Elvyna Tunggawan
https://github.com/elvyna

Predicting E-Commerce Cart Abandonment

AWS Machine Learning Engineer Nanodegree
January 6, 2022

1. Definition

1.1. Project Overview

In the past decade, e-commerce platforms have been rapidly growing. On a survey on Organisation for Economic Co-operation and Development (OECD) countries, it was found that more than half of the individuals made at least one online purchase within the last 12 months[1]. Additionally, the Covid-19 pandemic resulted in a huge shift from brick-and-mortar business towards online shopping. In the UK, China, Germany, and the US, the share of retail sales from online purchases increased by 4-7% in 2020 compared to 2019[2]. As more people become more used to online purchases, there are increasing numbers of e-commerce platforms. To be a market leader, having a reliable platform and providing an amazing user experience become more important.

Although every e-commerce platform may require different purchasing steps, they mainly share a similar conversion funnel. To maximize their revenue, they strive to increase the overall conversion rate, i.e., aiming to convert as many web/app visitors into their customers.

One of the earlier steps of the conversion funnel is adding items into the shopping cart. Although having the cart filled is one step closer to the purchase, there are possibilities that the users will abandon their cart. For example, the users might consider a shopping cart as a helper to save the items that they are interested in. This misuse might happen since not all e-commerce platforms have a wishlist or bookmark feature.

In this project, we use the dataset provided by Data Mining Cup 2013[3], which contains 429,013 rows of e-commerce sessions with 24 columns. Each row represents the activity of each session at a particular hour of the day.

---

[1] *Unlocking the Potential of E-commerce*. (2019, March). OECD. https://www.oecd.org/going-digital/unlocking-the-potential-of-e-commerce.pdf
[2] Alonso, V., Boar, C., Frost, J., Gambacorta, L., and Liu, J. (2021, January 12). *E-commerce in the pandemic and beyond*. Bank for International Settlements. https://www.bis.org/publ/bisbull36.pdf
[3] Data Mining Cup 2013. https://www.data-mining-cup.com/reviews/dmc-2013/

1.2. Problem Statement

E-commerce platforms usually introduce nudge marketing to influence user behaviour and encourage them to purchase. However, nudges might backfire if they are implemented without understanding the user behaviour[4].

Analyzing users' purchasing intent has become one of the research areas in e-commerce. Most e-commerce platforms have pivoted to more personalized features and services, which is also the case for Pinterest[5]. Website clickstream and user session data are the main data sources to understand the purchasing intent since they contain the whole picture of the e-commerce platform (not only the bookings)[6,7]. By utilizing user activity logs to identify the likelihood of cart abandonment, e-commerce platforms could provide more personalized treatment to the users. For example, they might offer coupon codes to convert the abandoning users into customers and offer some upselling for users with a high likelihood to purchase.

Predicting shopping cart abandonment can be approached as a binary classification problem. We aim to train a model that accurately predicts the cart abandonment rate of each web session based on the provided information in the dataset. Before going straight into modelling, one of the challenges is to understand the key difference between abandoning and converting users. Hence, some exploratory analyses are required, which will inform the required data preprocessing steps and also lead to further hypotheses for the feature engineering and modelling steps.

1.3. Metrics

Since the original dataset has an imbalanced class distribution (only 33% of them are abandoned carts), we use the F_1 score to complement the prediction accuracy. The F_1 score is a harmonic mean of the model precision and recall, as displayed below.

[4] Sanghi, R., Gupta, S., Mishra, R.C., Singh, A., Abrol, H., Madan, J., & Sagar, M. (2019). *Nudge Marketing in E-Commerce Businesses*. International Journal of Science and Research (IJSR). https://www.ijsr.net/archive/v8i8/ART2020223.pdf

[5] Lo, C., Frankowski, D., & Leskovec, J. (2016). Understanding behaviors that lead to purchasing. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/2939672.2939729

[6] Sakar, C. O., Polat, S. O., Katircioglu, M., & Kastro, Y. (2018). Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. *Neural Computing and Applications*, 31(10), 6893-6908. https://doi.org/10.1007/s00521-018-3523-0

[7] Kompan, M., Kassak, O., & Bielikova, M. (2019). The Short-term User Modeling for Predictive Applications. *Journal on Data Semantics, 8(1)*, 21–37. https://doi.org/10.1007/s13740-018-0095-1

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

We use a similar weight for precision and recall because we aim to have a model that performs well both on the checked out and abandoned carts. If the e-commerce platforms have different risks if they incorrectly predict a particular class, we may consider using a different weight in the F-measure formula. However, it is out of the scope of this project.

2. Analysis

2.1. Data Exploration

The original dataset does not contain any null values in any of its 24 columns. However, as illustrated in the table below, the missing values are encoded as "?".

Table 2.1. Preview of the dataset content.

Column name	Value	Data type
sessionNo	1	int64
startHour	6	int64
startWeekday	5	int64
duration	0.0	float64
cCount	1	int64
cMinPrice	59.99	object
cMaxPrice	59.99	object
cSumPrice	59.99	object
bCount	1	int64
bMinPrice	59.99	object
bMaxPrice	59.99	object
bSumPrice	59.99	object
bStep	?	object
onlineStatus	?	object
availability	?	object
customerNo	1	object

maxVal	600	object
customerScore	70	object
accountLifetime	21	object
payments	1	object
age	43	object
address	1	object
lastOrder	49	object
order	y	object

Most numeric columns, such as *cMinPrice* and *cMaxPrice* are stored as string objects since they contain "?". Hence, we need to replace those "?" values accordingly and convert the data types of those columns into numeric. Our target class is the *order* column, which has an imbalanced distribution. We might consider either resampling the data or generating synthetic samples to have a balanced class distribution.

