Machine Learning Project (R4CO3012P)

E-commerce Shopper's Behaviour Understanding:
Understanding shopper's purchasing pattern through Machine Learning

Batch - D

Student Name	Enrollment Number
Humanshu Dilipkumar Gajbhiye	221070801
Bhavika Milan Purav	211071902
Anisha Pravin Jadhav	201071044
Mayuri Premdas Pawar	201071026

Veermata Jijabai Technological Institute
H.R. Mahajani Road, Matunga
Mumbai, India
May, 2023

Contents

1 | Introduction

Machine Learning (ML) is a powerful technique that enables computers to learn patterns and make predictions based on data. Binary classification, one of the fundamental tasks in ML, focuses on categorizing data into two distinct classes or categories. In this beginner-level ML project, our goal is to build a binary classification model that can accurately classify new instances into one of the two predefined classes. The ability to perform binary classification has numerous applications across various domains. By training a binary classification model, we can leverage historical data to make informed decisions and automate the classification process. For example, we will be using it to predict whether the customer purchases the product or not based on the clickstream data.

Throughout this project, we will explore different aspects of building a binary classification model. We will follow the essential steps such as exploratory data analysis, data preprocessing, feature selection, model training, hyperparameter tuning, evaluation, and prediction. Additionally, we will delve into common evaluation metrics for assessing the performance of the model, including accuracy, precision, recall, and F1 score. And the final evaluation metric will remain to be the F1 score.

By the end of this project, we will gain hands-on experience in implementing a binary classification, understanding the nuances of working with labeled data, and effectively evaluating the model's performance. This project will serve as a solid foundation for further exploration into more advanced ML concepts and techniques, allowing us to tackle more complex classification problems and contribute to a wide range of real-world applications.

As a beginner, we wanted to implement our learning and the skills we picked up in our Lab sessions, to a real-world problem. We explored Kaggle for datasets and found a competition[2] that had the following description and evaluation criteria.

1.1 | Description

Assume that you are working in a consultancy company and one of your client is running an e-commerce company. They are interested in understanding the customer behavior regarding the shopping. They have already collected the users' session data for a year. Each row belongs to a different user. The Made_purchase is an indicator that whether the user has made a purchase or not during that year. Your client is also interested in predicting that column using other attributes of the users. The client also informs you that the data is collected by non-experts. So, it might have some percentage of error in some columns.

1.2 | Evaluation

The evaluation metric for this competition is Mean F1-Score. The F1 score, commonly used in information retrieval, measures accuracy using the statistics precision. The F1 metric weights recall and precision equally, and a good retrieval algorithm will maximize both precision and recall simultaneously. Thus, moderately good performance on both will be favored over extremely good performance on one and poor performance on the other.

1.3 | Submission format

The CSV file should contain a header and have the following format:

id	*Made_Purchase*
1	*False*

2 | Methodology

Our problem statement is derived from Kaggle Competition[2]. For this binary classification problem we followed the standard ML workflow snd took inspiration from a research paper by Sabina-Cristiana Necula[1].

2.1 | Data Collection

We used the data from a Kaggle competition.

2.2 | Preprocessing

Perform missing value imputation, encoding, scaling and outlier removal.

2.3 | Exploratory Data Analysis

Create standard visualisation for the processed data to observe some patterns.

2.4 | Model Selection

Explore all the candidate models for the problem and then do a rough evaluation for each one.

2.5 | Training

Train all the candidate models on the training data set.

2.6 | Evaluation

Calculate the evaluation metrics for all the candidate models.

2.7 | Hyperparameter Tuning

Tune the hyperparameters of all the candidate algorithms and re-evaluate the metrics.

2.8 | Prediction

Select the best performing model based on the defined evaluation metric and then us it to predict the test data set.

