Often Pizza/Italy related)
  - Teenage Mutant Ninja Turtles: The turtles are known for loving Pizza.
  - Home alone: MC loves cheese Pizza, memorable Pizza scene.
  - Godfather: Italian gangster setting.
  - Finding Nemo/Monsters Inc.: Pizza and kids' movies are often played at birthday parties.

- Titles correlated with Sushi (Often Anime/Japan-related)
  - Jiro Dreams of Sushi: Explicitly about sushi.
  - Spirited Away: A well-known Japanese anime

    - Tokyo Drift: Set in Japan.
    - Kill Bill: MC travels to Japan during the major arc of the film.

- Titles correlated with Shawarma (Often Shawarma/Middle East-related)
    - Aladdin: Set in the Middle East.
    - The Dictator: References a Middle Eastern setting.
    - The Avengers (2012): A memorable post-credit scene involves the team eating shawarma.

These results show that movie titles serve as surprisingly strong predictors for food preferences, as students seem to associate foods with the setting of the movie, sometimes in stereotypical ways. By filtering the most predictive words, we effectively reduced dimensionality from 1000 to 50, eliminating noise and weak predictors while maintaining interpretability.

Similar to Q6, it is important to note that many students typed the word “None,” which was recognized as a missing value (NaN) rather than a string literal. To address this, we mapped these inputs to a one-hot vector corresponding to unknown movies or words. Although niche titles were lost in this filtered Bag of Words approach, they were unlikely to generalize well due to their low frequency, making this tradeoff acceptable. Given these findings, we finalized Bag of Words with filtered predictive words as our preprocessing approach for Q5.

Section 1.1.5. Summary

The results of our data preprocessing and feature transformation are shown in Table 2. Overall, only string-based features (Q2, Q4, Q5, Q6) required direct handling of missing features, given that one-hot vectors inherently handle missing values and Q1 and Q8 require responses.

Table 2. Summary of Feature Transformations

Feature	Original Format	Processing Method	Rationale
Q1	Integer	As-is	Already numerical
Q2, Q4	String containing possible numbers	Extract the first integer, convert floats to integers, and identify number words; else discard. Missing values were assigned -1	Ensures numerical representation. Synthetic value maintains accuracy, ensures no rows were dropped, and does not introduce an artificial statistical relationship
Q5	String containing movie names	Determined the most predictive features and left out the rest to reduce dimensionality. Missing values were mapped to “Unknown” one-hot	Otherwise, we risk: High feature space, overfitting, possibility of reducing generalization, project restrictions, etc.
Q6	String containing drink names	Mapped to grouped categories. Missing values were mapped to “Unknown” one-hot	Reduces redundancy and prevents excessive categories
Q8	Ordinal relationship of hot sauce amount	Converted to integer scale 0-3	The ordinal relationship is preserved upon conversion to numerical
Q3, Q7	Multiple choice	One-hot encoding	Retains categorical information without ordering

Section 1.2. Data Splitting and K-Fold Cross-Validation

For our final model, we split the dataset into 80% training and 20% testing data, stratifying by classes in order to ensure the distribution of food types remained consistent across both sets.

However, to develop, validate, and optimize our model effectively before final training, we used "**k-fold cross-validation**," a technique that provides a more reliable estimate of model performance than a single train-test split. K-fold cross-validation splits the dataset into k equally-sized subsets (or "folds"). The model is trained on k-1 folds and tested on the remaining fold, repeating this process k times. By averaging the final performance metric across all folds, we obtain a more reliable generalization performance estimate compared to the variance introduced by a single random split.

We used k-fold cross-validation in two ways when developing our model:

1. **To tune our hyperparameters.** As further described in Section 3, we used an iterative grid search approach to optimize the hyperparameters used in our Random Forest classifier, including the number of estimators, maximum depth, and minimum samples to split at a node. Testing different combinations across multiple folds helps ensure our final hyperparameters are more generalized and not overfitted to a specific train-specific split.

2. **To estimate how well the model will perform on unseen data.** Using multiple folds gives us a more robust estimate of our model's accuracy compared to a single train-test split. We used cross-validation results to compare the Random Forest classifier against three other families of models: k-Nearest Neighbours (kNN), Logistic Regression, and Multilayer Perceptron (MLP). The findings from this comparative analysis are presented in Section 3.