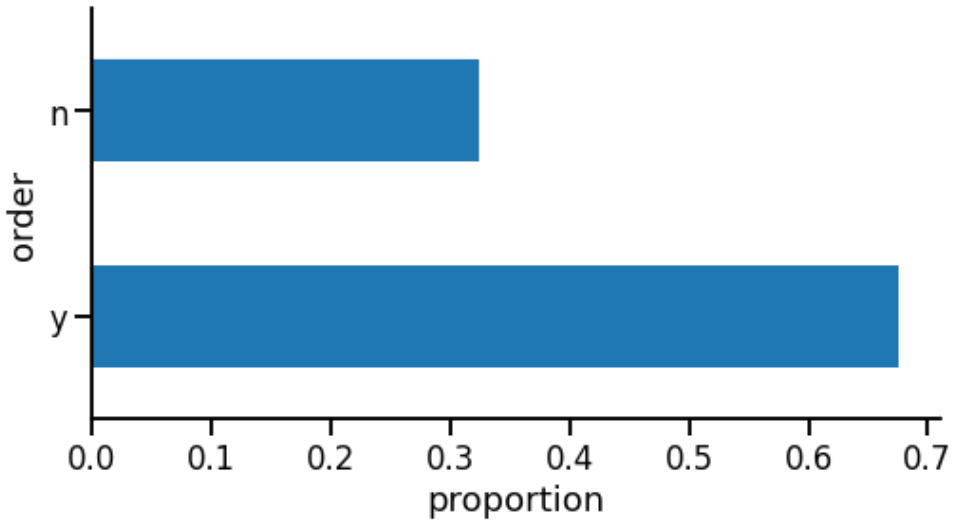


Figure 2.1. Only 33% of the records are abandoned sessions with abandoned carts.

Although the data dictionary mentioned that *startWeekday* encoding starts with 1 (for Monday) and ends with 7 (for Sunday), the dataset only contains 5, 6, and 7 values in this column. We also identify potential outliers on each numeric column using the interquartile range rule. A record is considered as an outlier if it satisfies one of the following conditions:

- value < $Q_1 * 1.5 (Q_3-Q_1)$
- value > $Q_3 * 1.5 (Q_3-Q_1)$

The following table shows the number of identified outliers for each column.

Table 2.2. Summary of the identified outliers in the whole dataset using the interquartile range rule.

Column name	Outlier count	% of outliers
startHour	3,630	0.85%
startWeekday	0	0%
duration	35,625	8.3%
cCount	33,970	7.92%
cMinPrice	59,980	13.98%
cMaxPrice	70,443	16.42%
cSumPrice	45,186	10.53%
bCount	25,780	6.01%
bMinPrice	67,004	15.62%
bMaxPrice	68,943	16.07%
bSumPrice	46,650	10.87%
bStep	0	0%
maxVal	18,217	4.25%
customerScore	0	0%
accountLifetime	31	0.01%
payments	1,772	0.41%
age	0	0%
lastOrder	20.289	4.73%

2.2. Exploratory Visualization

To get a high-level understanding of the data, we compute the Pearson correlation of the numeric features. As mentioned before, we need to convert the data types after handling the “?” values. Here, we convert “?” into -99. The original dataset does not contain any negative values, hence, we can use it to represent the missing values. Additionally, we convert “y” into 1 and “n” into 0.

