Machine Learning Project
(R4CO3012P)

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

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.

Evaluation

The evaluation metric for this competition is Mean F1-Score (https://en.wikipedia.org/wiki/F-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.

Submission Format

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

$$\begin{array}{c|c} `id` & `Made\_Purchase` \\ \hline 1 & False \end{array}$$

Submitted by

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

0.1 Standard Library Imports

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

0.2 Importing Libraries

In [2]:
```
import xgboost as xgb, scipy as sp, matplotlib as mpl, seaborn as sns

from imblearn import under_sampling, over_sampling

%matplotlib inline
```

1. Get the Data

1.1 Importing Dataset

In [3]:

```
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
```

Out[3]:

```
((14731, 22), (6599, 21))
```

2. Preprocessing

2.2 Data Imputation

In [4]:

```
train_data.isnull().sum()
```

Out[4]:

```
HomePage                             153
HomePage_Duration                    150
LandingPage                          153
LandingPage_Duration                 135
ProductDescriptionPage               123
ProductDescriptionPage_Duration      167
GoogleMetric:Bounce Rates            151
GoogleMetric:Exit Rates              129
GoogleMetric:Page Values             132
SeasonalPurchase                     150
Month_SeasonalPurchase               144
OS                                   134
SearchEngine                         122
Zone                                 117
Type of Traffic                      143
CustomerType                         144
Gender                               145
Cookies Setting                      144
Education                            136
Marital Status                       130
WeekendPurchase                      121
Made_Purchase                          0
dtype: int64
```

2.2.1 Preprocessing Pipeline

In [5]:

```
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, OrdinalE
ncoder
```

In [6]:

```
numerical_features = train_data.select_dtypes(include = ['int64', 'float64']).columns.tolist()
categorical_features = train_data.select_dtypes(include = ['object']).columns.tolist()

num_estimators = [
    ('simple_imputer', SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')),
    ('standard_scaler', StandardScaler()),
    ('quantile_transformer', QuantileTransformer())
]
num_pipeline = Pipeline(steps = num_estimators)

cat_estimators = [
    ('one_hot_encoder', OneHotEncoder())
]
cat_pipeline = Pipeline(steps = cat_estimators)


preprocessing_pipe = ColumnTransformer([
    ('cat_pipeline', cat_pipeline, categorical_features),
    ('num_pipeline', num_pipeline, numerical_features)

])
```

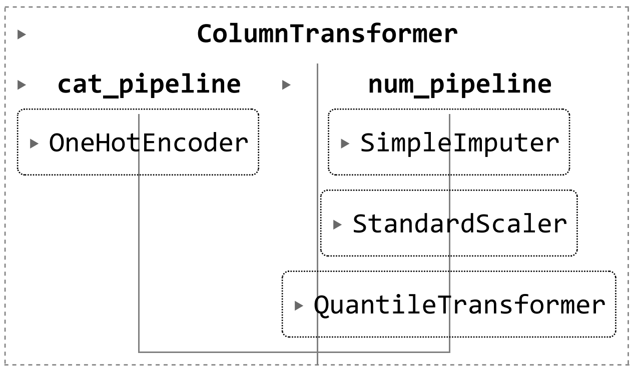
In [7]:

```
from sklearn import set_config

# displays HTML representation in a jupyter context
set_config(display = 'diagram')

preprocessing_pipe
```

Out[7]:



2.2.1 Data Imputation on `train_data`

In [8]:

```
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])

# ohe = OneHotEncoder(handle_unknown = 'ignore')
# enc_train_data = ohe.fit_transform(train_data[cat])
# enc_train_data_df = pd.DataFrame(enc_train_data.toarray(), columns = oe.get_feature_names_out(cat))
# train_data = pd.concat([train_data, enc_train_data_df], axis = 1)
# train_data.drop(cat, axis = 1, inplace = True)

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])
# train_data.head()
```

2.2.1 (Alternative) Data Imputation on `train_data`

In [9]:

```
# X = preprocessing_pipe.fit_transform(X)
# X = pd.DataFrame(X)
# X.head()
```