In our case, we used k = 5 folds for hyperparameter tuning and evaluation. This means that for each iteration, the model was trained on 80% (4 folds) and tested on 20% (1 fold) of the data.

Section 2. Comparing and Experimenting with Models

When considering how to most accurately predict each food category, we explored various types of models. In this section, we present our rationale for experimenting with **k-Nearest Neighbours (kNN), Logistic Regression, Multilayer Perceptrons (MLPs)** and **Decision Trees / Random Forests,** highlighting each model's strengths and limitations. Our selection process considered several factors, including the structure of our data, feature distributions, and the interpretability of predictions.

For each model, we performed a 5-fold cross-validation grid search to optimize key hyperparameters within a limited range. This section details the hyperparameter choices and optimization strategies we employed.

The performance and comparative results of these models are presented and analyzed in Section 3.1.

Section 2.1 k-Nearest Neighbours (kNN)

kNN is a simple and intuitive algorithm that requires no explicit training phase and is very easy to implement. The algorithm works by calculating the Euclidean distance between points and assigning a class based on the majority vote of the k closest neighbours. Notably, kNN is a non-parametric model, meaning it makes minimal assumptions about the distributions of the data. This non-parametric nature makes kNN potentially adaptable to a wide range of data distributions and possibly effective in handling any complex relationships underlying our large dataset.

However, given that our dataset consists of several features, including multiple one-hot encoded categorical variables, we have a high-dimensional feature space. Due to the curse of dimensionality, kNN loses its effectiveness as data points tend to become equidistant. Furthermore, the computational cost of kNN increases significantly with the size of the dataset. Given that our dataset consists of hundreds of entries and several dimensions, the time required to compute distances, and therefore make predictions, can be substantial depending on the size of the input.

While it may not be the best choice given the scale of our data, kNN is a good starting point for understanding the basic concepts of classification and serves as a highly intuitive model. We experimented with different values for *k* to optimize kNN before comparing performance with the other listed models.

Section 2.2 Logistic Regression

Logistic regression is a widely used classification model that predicts the probability of an outcome based on input features. While primarily designed for binary classification, it can be extended to multi-class problems using softmax regression. In our case, given that we are dealing with a multi-class categorical outcome (i.e., only one of either Pizza, Shawarma, or Sushi), logistic regression is another reasonable choice for a straightforward baseline model.

While this model serves as a good point of comparison for assessing the impact of individual features on the prediction, it is important to acknowledge that logistic regression ultimately assumes linear separability, even among multiple features and classes. If the data is not linearly separable, which may be the case with a complex dataset such as ours, the model may struggle to accurately classify outcomes. We experimented with various values for C and regularization types to optimize our logistic regression model.

Section 2.3. Multilayer Perceptrons (MLPs)

Multilayer Perceptrons (MLPs) are a type of neural network capable of learning complex patterns and relationships within data. Unlike logistic regression, which assumes linear decision boundaries, MLPs can capture nonlinear relationships through activation functions. This makes them possibly well-suited for a large dataset such as ours, where relationships are likely more complex than linear relationships.

While MLPs generally offer greater representational power than logistic regression, they are also more computationally intensive and sensitive to hyperparameters, leading to a greater risk of overfitting. We experimented with different combinations of hidden layer sizes, activation functions, alpha values, and learning rates to optimize our MLP model before comparison.

Section 2.4. Decision Trees and Random Forests

Lastly, we considered Decision Trees and Random Forests, which we deemed most appropriate for the size and nature of our dataset. Decision Trees can handle both numerical and categorical data effectively, making them particularly relevant given the diversity of features and feature types present in our dataset. They are also more robust against the curse of dimensionality compared to distance-based methods like kNN because they make splits based on feature thresholds rather than distances.

Furthermore, given that Decision Trees are non-parametric universal estimators, capable of capturing intricate patterns without being overly sensitive to noise, they may be useful given that our dataset consists of many features and potentially intricate relationships.