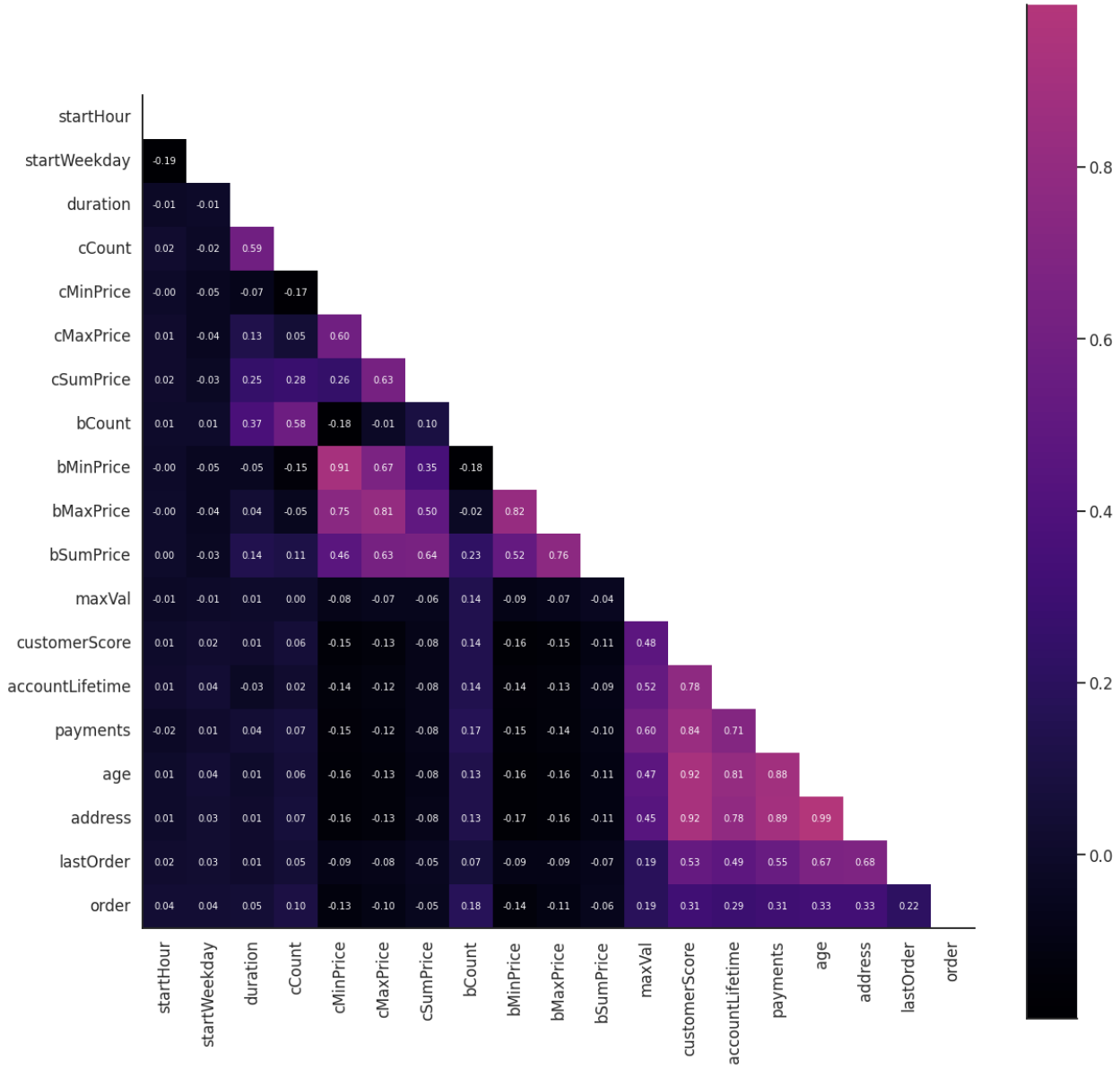


Figure 2.2. Correlation matrix of the full dataset.

Our main focus is the *order* column, which has negative correlations with price-related features ($-0.14 \leq r \leq -0.05$), i.e., carts with higher prices tend to be abandoned. Carts with more items are more likely to be checked out ($r=0.18$). Customers who are perceived well by the stores and those who are long-time customers also tend to check out their carts ($r=0.31$ and $r=0.29$, respectively). Users' age is also positively correlated with checked-out carts ($r=0.33$), and Figure 2.3 shows that checked-out carts are mostly created by users between 25 and 60 years old.

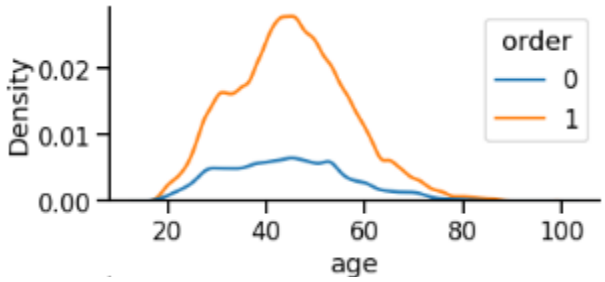


Figure 2.3. Distribution of age in checked-out (order=1) and abandoned carts (order=0).

2.3. Algorithms and Techniques

In our initial iteration, we start with a logistic regression model with basic hyperparameters without any tuning. This model provides good interpretability and does not require too many computing resources. Based on this initial result, we continue with more complex algorithms, i.e., Random Forest, LightGBM, and CatBoost. We choose tree-based models rather than deep learning models since we want to have more interpretable results and save computing time. This reasoning is also commonly found in real-world use cases as the stakeholders avoid using black-box machine learning models.

2.4. Benchmark

The provided dataset contains 67% of confirmed orders. As a simple benchmark, we aim to train a model with higher prediction accuracy than this naive prediction. We could not find past research that uses this dataset. However, as an additional benchmark, Sakar et al. achieved between 87% and 89% of prediction accuracy using multilayer perceptron, tree-based models, and support vector machine (SVM)[5]. The best F_1 score they achieved was 0.58 using either random forest or multilayer perceptron.

In our work, we use the initial results from the logistic regression model as our baseline results. Then, we aim to improve the results by using other algorithms and iterating on the modelling steps (from data preparation and feature engineering until hyperparameter tuning).

3. Methodology

3.1. Data Preprocessing

As discussed in Section 2.1, we handle missing values (which are encoded as “?”) by replacing them with -99. Then, we convert the relevant columns that are originally stored as string objects into numeric. Additionally, we convert “y” and “n” values that represent either True or False into 1 and 0. Before we proceed to further steps, we also remove ID-related columns, i.e., *sessionNo*

and *customerNo*. Despite being an identifier for specific users, we should train the model based on the user behaviour, not the identity of the users, which could introduce unfair treatment to a particular group of users[8].

To get a fair evaluation of the model performance, we split the dataset into training and test set with 70:30 proportion. Operations regarding data preparation and feature engineering are determined based on our knowledge of the training set, while we leave the test set since it represents the unknown real-world dataset for our final evaluation. This train-test split process returns 300,309 and 128,704 rows for the training and test sets, respectively.