3 | Approach

We adhered to the standard methodology and implemented functionalities using the following libraries:

- SciPy
- NumPy
- Pandas
- Seaborn
- Imblearn
- Matplotlib
- Scikit-learn
- Base Python libraries

3.1 | Imports

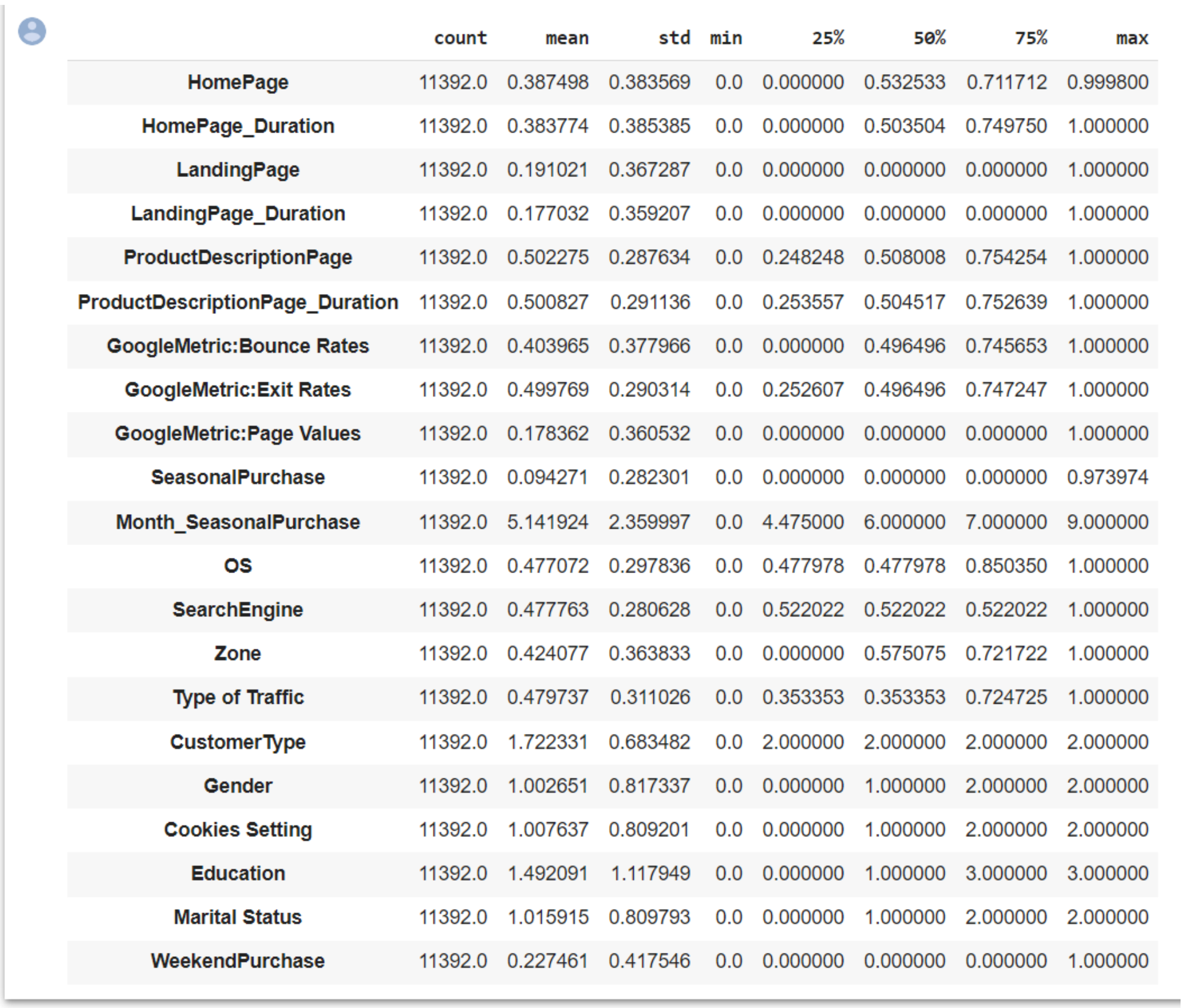
Standard imports for libraries.

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import xgboost as xgb, scipy as sp, matplotlib as mpl, seaborn as sns
from imblearn import under_sampling, over_sampling
from scipy import stats
from sklearn import set_config
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer, KNNImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder,
↪  QuantileTransformer, OrdinalEncoder
```

3.2 | Data Collection

We used the data from a Kaggle competition[2].