2.2.2 Data Imputation on `test_data`

In [10]:

```
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')
# test_data[num] = si_mean.fit_transform(test_data[num])

# si_mode = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
# test_data[cat] = si_mode.fit_transform(test_data[cat])

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])

# ohe = OneHotEncoder(handle_unknown = 'ignore')
# enc_test_data = ohe.fit_transform(test_data[cat])
# enc_test_data_df = pd.DataFrame(enc_test_data.toarray(), columns = oe.get_feature_names_out(cat))
# test_data = pd.concat([test_data, enc_test_data_df], axis = 1)
# test_data.drop(cat, axis = 1, inplace = True)

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])
# test_data.head()
```

2.2.2 (Alternative) Data Imputation on `test_data`

In [11]:

```
# test_data = preprocessing_pipe.fit_transform(test_data)
# test_data = pd.DataFrame(test_data)
# test_data.head()
```

Balancing using `imblearn`

In [12]:

```
# from imblearn.over_sampling import SMOTE

# X, y = SMOTE().fit_resample(X, y)
# X.shape, y.shape
```

Outlier Adjustment

In [13]:

```
from scipy import stats

train_data = train_data[(np.abs(stats.zscore(train_data[num])) < 3).all(axis=1)]
# test_data = test_data[(np.abs(stats.zscore(test_data[num])) < 3).all(axis=1)]
```

In [14]:

```
train_data.shape, test_data.shape
```

Out[14]:

```
((14540, 22), (6599, 21))
```

Dropping Duplicates

In [15]:

```
print(train_data.duplicated().sum())
train_data = train_data.drop_duplicates()
```

```
299
```

In [16]:

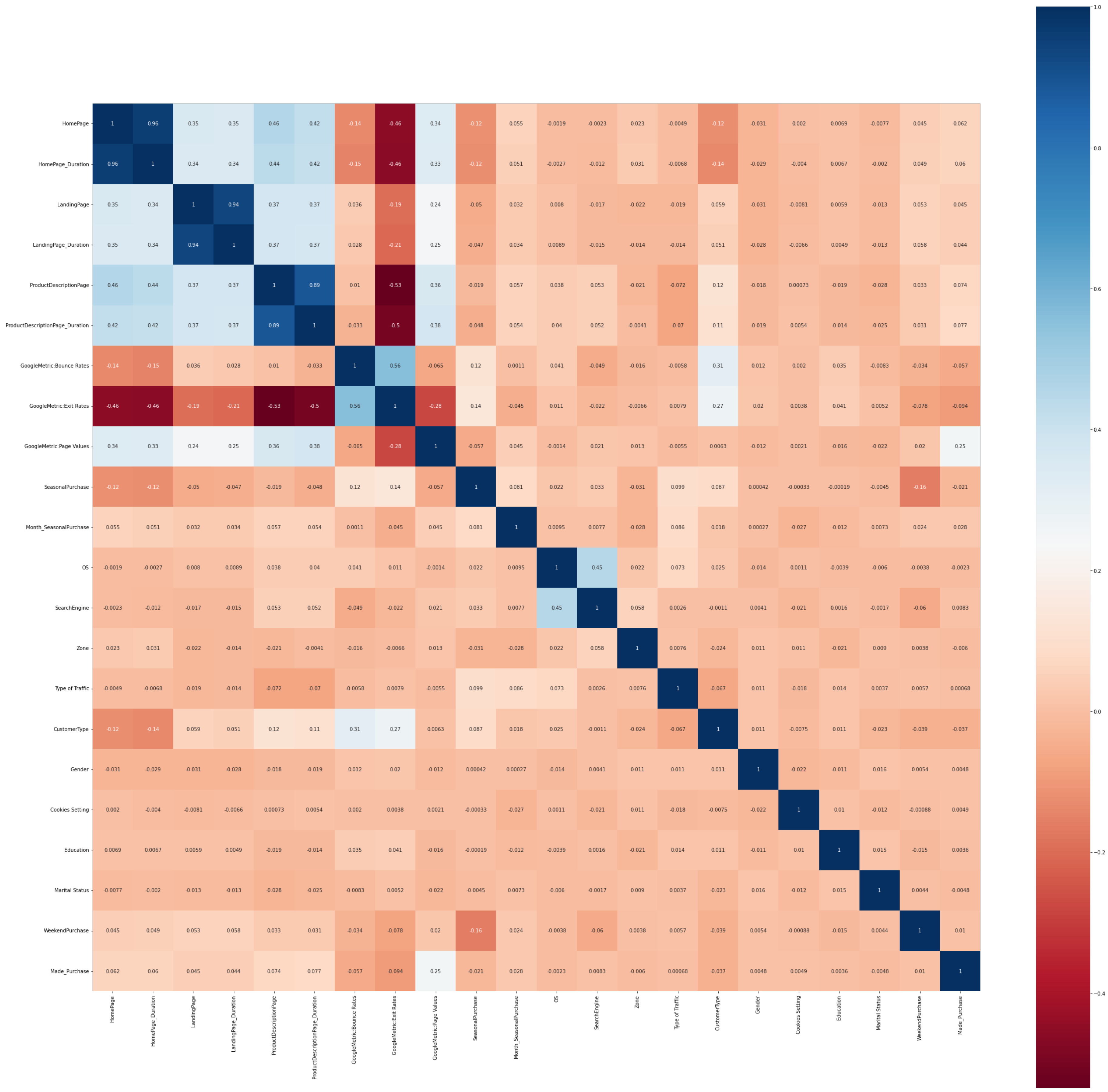
```
train_data.shape, test_data.shape
```