We take a conservative approach to handle the outliers. As displayed in Table 2.2, the number of potential outliers vary depending on the column. In this work, we remove rows that contain a potential outlier in at least one of the columns. Note that we only remove outliers in the training set since it should represent the unseen dataset.

```
01-split-train-test-data.py 2022-01-04 13:03:44+1300 INFO Line no: 82 Identified outliers
           colnames  outliers_count  outliers_pct
0         startHour          3630.0      0.846128
0      startWeekday             0.0      0.000000
0          duration         35625.0      8.303944
0             cCount         33970.0      7.918175
0         cMinPrice         59980.0     13.980928
0         cMaxPrice         70443.0     16.419782
0         cSumPrice         45186.0     10.532548
0             bCount         25780.0      6.009142
0         bMinPrice         67004.0     15.618175
0         bMaxPrice         68943.0     16.070142
0         bSumPrice         46650.0     10.873796
0             bStep             0.0      0.000000
0      onlineStatus             0.0      0.000000
0            maxVal         18217.0      4.246258
0    customerScore             0.0      0.000000
0  accountLifetime            31.0      0.007226
0         payments          1772.0      0.413041
0              age             0.0      0.000000
0        lastOrder         20289.0      4.729227
01-split-train-test-data.py 2022-01-04 13:03:44+1300 INFO Line no: 110 Row count before outlier removal: 300,309
01-split-train-test-data.py 2022-01-04 13:03:46+1300 INFO Line no: 122 Row count after outlier removal: 173,009 (57.610% of the original rows)
```

Figure 3.1. We remove outliers identified on the training set, which leaves 57% of the original training set.

As discussed in Section 2.2, the dataset has an imbalanced class distribution. Although we already choose the F_1 score as the evaluation metric, not all algorithms can learn well from imbalanced datasets. Hence, we generate synthetic samples of the minority class (abandoned carts) in the training set[9]. We do not use undersampling because it will decrease the dataset size and might remove important information that exists in the dataset.

For the feature engineering steps, we create *time_of_day* column based on *startHour*. This is a way of binning the 24 discrete values of the hour of the day, which might allow the model to learn better. We also perform one-hot encoding on the categorical columns since not all algorithms can handle categorical features. Although it may result in sparse features, we deem it is more suitable

[8] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys*, 54(6), 1-35. https://doi.org/10.1145/3457607

[9] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16, 321-357. https://doi.org/10.1613/jair.953

than using the label encoder, as the dataset does not have any ordering within the label definition.

3.2. Implementation

For our exploration and development stages, we mainly work on local machines to allow faster iterations. Then, once we've got a working code and require more computing power (e.g., for hyperparameter tuning), we start running our code as Sagemaker training jobs.

Most operations mentioned in Section 3.1 could be done using modules provided on Scikit Learn and imbalanced-learn packages. On the outlier identification step, we implement a custom function for the interquartile range rule.

```
def check_outliers(series: pd.Series):
    """
    Return outlier flag and the number of outliers in the series.
    Outliers: if < Q1 - 1.5*IQR or > Q3 + 1.5*IQR

    :param series: series of numeric values to be analysed
    :type series: pd.Series
    :return: outlier flag (pd.Series) and number of outliers in that series
    :rtype: tuple
    """
    q1, q3 = np.percentile(series, q=[25, 75])
    iqr = q3 - q1
    lower_threshold = q1 - (1.5 * iqr)
    upper_threshold = q3 + (1.5 * iqr)
    conditions = [(series < lower_threshold) | (series > upper_threshold)]
    choice = [1]
    outlier_flag_list = np.select(conditions, choice, default=0)

    return outlier_flag_list, sum(outlier_flag_list)
```