```
test_data =
↪  pd.read_csv("https://raw.githubusercontent.com/HumanshuDG/CS2008P/main/test_data.csv")
train_data =
↪  pd.read_csv("https://raw.githubusercontent.com/HumanshuDG/CS2008P/main/train_data.csv")
sample_data =
↪  pd.read_csv("https://raw.githubusercontent.com/HumanshuDG/CS2008P/main/sample.csv")

train_data.shape, test_data.shape
```



	count	mean	std	min	25%	50%	75%	max
HomePage	11392.0	0.387498	0.383569	0.0	0.000000	0.532533	0.711712	0.999800
HomePage_Duration	11392.0	0.383774	0.385385	0.0	0.000000	0.503504	0.749750	1.000000
LandingPage	11392.0	0.191021	0.367287	0.0	0.000000	0.000000	0.000000	1.000000
LandingPage_Duration	11392.0	0.177032	0.359207	0.0	0.000000	0.000000	0.000000	1.000000
ProductDescriptionPage	11392.0	0.502275	0.287634	0.0	0.248248	0.508008	0.754254	1.000000
ProductDescriptionPage_Duration	11392.0	0.500827	0.291136	0.0	0.253557	0.504517	0.752639	1.000000
GoogleMetric:Bounce Rates	11392.0	0.403965	0.377966	0.0	0.000000	0.496496	0.745653	1.000000
GoogleMetric:Exit Rates	11392.0	0.499769	0.290314	0.0	0.252607	0.496496	0.747247	1.000000
GoogleMetric:Page Values	11392.0	0.178362	0.360532	0.0	0.000000	0.000000	0.000000	1.000000
SeasonalPurchase	11392.0	0.094271	0.282301	0.0	0.000000	0.000000	0.000000	0.973974
Month_SeasonalPurchase	11392.0	5.141924	2.359997	0.0	4.475000	6.000000	7.000000	9.000000
OS	11392.0	0.477072	0.297836	0.0	0.477978	0.477978	0.850350	1.000000
SearchEngine	11392.0	0.477763	0.280628	0.0	0.522022	0.522022	0.522022	1.000000
Zone	11392.0	0.424077	0.363833	0.0	0.000000	0.575075	0.721722	1.000000
Type of Traffic	11392.0	0.479737	0.311026	0.0	0.353353	0.353353	0.724725	1.000000
CustomerType	11392.0	1.722331	0.683482	0.0	2.000000	2.000000	2.000000	2.000000
Gender	11392.0	1.002651	0.817337	0.0	0.000000	1.000000	2.000000	2.000000
Cookies Setting	11392.0	1.007637	0.809201	0.0	0.000000	1.000000	2.000000	2.000000
Education	11392.0	1.492091	1.117949	0.0	0.000000	1.000000	3.000000	3.000000
Marital Status	11392.0	1.015915	0.809793	0.0	0.000000	1.000000	2.000000	2.000000
WeekendPurchase	11392.0	0.227461	0.417546	0.0	0.000000	0.000000	0.000000	1.000000

Figure 3.1: Summary of Training Data

3.3 | Preprocessing

Perform missing value imputation, encoding, scaling and outlier removal.