By aggregating predictions across multiple Decision Trees, we can improve generalizability by reducing variance between predictions. This is known as an “ensemble” learning method, whereby each tree is trained on a bootstrapped subset of the data, and at each split, a random subset of features is considered. By averaging predictions across multiple such trees, overfitting is reduced while also decreasing variance, potentially improving generalizability. To optimize our Random Forest for comparison, we experimented with several hyperparameters, including the maximum depth of each tree and the minimum samples required to split.

Section 2.5. Summary of Models

Table 3. Pros, Cons, and Hyperparameters Tuned for Each Model

Model	Pros	Cons	Hyperparameters
k-Nearest Neighbours (kNN)	Non-parametric; very simple to implement; good baseline	Computationally intensive to make predictions, especially with many features; sensitive to irrelevant features	Number of neighbours
Logistic Regression	Simple and interpretable; works well for linearly separable data	Inappropriate for non-linear relationships, relatively sensitive to outliers	Regularization strength and type
Multilayer Perceptron (MLP)	Captures complex patterns; works well for larger datasets	Prone to overfitting and requires significant tuning	Number of layers and neurons, activation function, learning rate, solver,
Decision Tree / Random Forest	No linear assumptions; robust to noise; handles high-dimensional data well	Computationally expensive to train	Number of estimators, max depth, min samples per split, min samples per leaf, max features

Section 3. Model Choice and Hyperparameters

With four different families of models to choose from, we needed a **systematic approach** to compare their performance and determine the most suitable model for our task. This section outlines our process for defining evaluation metrics, comparing model performance, selecting the final model, and fine-tuning its hyperparameters to maximize predictive accuracy.

Section 3.1. Evaluation Metrics and Selecting Our Model

As described in Section 2, we experimented with multiple models, including Logistic Regression, k-Nearest Neighbours (kNN), Multilayer Perceptrons, and Random Forests.

To ensure a fair and systematic comparison among these models, we performed a 5-fold cross-validation grid search to optimize a select range of each model's hyperparameters After selecting the best hyperparameters, we then used 5-fold cross-validation once more to evaluate each model type's generalization performance using four evaluation metrics, calculated and interpreted as follows:

- **Accuracy**: Measures the overall percentage of correct predictions out of all predictions. This provides a general assessment of model performance.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Samples}$$

- **Precision**: Measures the proportion of correctly identified true positives out of all predicted positives (e.g., out of all "Pizza"s predicted, how many are actually "Pizza"s?). This is useful for minimizing false positives, as a high precision suggests that when a model predicts a certain food category, it is more likely to be correct. In the context of our project, this is a crucial metric if misclassification may have consequences; for example, if misidentifying a food could impact food recommendations.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

- **Recall**: Measures the proportion of true positives correctly identified (e.g., out of all "Pizza"s in the dataset, how many were identified as such?). This is useful for minimizing false negatives, as a high recall suggests that a model is capable of correctly identifying as many true instances as possible. In the context of our project, this may be important if failing to recognize a food type could lead to consequences such as negatively impacting user experience.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- **F1 Score**: The harmonic mean of recall and precision. This is particularly important if our dataset happens to be imbalanced (e.g., if “Pizza” appears more frequently than “Shawarma”). The F1 score helps mitigate this by combining precision and recall into a single metric, ensuring that a model is not overly biased toward a majority class.

$$F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

The optimal hyperparameters for each model as determined using 5-fold cross-validation are shown in Table 4. To further validate our model comparisons, we visualized the training curves of each model, as shown in Fig. 11, to ensure that none were overfitted to the training data. The training and validation curves for each model were relatively similar, indicating that overfitting was not a significant concern.

Lastly, the results of our comparative analysis using 5-fold cross-validation are shown in Table 5.

Table 4. Optimal Hyperparameters For Each Model Using 5-Fold Cross-Validation

Model	Optimal Hyperparameters
Logistic Regression	'logreg__C': 0.1, 'logreg__penalty': 'l1'
kNN	'knn__metric': 'manhattan', 'knn__n_neighbors': 7, 'knn__weights': 'distance'
Decision Tree	'max_depth': 5, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2
Random Forest	'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 15, 'n_estimators': 300
MLP	'mlp__activation': 'tanh', 'mlp__alpha': 1e-05, 'mlp__early_stopping': True, 'mlp__hidden_layer_sizes': (64,), 'mlp__learning_rate_init': 0.1, 'mlp__max_iter': 200, 'mlp__solver': 'sgd'

Fig. 11. Training Curves of Each Model Family During Optimization. Validation accuracy improved steadily for each model while training accuracy remained high, suggesting each model was likely neither overfitting nor underfitting.