In addition to the hold-out test set, we also use k-fold cross-validation to have an unbiased estimation of the model performance. These cross-validation results are later used for our hyperparameter tuning step, i.e., we want to use a model that has the best F_1 score. Then, we use the test set as the final evaluation dataset. On our work, we use k=5.

```
def evaluate_performance(
    y_true: pd.Series,
    y_pred: pd.Series,
    print_results: bool = True,
    evaluation_type: str = "test",
):
    """
    Evaluate classification model performance.

    :param y_true: actual class
    :type y_true: pd.Series
    :param y_pred: predicted class
```

```
    :type y_pred: pd.Series
    :param print_results: if True, show evaluation metrics in the console,
defaults to True
    :type print_results: bool, optional
    :param evaluation_type: dataset used to generate y_pred, e.g., train,
validation, or test
    :type evaluation_type: str
    :return: tuple of three elements (accuracy, F1 score, and AUC)
    :rtype: tuple
    """
    assert evaluation_type in [
        "train",
        "validation",
        "test",
    ], f"Invalid {evaluation_type}! It must be either 'train', 'validation',
or 'test'."

    accuracy = accuracy_score(y_true, y_pred)
    f_score = f1_score(y_true, y_pred)
    auc = roc_auc_score(y_true, y_pred)

    if print_results:
        log.info(f"{evaluation_type} set - Accuracy : {accuracy:.3%}")
        log.info(f"{evaluation_type} set - F1-score : {f_score:.3%}")
        log.info(f"{evaluation_type} set - AUC: {auc:.3f}")
        log.info(classification_report(y_true, y_pred))

    return accuracy, f_score, auc


def kfold_cv(clf, X_train: pd.DataFrame, y_train: pd.Series, k: int = 10,
random_state: int = 121):
    """
    Run k-fold cross-validation by splitting the training set into train and
validation set.

    :param clf: model object / estimator
    :type clf: [type]
    :param X_train: features of training set
    :type X_train: pd.DataFrame
    :param y_train: actual target class of the training set
    :type y_train: pd.Series
    :param k: number of folds for CV, defaults to 10
    :type k: int, optional
    :param random_state: random seed to allow reproducible results, defaults
to 121
    :type random_state: int, optional
    """
    kfold = KFold(random_state=random_state, shuffle=True, n_splits=k)

    cv_accuracy = np.zeros(shape=k)
    cv_f1 = np.zeros(shape=k)
    cv_auc = np.zeros(shape=k)

    i = 0
```

```
    for train_index, val_index in kfold.split(X_train):
        X_tr = X_train.loc[train_index]
        y_tr = y_train.loc[train_index]
        X_val = X_train.loc[val_index]
        y_val = y_train.loc[val_index]

        clf.fit(X_tr, y_tr)
        y_pred = clf.predict(X_val)
        accuracy, f_score, auc = evaluate_performance(
            y_val, y_pred, print_results=False
        )
        log.info(
            f"Iteration {i+1}: Accuracy={accuracy:.3%} |
F1-score={f_score:.3%} | AUC={auc:.3f}"
        )

        cv_accuracy[i] = accuracy
        cv_f1[i] = f_score
        cv_auc[i] = auc

        i += 1

    log.info("Cross-validation results")
    log.info("========================")
    log.info(f"CV Accuracy: {np.mean(cv_accuracy):.3%} +-
{np.std(cv_accuracy):.3%}")
    log.info(f"CV F1-score: {np.mean(cv_f1):.3%} +- {np.std(cv_f1):.3%}")
    log.info(f"CV AUC: {np.mean(cv_auc):.3f} +- {np.std(cv_auc):.3f}"
```

3.3. Refinement

Our initial model was trained using all features (33 features). According to Occam's Razor, it is preferred to have a simpler model. Hence, we decrease the number of features using Recursive Feature Elimination (RFE). It works by training the model using all feature sets, then iteratively removing the least important feature until it ends up with the specified number of features[10]. In this work, we retain 70% of the features, i.e., we keep the 24 most important features. These features are derived by recursively training random forest model on the training set.

In our initial iterations (available on the notebook directory), random forest consistently have the best evaluation results (both on the cross-validation and the hold-out test set). Hence, to save our resources, we only do the hyperparameter tuning on this algorithm. Since random forest is available on Sagemaker's Scikit Learn image, we do not need to create a custom Docker container. Our setup for the hyperparameter tuning step is provided on notebook/sagemaker/00-sm-hp-tuning.ipynb. The following figure shows our hyperparameter search ranges, where we aim to maximize the F_1 score in our cross-validation step.

[10] Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1-3), 389-422. https://doi.org/10.1023/A:1012487302797

```python
from sagemaker.tuner import (
    IntegerParameter,
    CategoricalParameter,
    ContinuousParameter,
    HyperparameterTuner,
)

role = sagemaker.get_execution_role()

## declare your HP ranges, metrics etc.
hyperparameter_ranges = {
    "n_estimators": CategoricalParameter([100, 150, 200, 300]),
    "max_depth": CategoricalParameter([15, 20, 25, 30]),
    "min_samples_split": CategoricalParameter([2, 10, 20])
}

objective_metric_name = "cv f1-score"
objective_type = "Maximize"
metric_definitions = [{"Name": "cv f1-score", "Regex": "CV F1-score: ([0-9\\.]+)"}]
```

```python
## create estimators for your HPs
from sagemaker.sklearn.estimator import SKLearn

estimator = SKLearn(
    entry_point="../../src/modelling/train-rf.py",
    role=role,
    py_version='py3',
    framework_version="0.20.0",
    instance_count=1,
    instance_type="ml.m5.2xlarge",
)

## set hp tuner
tuner = HyperparameterTuner(
    estimator=estimator,
    objective_metric_name=objective_metric_name,
    hyperparameter_ranges=hyperparameter_ranges,
    metric_definitions=metric_definitions,
    max_jobs=6,
    max_parallel_jobs=2,
    objective_type=objective_type,
    base_tuning_job_name='rf-hp-tuning'
)
```

Figure 3.2. Sagemaker hyperparameter tuning job definition.