```
train_data.replace('nan', np.nan, inplace = True)
num = train_data.select_dtypes(include = ['int64', 'float64']).columns.tolist()
cat = train_data.select_dtypes(include = ['object']).columns.tolist()
# si_mean = SimpleImputer(missing_values = np.nan, strategy = 'mean')
# train_data[num] = si_mean.fit_transform(train_data[num])
# si_mode = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
# train_data[cat] = si_mode.fit_transform(train_data[cat])
oe = OrdinalEncoder()
train_data[cat] = oe.fit_transform(train_data[cat])
knn = KNNImputer(n_neighbors = 10)
train_data[num] = knn.fit_transform(train_data[num])
train_data[cat] = knn.fit_transform(train_data[cat])
ss = StandardScaler()
train_data[num] = ss.fit_transform(train_data[num])
qt = QuantileTransformer(output_distribution = 'uniform')
train_data[num] = qt.fit_transform(train_data[num])
test_data.replace('nan', np.nan, inplace = True)
num = test_data.select_dtypes(include = ['int64', 'float64']).columns.tolist()
cat = test_data.select_dtypes(include = ['object']).columns.tolist()
# si_mean = SimpleImputer(missing_values = np.nan, strategy = 'mean')
oe = OrdinalEncoder()
test_data[cat] = oe.fit_transform(test_data[cat])
knn = KNNImputer(n_neighbors = 10)
test_data[num] = knn.fit_transform(test_data[num])
test_data[cat] = knn.fit_transform(test_data[cat])
ss = StandardScaler()
test_data[num] = ss.fit_transform(test_data[num])
qt = QuantileTransformer(output_distribution = 'uniform')
test_data[num] = qt.fit_transform(test_data[num])
```

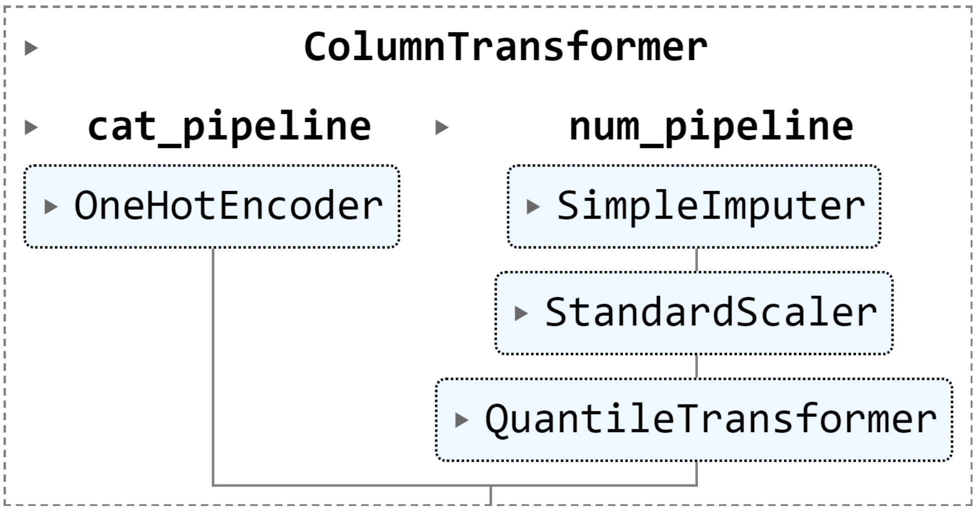


Figure 3.2: Pre-processing pipeline

3.4 | Exploratory Data Analysis

Create standard visualisation for the processed data to observe some patterns.

