Classifier Model Process Overview

Since the objective of the exercise is to predict the food order from 10 given options, the given problem is one of **multi-class classification**.

The problem can be modelled in several ways. The following are a few options:

a) **Logistic regression:** The algorithm performs well on binary classification. Hence, we reframe the multi-class classification as a series of binary classification problems.
For example: For an order to be predicted as “Sweet potato fries”, the output of the Logistic regression on “Sweet potato fries Vs others” should yield a high probability in favor of sweet potato fries.

   For K classes, we need (k-1) binary classifications.

   This vastly decreases the reliability of our output as there are 9 model outputs to be evaluated.
   The final output would consider the probabilities shown by 9 models. It is possible in many cases, where arriving at a final output could be difficult and highly unreliable.

   Since we have a high number of classes, **we rule out Logistic regression** for the reasons mentioned above.

b) **Native classifier Models:** There are many classifier models who, by their nature, are capable of multi-class classification. These include:
   - Naïve-Bayes Classifier
   - Support Vector Machine
   - Random Forest Classifier
   - Decision Tree Classifier
   - K-Nearest Neighbors
   - AdaBoost Classifier

   Within these algorithms, we can fine tune the accuracy by changing the test-train split and the hyperparameters specific to the algorithm.

   We choose the algorithm and the parameters that best fits our purpose from the above options.

Data Preparation

The given data is made entirely of categorical columns. To use them as features, they must be converted to numerical form. We use **One-Hot Encoding** to achieve this transformation. It creates a new column for each distinct value in the given categorical column.

We also perform **feature engineering** on the data to add a new feature “Meal_type”. It takes up one of the three values {‘Breakfast’, ‘Lunch’, ‘Evening snack’} depending on the order time. This added feature increased the model accuracy by about 5% on average across the native models. This feature also is the second most correlated to the order.

Model selection

On implementing the native algorithms mentioned above, we observe that Naïve-Bayes, Decision tree and AdaBoost give less than desirable results. We rule them out on the grounds of low accuracy across multiple test-train splits and hyperparameters.

Random Forest Vs Support Vector Machine:
SVM and Random Forest models have proven to be significantly better than the other algorithms. We shortlist these two to enhance accuracy by hyperparameter tuning.

- Although SVM can handle multi-class classification, it was designed originally for binary classification and works best when there is a clear decision boundary.
- Random Forest extends to multi-class classification naturally and hence maybe better suited for our case.

SVM Hyperparameters:

a) kernel = ‘poly’
   The kernel parameter determines the type of function that determines the decision boundary. Out of the available options (‘Linear’, ‘poly’, ‘sigmoid’ and ‘rbf’) the ‘poly’ (referring to polynomial) yielded the best accuracy.
b) Degree = 4
   The degree parameter sets the degree of the polynomial in the ‘poly’ kernel. It is set to 3 by default but setting it at 4 yielded better results.

Random Forest Hyperparameters:

We rely on experimental data to tune the parameters. To choose from the numerous options available, we use the **RandomizedSearchCV** library in Python. This library identifies the best values for the hyperparameters to optimize the accuracy by trial and error on a large set of values.

For our case, we found the most optimal parameters to be:

```
{'n_estimators': 2000,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 100,
 'bootstrap': True}
```

On applying these parameters, we find that the Random Forest algorithm performs better than all other options and has an accuracy near about 65%.

Thus, we finalize on Random Forest Classifier to carry out our task. In the “Technical Implications” section, we inferred that Revenue is related to the model’s accuracy as:

$$\mathbf{R = (0.1a + 0.9)nX}$$

Where R = revenue in \$ in time t
a = model accuracy
n = Total number of orders in time t
X = revenue per order

On substituting a = 0.65, we get R = 0.965nX

Thus, our model can limit the opportunity loss due to wrong prediction to 3.5%