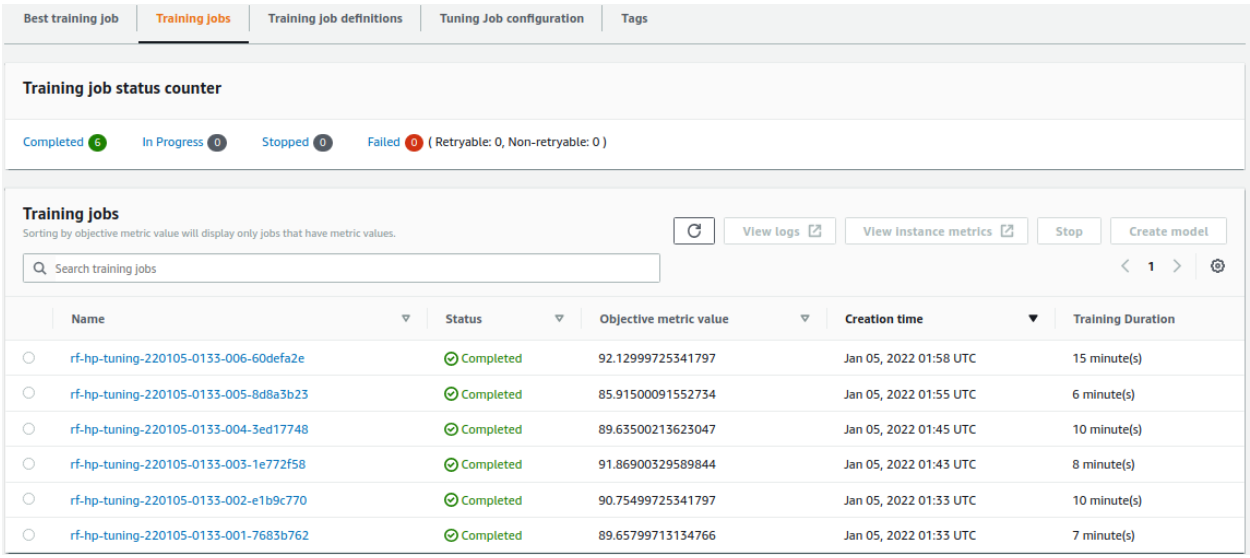


Figure 3.3. Sagemaker hyperparameter tuning job results. The hyperparameter tuner optimizes the cross-validation F_1 score.

4. Results

4.1. Model Evaluation and Validation

We conducted several iterations in our modelling attempts. In the beginning, we also trained the model without handling the imbalanced class distribution (denoted as Iteration 1). On Iteration 2, we have generated synthetic samples on the training set, using 65 nearest neighbours as one of SMOTE input parameters. In Iteration 3, we decrease the number of features using recursive feature elimination (find more details in Section 3.3). In each of these iterations, we train four models: logistic regression, random forest, Light GBM, and CatBoost. In Iteration 1 to 3, we mainly use the default model hyperparameters and only make a few changes:

Table 4.1. Default hyperparameters used in Iteration 1, 2 and 3.

Algorithm	**Hyperparameters**
Logistic regression	max_iter = 200 solver = ‘saga’ penalty = ‘elasticnet’ l1_ratio = 0.2
Random forest	bootstrap = True ccp_alpha = 0.0 criterion = ‘gini’ max_depth = None max_features = ‘auto’ max_leaf_nodes = None min_samples_leaf = 1

	min_samples_split = 2 n_estimators = 100
Light GBM	boosting_type = 'gbdt' objective = 'binary' colsample_bytree = 0.8 learning_rate = 0.2 subsample = 0.7 subsample_freq = 3 reg_alpha = 0.1 reg_lambda = 0.1 max_depth = -1
CatBoost	Iterations = 100 learning_rate= 0.2 max_depth = 10 l2_leaf_reg = 0.1

As displayed in Table 4.2, random forest consistently outperforms the other algorithms. Furthermore, eliminating 30% of the features only have a negligible decrease in the model performance. Hence, we keep using the reduced feature sets and refine the random forest model by finding hyperparameters that generate the best F_1 score in the cross-validation results. The resulting hyperparameters are stored and used to train our final model (Iteration 4). The hyperparameters are:

- max_depth = 30
- min_samples_split = 2
- n_estimators = 300

Figure 4.1 shows the model training logs of the random forest model in Iteration 4. The evaluation results of Iteration 1, 2, and 3 are available under the notebook/report directory.

Table 4.2. Comparison of model performance in each iteration.

Algorithm	**Description**	**Cross-validation**		**Test**	
		F_1 score	**Accuracy**	**F_1 score**	**Accuracy**
Logistic Regression	Iteration 1	81.891%	73.284%	71.578%	80.527%
Random Forest	Iteration 1	91.565%	87.991%	87.343%	82.325%
Light GBM	Iteration 1	86.043%	79.647%	83.055%	76.030%
CatBoost	Iteration 1	88.894%	84.089%	85.001%	79.025%
Logistic Regression	Iteration 2	69.032%	67.691%	75.565%	68.324%
Random Forest	Iteration 2	92.000%	91.862%	87.463%	82.396%

Light GBM	Iteration 2	85.540%	84.829%	83.252%	76.318%
CatBoost	Iteration 2	88.617%	88.279%	84.620%	78.146%
Logistic Regression	Iteration 3	69.029%	67.689%	75.566%	68.326%
Random Forest	Iteration 3	92.020%	91.885%	87.415%	82.347%
Light GBM	Iteration 3	85.539%	84.814%	83.402%	76.237%
CatBoost	Iteration 3	88.491%	88.143%	84.883%	78.657%
Random Forest	**Iteration 4**	**92.130%**	**91.986%**	**87.581%**	**82.506%**