```
mpl.pyplot.figure(figsize = (40, 40))
sns.heatmap(train_data.corr(), annot = True, square = True, cmap='RdBu');
```

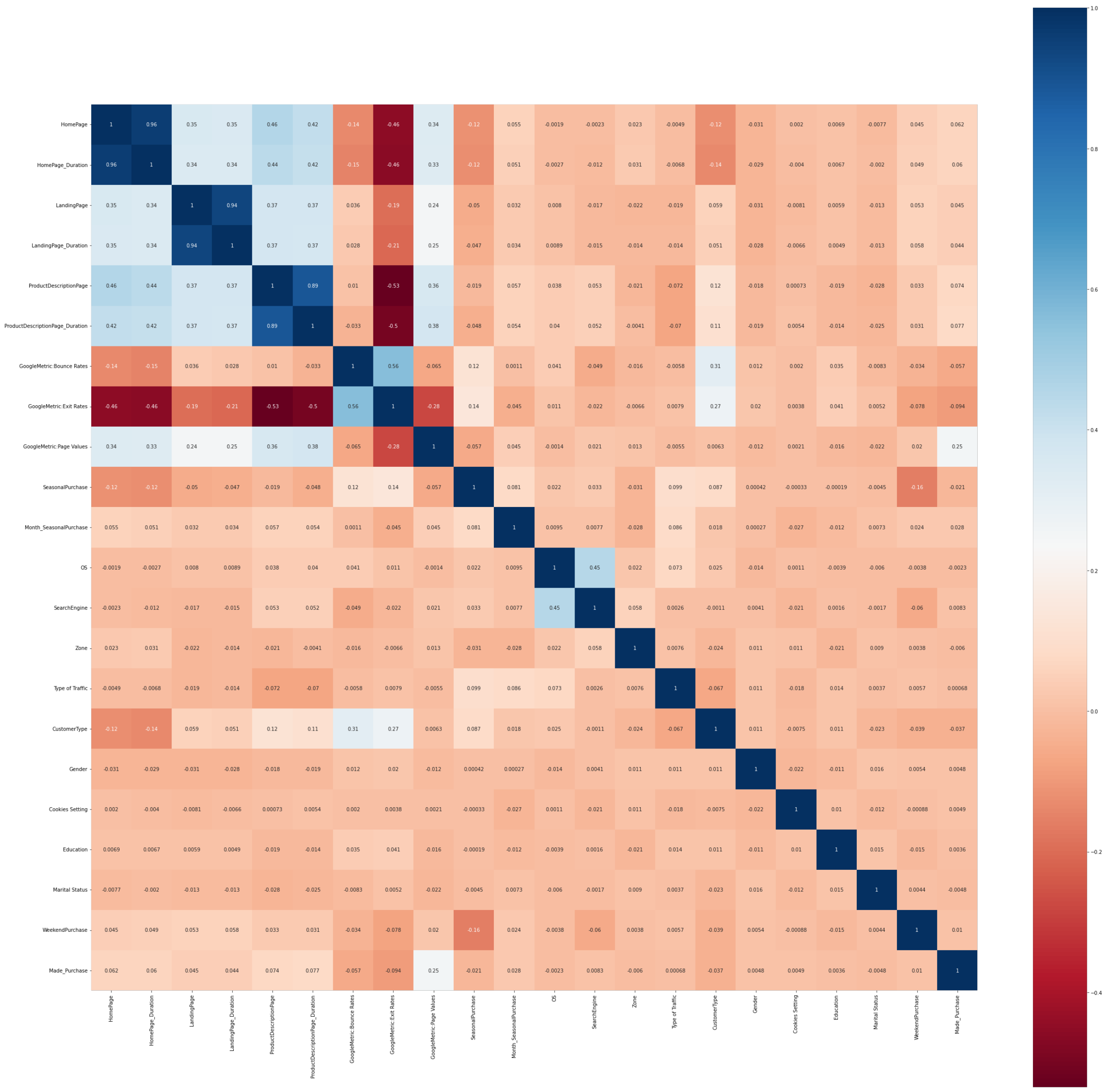
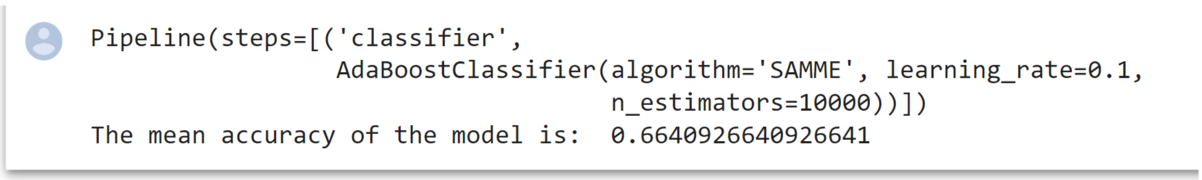


Figure 3.3: Heatmap

3.5 | Model Selection

Explore all the candidate models for the problem and then do a rough evaluation for each one.

```
from sklearn.ensemble import AdaBoostClassifier
pipeline_adaboost = Pipeline(
                        [
                            ('adaboost_classifier', AdaBoostClassifier())
                        ]
                      )
```



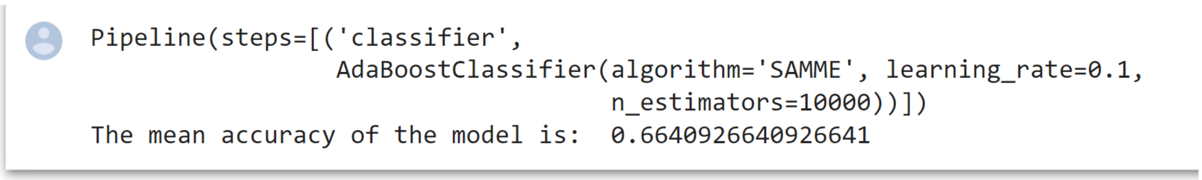
```
Pipeline(steps=[('classifier',
                 AdaBoostClassifier(algorithm='SAMME', learning_rate=0.1,
                                    n_estimators=10000))])
The mean accuracy of the model is:  0.6640926640926641
```

Figure 3.4: Model Selection

3.6 | Training

Train all the candidate models on the training data set.