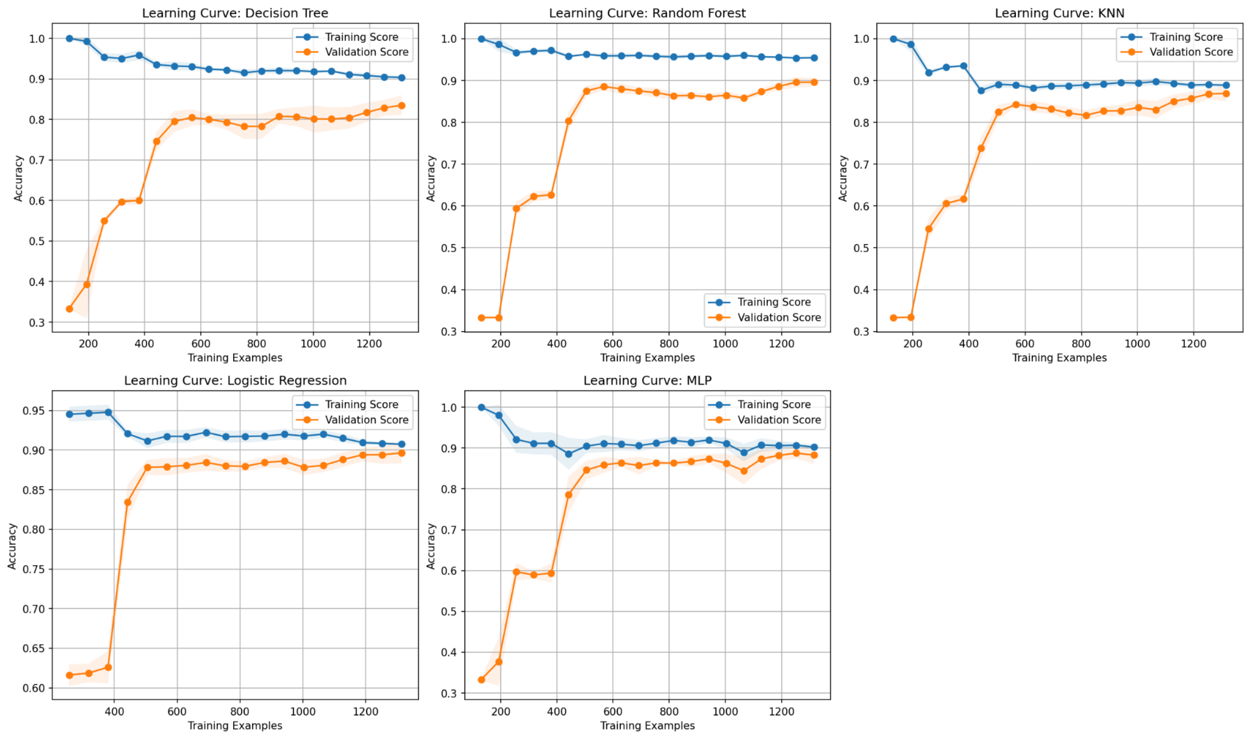


Table 5. Results of Comparative Analysis Using 5-Fold Cross-Validation

Metric	Logistic Regression	k-Nearest Neighbours (kNN)	Decision Tree	Random Forest	MLP
Accuracy	89.40%	85.02%	82.59%	90.28%	88.45%
Precision	89.47%	85.81%	82.73%	90.43%	88.80%
Recall	89.40%	85.02%	82.59%	90.28%	88.45%
F1 Score	89.43%	84.94%	82.56%	90.24%	88.94%

As shown in Table 4, Random Forest consistently achieved the best performance across all evaluation metrics. While a single Decision Tree performed relatively poorly on its own, a Random Forest showed the best generalizability, ability to avoid false positives, and ability to avoid false negatives, even accounting for potential class imbalances using the F1 score. This suggests that a Random Forest has the most potential for performing well on an unseen dataset, and the ability to capture complex relationships with respect to how we chose to represent the data.