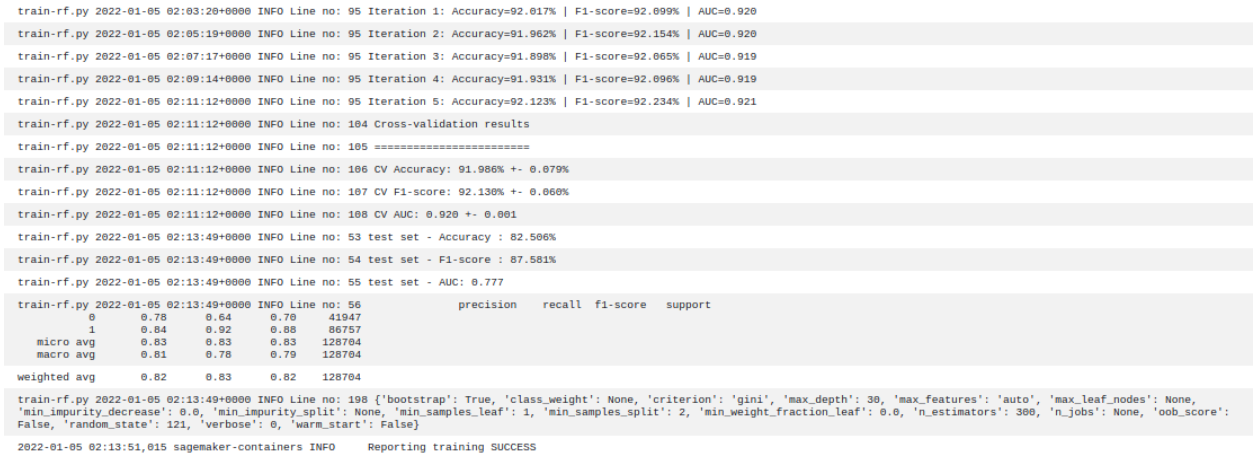


Figure 4.1. Cloudwatch logs of Iteration 4 (random forest with tuned hyperparameters).

4.2. Justification

Our final model results in 0.875 F_1 score and 82.506% prediction accuracy in our test set, which is much better than our benchmark model (0.58 F_1 score and 67% prediction accuracy). These results mean the model can predict checked out carts pretty well, hence, the e-commerce platforms could give some personalized treatments without having too much risk of giving incorrect treatments. For instance, users who are predicted to abandon their carts may receive a label in their carts to encourage them to purchase their items. Figure 4.2. shows the feature importance values of the final random forest model (from Iteration 4). As we observe in our exploratory analysis, price and customer-related features are considered important to predict cart abandonment. Interestingly, some of the most important features are rather unexpected: *startHour* and *duration*. The model could capture this information even if the relationship is not that obvious, since this random forest model is an ensemble of 300 individual decision trees (weak learners).

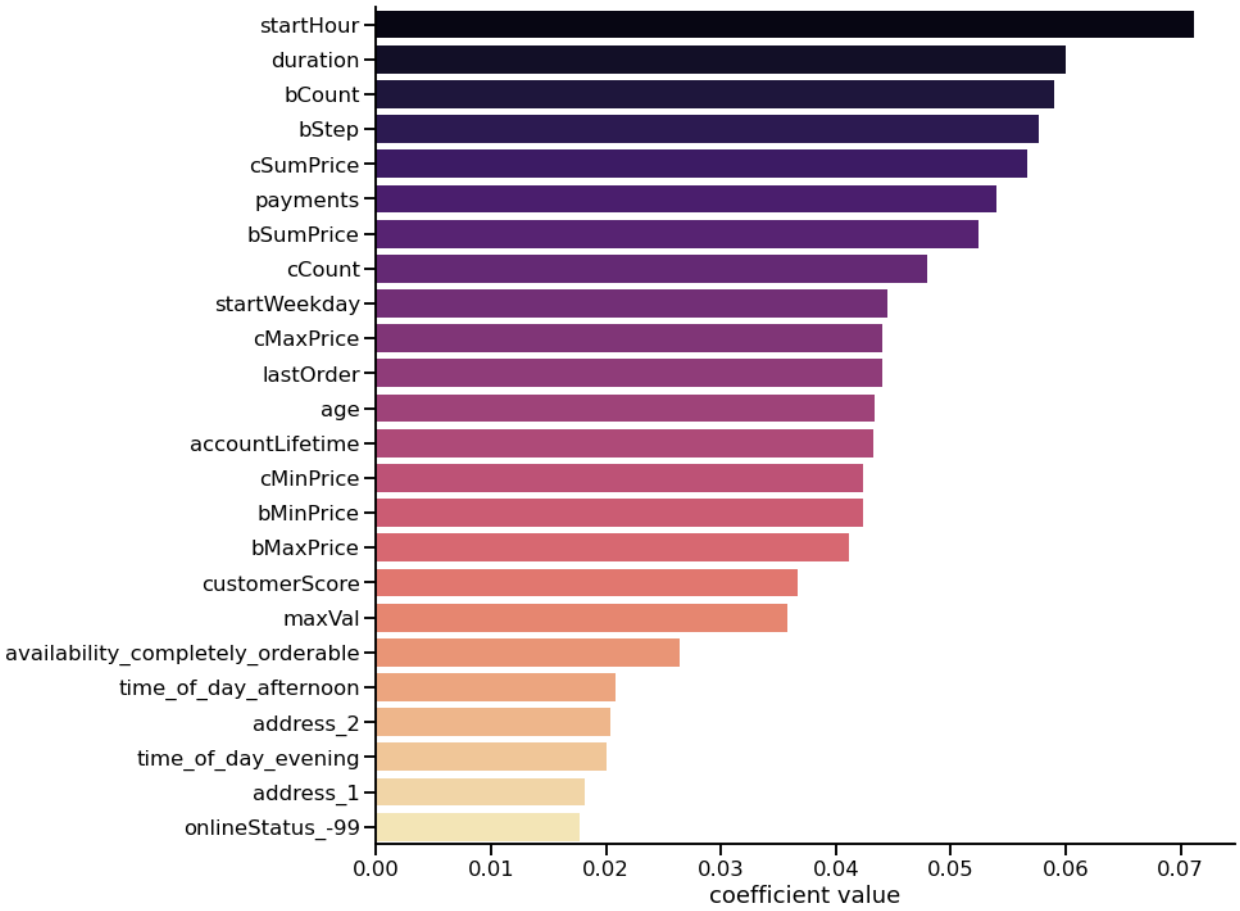
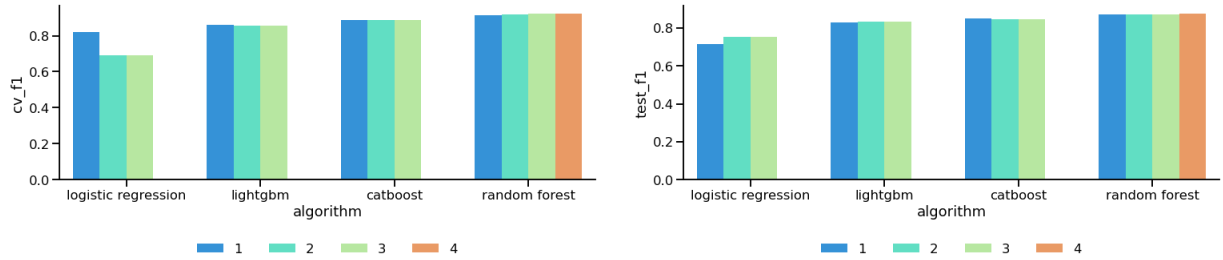