Out[16]:

```
((14241, 22), (6599, 21))
```

In [17]:

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



2.3 Splitting the Data for training

2.1 Separating features and labels

In [18]:

```
y, X = train_data.pop('Made_Purchase'), train_data

X.shape, y.shape
```

Out[18]:

```
((14241, 21), (14241,))
```

In [19]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1127)
```

In [20]:
```
X_train.describe().T
```

Out[20]:

	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

3. Baseline Model

In [21]:
```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier(strategy = 'most_frequent')
dummy_clf.fit(X, y)
DummyClassifier(strategy = 'most_frequent')
dummy_clf.predict(X)
```

Out[21]:
```
array([False, False, False, ..., False, False, False])
```

4. Candidate Algorithms

In [22]:
```
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score

best_accuracy = 0.0
best_classifier = 0
best_pipeline = ''
```

4.1 Logistic Regression

In [23]:

```
from sklearn.linear_model import LogisticRegression

pipeline_lr = Pipeline(
                        [
                            ('lr_classifier', LogisticRegression(max_iter = 100000))
                        ]
                      )
```

In [24]:

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

# 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']
                    }
                ]




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))


# LogisticRegression(C=2.7825594022071245, max_iter=10000, penalty='l1',
#                    solver='saga')
# Accuracy: 0.6628222523744912
# F1: 0.28694404591104733

# LogisticRegression(C=2.7825594022071245, max_iter=10000, penalty='l1',
#                    solver='saga')
# Accuracy: 0.6441581519324745
# F1: 0.24075829383886257

# LogisticRegression(C=2.7825594022071245, max_iter=10000, penalty='l1',
#                    solver='saga')
# Accuracy: 0.6612846612846612
# F1: 0.4197233914612147
```

```
Pipeline(steps=[('classifier',
                 LogisticRegression(max_iter=10000, penalty='l1',
                                    solver='saga'))])
The mean accuracy of the model is:  0.6612846612846612
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
160 fits failed out of a total of 640.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
80 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 680, in _fi
t_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py", line 449, in _check_s
olver
    % (solver, penalty)
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
80 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 680, in _fi
t_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py", line 449, in _check_s
olver
    % (solver, penalty)
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_search.py:972: UserWarning: One or more
of the test scores are non-finite: [      nan 0.6640625       nan 0.6640625 0.6640625 0.6640625 0.664062
5
 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625 0.6640625
 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625
 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625
 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan
 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625
       nan 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625       nan
 0.6640625       nan 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625
       nan 0.6640625       nan 0.6640625 0.6640625 0.6640625 0.6640625
 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625 0.6640625
 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625
 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625
 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan
 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625
       nan 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625       nan
 0.6640625       nan 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625
       nan 0.6640625       nan 0.6640625 0.6640625 0.6640625 0.6640625
 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625 0.6640625
 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625 0.6640625
 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan 0.6640625
 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625       nan
 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625       nan 0.6640625
```

```
    nan 0.6640625 0.6640625 0.6640625 0.6640625 0.6640625]
  category=UserWarning,
```

In [25]:

```
clf_lr = LogisticRegression(C = 2.7825594022071245, max_iter = 10000, penalty = 'l1', solver = 'saga')
clf_lr.fit(X_train, y_train)
y_pred = clf_lr.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_lr, '\nAccuracy:', clf_lr.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))
```

```
LogisticRegression(C=2.7825594022071245, max_iter=10000, penalty='l1',
                   solver='saga')
Accuracy: 0.6612846612846612
F1: 0.4197233914612147
```