Interestingly, however, Random Forest did show almost comparable performance with a Logistic Regression model. While the Random Forest did perform noticeably better, we nonetheless considered selecting a Logistic Regression model due to its speed of computation, ease of interpretability, and implementation simplicity. However, we ultimately proceeded with a Random Forest for several reasons:

- **Handling Non-Linearity.** Logistic Regression assumes linearity between features and the log-odds of a target. Although its high evaluation metrics may suggest a reasonable fit, Random Forests are non-parametric and are the safer choice for unseen data if the true relationships between features and labels are non-linear.
- **Robustness to Outliers & Variability.** Logistic Regression is more sensitive to outliers, requiring careful preprocessing as a result. In contrast, by aggregating outputs from multiple Decision Trees when making its prediction, Random Forests are less affected by outliers and more robust when generalizing to new data.
- **Generalization Performance.** Random Forest not only demonstrated the highest accuracy across all models but also the highest F1 score, suggesting strong performance even when accounting for potential class imbalances.

While Logistic Regression remains a good choice given its speed, simplicity, and interpretability, we ultimately prioritized robustness and flexibility in selecting the final model. Given its superior performance, non-parametric nature, and resilience to variations in unseen data, we ultimately selected a Random Forest as our final model.

Section 3.2. Hyperparameter Tuning

With our model type finalized, we employed a highly thorough iterative grid search approach to determine which hyperparameters would best support the predictive accuracy of the model.

Each hyperparameter we considered—n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features—plays a substantial role in the complexity, randomness, and consequently, generalizability of the Random Forest model.

- **n_estimators:** The number of trees in the random forest. As the number of trees increases, the prediction is averaged across a greater number of models, decreasing variability.

- **max_depth:** The maximum depth of each tree. A shallow tree may underfit, but limiting the depth can help prevent overfitting by restricting how complex the model can become. Finding an optimal depth is crucial for balancing bias (underfitting) and variance (overfitting).
- **min_samples_split:** The minimum number of samples required to split an internal node. Increasing this value can help reduce overfitting by preventing the creation of nodes based on very few samples.
- **min_samples_leaf:** The minimum number of samples that must be present in a leaf node. Setting this value helps ensure that a model does not create leaf nodes with very few samples, which can lead to overfitting.
- **max_features:** The number of features to consider when looking for the best split at each node. This helps introduce randomness into each model and reduce correlations between them, improving the effect of the bagging.

To systematically optimize these hyperparameters, we performed a grid search using 5-fold cross-validation. As previously described in Section 1.2, using k-fold cross-validation provides a more reliable estimate of model performance by ensuring that every data point in the dataset is used for both training and validation.

Our approach followed these steps:

1. First, we defined a range of values for each hyperparameter and iteratively tested different combinations.

2. The dataset was divided into 5 folds, where the model was trained on 4 folds and validated on the remaining fold, repeating the process across all combinations.

3. We averaged the accuracy across all folds, which ensures that our reported accuracy is not overfitted to a single training/validation split.

4. The hyperparameters with the highest cross-validation accuracy were selected, as they likely provide the strongest generalization performance.

The results of the 5-fold cross-validation grid search are summarized in Table 6.

Table 6. Hyperparameters Tested and Used When Optimizing the Final Random Forest Model. The final hyperparameters were determined based on the highest validation accuracy achieved during grid search, which was 89.7%. In total, 1800 different combinations of parameters were tested across the 5 folds, resulting in 9000 models trained.