Figure 4.2. Features used in the final random forest model and their importance values.

Figure 4.3 shows the comparison of the model performance in our four iterations. Random forest shows a consistently high F1 score in both the cross-validation results and the test set. The difference between the cross-validation and test results are not huge, i.e., the model does not show a sign of overfitting.



(a) F_1 score in the cross-validation results

(b) F_1 score in the test set

Figure 4.3. Comparison of F_1 scores of the different algorithms in our iterations.

We deploy the final model as a Sagemaker endpoint, which demonstrates the possibility for e-commerce systems to utilise our modelling results. Figure 4.4 and 4.5 illustrates how we can

deploy the model and send a request to the Sagemaker endpoint. To enable the model deployment, we prepare a custom endpoint script that handles the preprocessing logic of the input data and pass it to the model object.

```
from sagemaker.sklearn.model import SKLearnModel
from sagemaker.serializers import JSONSerializer

model_location = 's3://sagemaker-us-east-1-567220378588/sagemaker-scikit-learn-2022-01-05-02-18-31-464/output/model.tar.gz'
sklearn_model = SKLearnModel(
    model_data=model_location,
    role=role,
    entry_point='../../src/modelling/inference.py',
    py_version='py3',
    framework_version='0.20.0',
)

predictor = sklearn_model.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.large',
    serializer=JSONSerializer()
)

predictor
```

```
------!
<sagemaker.sklearn.model.SKLearnPredictor at 0x7fee413b8790>
```

Figure 4.4 Deploy the model object as a Sagemaker endpoint.

```
response = predictor.predict(
#     json.dumps(payload), ## not needed since we've used JSONSerializer()
    payload,
    initial_args={
        "ContentType": "application/json"
    }
)

print(response)
```

```
{"data": {"sessionNo": 101, "startHour": 4, "startWeekday": 7, "duration": 0, "cCount": 2, "cMinPrice": 30, "cMaxPrice": 40, "cSumPrice": 70, "bCount": 1, "bMinPrice": 30, "bMaxP
rice": 30, "bSumPrice": 30, "bStep": "?", "onlineStatus": "?", "availability": "?", "customerNo": 39, "maxVal": 200, "customerScore": 65, "accountLifetime": 30, "payments": 2, "a
ge": 39, "address": 1, "lastOrder": 30}}
[0]
```

Figure 4.5. Pass a JSON request to the endpoint and receive the predicted class.

5. Conclusion

5.1. Reflection

In this project, we demonstrate the process of an end-to-end machine learning project. Usually, data scientists focus on problem understanding, research, and modelling iterations, which is conducted using Jupyter notebooks for faster experimentations. However, to ensure the work could be utilised in real-world operations, machine learning engineers take over the working solutions from data scientists and create a production-level code for the automation. Although our work is not perfect yet (see Section 5.2), the best model performs better than our benchmark (both in terms of F_1 score and prediction accuracy), and we have done multiple iterations to improve our initial baseline model.

5.2. Improvement

While the model performs better than our benchmark and is ready to use in a production setting, there are more potential improvements for further work. The improvements could be in the research and modelling point of view, as well as in the engineering and machine learning operations side.

a. Modelling
    i. Further exploratory analysis to identify more important features to be engineered
    ii. Try using embedding methods, such as autoencoders
    iii. Explore different algorithms, such as using neural networks and Shapley values to allow model interpretability
    iv. Further hyperparameter tuning
b. Engineering and ML operations
    i. Code refactoring for better code readability and maintenance
    ii. Move hard-coded variables names into configuration files
    iii. Write unit tests and data validation scripts to ensure data quality
    iv. Orchestrate the end-to-end scripts as step functions (require containerization) or as a Sagemaker Pipeline (run the data preprocessing and feature engineering steps as Sagemaker Processing jobs)
    v. Model monitoring and retraining