```
# creating a pipeline object
pipe = Pipeline([('classifier', AdaBoostClassifier())])

# creating a dictionary with candidate learning algorithms and their
↪  hyperparameters
grid_parameters = [
                    {
                        'classifier': [AdaBoostClassifier()],
#                         'classifier__estimator': [GaussianNB(),
↪  DecisionTreeClassifier(max_depth = 1)],
                        'classifier__algorithm': ['SAMME', 'SAMME.R'],
                        'classifier__n_estimators': [1000, 4000, 6000, 10000],
                        'classifier__learning_rate': [0.01, 0.05, 0.1, 0.5]
                    }
                ]

# dict_keys(['memory', 'steps', 'verbose', 'classifier',
#              'classifier__algorithm', 'classifier__base_estimator',
#              'classifier__learning_rate', 'classifier__n_estimators',
↪  'classifier__random_state'])

grid_search = GridSearchCV(pipe, grid_parameters, cv = 4, verbose = 0, n_jobs =
↪  -1)
best_model = grid_search.fit(X_train, y_train)

print(best_model.best_estimator_)
print("The mean accuracy of the model is: ", best_model.score(X_test, y_test))

# AdaBoostClassifier(algorithm='SAMME',
#                    base_estimator=DecisionTreeClassifier(max_depth=1),
#                    learning_rate=0.05, n_estimators=6000)
# Accuracy: 0.7192498621070049
# F1: 0.6563133018230924
```



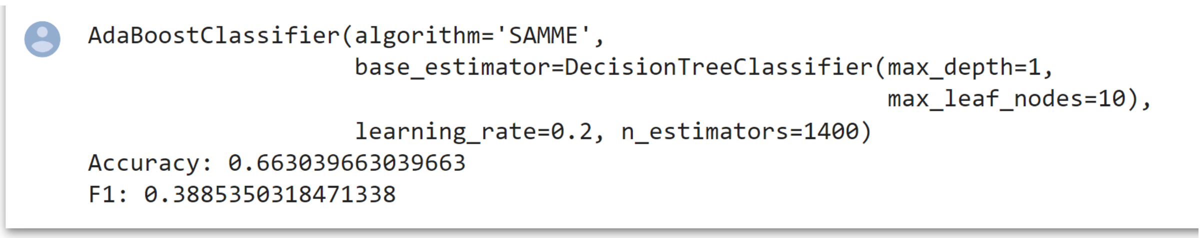
```
Pipeline(steps=[('classifier',
                 AdaBoostClassifier(algorithm='SAMME', learning_rate=0.1,
                                    n_estimators=10000))])
The mean accuracy of the model is:  0.6640926640926641
```

Figure 3.5: Training

3.7 | Evaluation

Calculate the evaluation metrics for all the candidate models.

```
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
best_accuracy = 0.0
best_classifier = 0
best_pipeline = ''
```



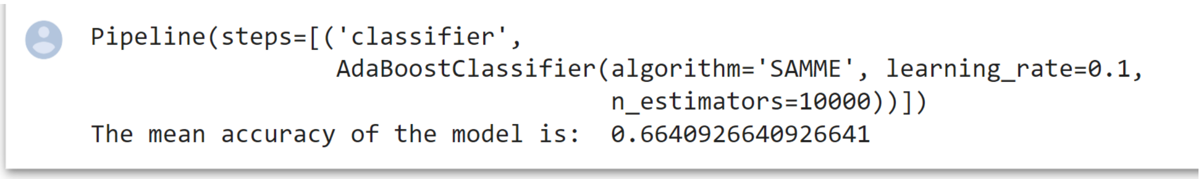
```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         max_leaf_nodes=10),
                   learning_rate=0.2, n_estimators=1400)
Accuracy: 0.663039663039663
F1: 0.3885350318471338
```

Figure 3.6: Evaluation Metrics

3.8 | Hyperparameter Tuning

Tune the hyperparameters of all the candidate algorithms and re-evaluate the metrics.