Hyperparameter	Explanation	Values Tried	Best Choice
n_estimators	Number of trees in the random forest	50, 100, 200,300, 500, 1000	300
max_depth	Maximum depth of each tree	None, 10, 20, 30	10
min_samples_split	Minimum required samples in a node before splitting	2, 5, 10, 15, 20	15
min_samples_leaf	Minimum required of samples in a leaf in order to split	1, 2, 5, 10, 15, 20	1
max_features	Number of features to consider when looking for the best split	sqrt, log2, None	log2

Section 3.3. Final Model

Based on the results of the hyperparameter grid search, our final model is described as follows:

Algorithm: Random Forest Classifier

Final Hyperparameters

- n_estimators: 300
- max_depth: 10
- Min_samples_split: 15
- min_samples_leaf: 1
- max_features: log2

Implementation: The model was trained on 80% of the data, stratified by each class of food item. The remaining 20% of the data was used to evaluate the final model performance, which is described in Section 4. The file pred.py contains the classifier used to make predictions.

Prediction Implementation: The pred.py script is designed to load the trained model (model.py), which we trained using sklearn and serialized using the m2c library. Serialization and the training script are not included in the prediction package. Once loaded, the script uses the loaded model to generate predictions for new input data.

Given a dataset in CSV format, the script:

1. **Parses the data:** Converts each row to the appropriate data types (e.g., Q6 to categories, Q3 and Q7 to one-hot vectors, etc.) as described in Section 1.

2. **Reindexes the columns:** Ensures that all features align correctly with the trained model's expected input structure.

3. **Generates predictions:** The pred.py script iterates through each processed row and makes a prediction using model.py, which checks feature values using a series of nested if-statements. The script assigns a probability distribution over the three food categories depending on the path the tree takes, returning a prediction output (a numerical value) that is mapped to its corresponding food category:
    - 0 → Pizza
    - 1 → Shawarma
    - 2 → Sushi

By following this structured pipeline, pred.py ensures compatibility with the trained Random Forest classifier, allowing for accurate predictions on new data.

Section 4. Predictions

For our final model, we trained our Random Forest Classifier on 80% of the data with our optimized hyperparameters and tested our model on the remaining 20%. This test set acts as a completely unseen dataset, allowing us to evaluate how well our model generalizes to new data.

Our Random Forest Classifier achieved an accuracy of **89.7% on our test set**, which reflects the model's overall ability to correctly classify food categories based on the features extracted from our dataset.

It is important to note that our classifier also achieved a **training accuracy of 92.7%**, which is very close to the test accuracy. This similarity strongly implies that our model is not overfitted to the training set, but rather generalizes relatively well. Taken together, this suggests that we would likely see an accuracy of around ~89% as well on the unseen dataset.

Final Model Training Accuracy: 92.7%
Final Model Test Accuracy: 89.7%

Rationale

The expected strong performance of our model on the test set is supported by several factors. Firstly, our extensive grid search and hyperparameter tuning process, using 5-fold cross-validation, helped us ensure that our hyperparameters were not overfitted to a specific train/test split. The similar accuracy between our final model test (89.7%) and training sets (92.7%) further indicates that overfitting is not a significant concern.

As shown in Table 4, the Random Forest model consistently outperformed other models such as Logistic Regression, kNN, and MLPs, all of which were evaluated using 5-fold cross-validation. Furthermore, the Random Forest's ability to aggregate predictions across multiple decision trees enhances its robustness and generalization capabilities. This is particularly beneficial given the complexities and potential non-linear relationships present in our dataset and unseen test set.

Overall, empirical evidence from our cross-validation results, and high, yet similar, accuracies on our final test and training sets suggest that our model is well-equipped to handle the unseen dataset effectively. Therefore, we are confident that it will perform consistently well with our presently observed results.

Section 5. Acknowledgements and Workload Distribution

We would like to give our sincerest thanks to Dr. Alice Gao and the entire CSC311 teaching team for their unwavering support throughout this *extremely* long and arduous, yet highly rewarding, learning experience.

As for within our own team, each of us contributed extensively to all aspects of the project, though we did have some areas of specialization:

Franco Miguel Valencia, for writing the majority of the final report and designing the model along with its rationale;

Jeremy Xie, for optimizing the model, thoroughly testing its performance, testing alternative models, experimenting with features, and designing methods for using them;

Mykola Zhuk, for maintaining consistency and developing the final prediction package, ensuring its integrity;

And special thanks to Khoa Pham, who didn't make it with us to the end, but for showing immense courage in prioritizing his family.

Thanks to you all!