```
# creating a pipeline object
pipe = Pipeline([('classifier', RandomForestClassifier())])

# creating a dictionary with candidate learning algorithms and their
↪  hyperparameters
grid_parameters = [
                    {
                        'classifier': [LogisticRegression()],
                        'classifier__max_iter': [10000, 100000],
                        'classifier__penalty': ['l1', 'l2'],
                        'classifier__C': np.logspace(0, 4, 10),
                        'classifier__solver': ['newton-cg', 'saga', 'sag',
                        ↪  'liblinear']
                    },
                    {
                        'classifier': [RandomForestClassifier()],
                        'classifier__n_estimators': [10, 100, 1000],
                        'classifier__min_samples_leaf': [2, 5, 10, 15, 100],
                        'classifier__max_leaf_nodes': [2, 5, 10, 20],
                        'classifier__max_depth': [None, 1, 2, 5, 8, 15, 25, 30]

                    },
                    {
                        'classifier': [AdaBoostClassifier()],
                        'classifier__n_estimators': [100, 1000],
                        'classifier__learning_rate': [0.001, 0.01],
                        'classifier__random_state': [1127]
                    },
                    {
                        'classifier': [VotingClassifier()],
                        'classifier__voting': ['hard', 'soft']
                    },
                    {
                        'classifier': [MLPClassifier()],
                        'classifier__hidden_layer_sizes': [(100, ), (1000,)],
                        'classifier__activation': ['identity', 'logistic', 'tanh',
                         ↪   'relu'],
                        'classifier__solver': ['lbfgs', 'sgd', 'adam'],
                        'classifier__learning_rate': ['constant', 'invscaling',
                         ↪   'adaptive'],
                        'classifier__max_iter': [200, 1000, 10000]
                    },
                ]

grid_search = GridSearchCV(pipe, grid_parameters, cv = 4, verbose = 0, n_jobs =
↪   -1)
best_model = grid_search.fit(X_train, y_train)

print(best_model.best_estimator_)
print("The mean accuracy of the model is: ", best_model.score(X_test, y_test))
```



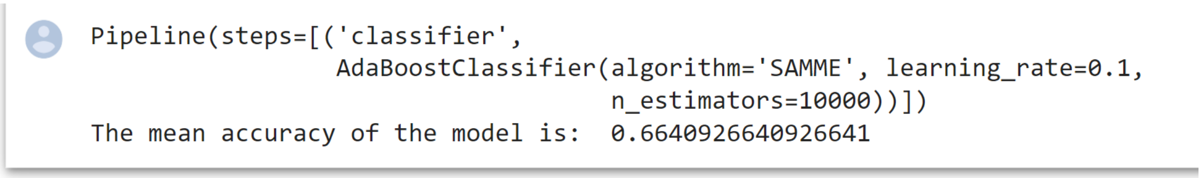
```
Pipeline(steps=[('classifier',
                 AdaBoostClassifier(algorithm='SAMME', learning_rate=0.1,
                                    n_estimators=10000))])
The mean accuracy of the model is:  0.6640926640926641
```

Figure 3.7: Tuned Hyper-parameters of the model

3.9 | Prediction

Select the best performing model based on the defined evaluation metric and then us it to predict the test data set.

```
sub = pd.DataFrame(clf_ada.predict(test_data), columns=['Made_Purchase'])
sub.index.name = 'id'
sub.to_csv("submission.csv", encoding='UTF-8')
output = pd.read_csv("submission.csv")
```



```
Pipeline(steps=[('classifier',
                 AdaBoostClassifier(algorithm='SAMME', learning_rate=0.1,
                                    n_estimators=10000))])
The mean accuracy of the model is:  0.6640926640926641
```

Figure 3.8: Prediction

4 | Conclusion

We have successfully implemented a binary classifier for a real life problem by following standard Machine Learning workflow. We learned about the importance of pre-processing, functioning of different classifiers, its parameters and hyper-parameters.

4.1 | Final Metrics

Accuracy	71.92%
F1	0.6563

5 | References

[1] Sabina-Cristiana Necula. Exploring the Impact of Time Spent Reading Product Information on E-Commerce Websites: A Machine Learning Approach to Analyze Consumer Behavior. *MDPI - Behavioral Sciences*, 2023.

[2] Balaji SPK and Swarnim Soni. E-commerce Shopper's Behaviour Understanding — Kaggle, 2022.