In [26]:

```
score_ = cross_val_score(clf_lr, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)
```

```
Cross Validation Score: 0.6640642761425073
```

4.2 Decision Tree Classifier

In [27]:

```
from sklearn.tree import DecisionTreeClassifier

pipeline_dt = Pipeline(
                        [
                            ('dt_classifier', DecisionTreeClassifier())
                        ]
                      )
```

In [28]:

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

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [DecisionTreeClassifier()],
                        'classifier__max_depth': [1, 2, 5, 10, 100]
                    }
                ]



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))



# DecisionTreeClassifier(max_depth=1, max_leaf_nodes=10)
# Accuracy: 0.6609336609336609
# F1: 0.4180722891566265
```

```
Pipeline(steps=[('classifier', DecisionTreeClassifier(max_depth=1))])
The mean accuracy of the model is:  0.6609336609336609
```

In [29]:

```python
clf_dt = DecisionTreeClassifier(max_depth = 1, max_leaf_nodes = 10)
clf_dt.fit(X_train, y_train)
y_pred = clf_dt.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_dt, '\nAccuracy:', clf_dt.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))



# DecisionTreeClassifier(max_depth=1)
# Accuracy: 0.6486006219458018
# F1: 0.3500410846343468

# DecisionTreeClassifier(max_depth=1, max_leaf_nodes=10)
# Accuracy: 0.6609336609336609
# F1: 0.4180722891566265
```

```
DecisionTreeClassifier(max_depth=1, max_leaf_nodes=10)
Accuracy: 0.6609336609336609
F1: 0.4180722891566265
```

In [30]:

```python
score_ = cross_val_score(clf_dt, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)

# Cross Validation Score: 0.6619583968701385
```

```
Cross Validation Score: 0.6619583968701385
```

4.3 Random Forest Classifier

In [31]:

```python
from sklearn.ensemble import RandomForestClassifier

pipeline_rf = Pipeline(
                        [
                            ('rf_classifier', RandomForestClassifier(criterion='gini', n_estimators=100))
                        ]
                      )
```

In [32]:

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

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        '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]
                    }
                ]



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))


# RandomForestClassifier(max_depth=30, max_leaf_nodes=20, min_samples_leaf=100,
#                        n_estimators=1000)
# Accuracy: 0.6784260515603799
# F1: 0.35422343324250677
```

```
Pipeline(steps=[('classifier',
                 RandomForestClassifier(max_depth=15, max_leaf_nodes=10,
                                        min_samples_leaf=15))])
The mean accuracy of the model is:  0.6577746577746578
```

In [33]:

```
clf_rf = RandomForestClassifier(max_depth = 30,
                                max_leaf_nodes = 20,
                                min_samples_leaf = 100,
                                n_estimators = 1000)
clf_rf.fit(X_train, y_train)
y_pred = clf_rf.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_rf, '\nAccuracy:', clf_rf.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))

# RandomForestClassifier(max_depth=30, max_leaf_nodes=20, min_samples_leaf=100,
#                        n_estimators=1000)
# Accuracy: 0.6534873389604621
# F1: 0.28702010968921393

# RandomForestClassifier(max_depth=30, max_leaf_nodes=20, min_samples_leaf=100,
#                        n_estimators=1000)
# Accuracy: 0.6595296595296596
# F1: 0.3618421052631579
```

```
RandomForestClassifier(max_depth=30, max_leaf_nodes=20, min_samples_leaf=100,
                       n_estimators=1000)
Accuracy: 0.6567216567216567
F1: 0.34973404255319146
```

In [34]:

```
score_ = cross_val_score(clf_rf, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)

# Cross Validation Score: 0.6655923048464668
# Cross Validation Score: 0.666170694515041
```

```
Cross Validation Score: 0.66581966329344
```

4.4 ADAboost Classifier

In [35]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier

pipeline_adaboost = Pipeline(
                        [
                            ('adaboost_classifier', AdaBoostClassifier())
                        ]
                      )
```

In [36]:

```
# 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',
#                    estimators = clf_dt,
#                    learning_rate = 0.05,
#                    n_estimators = 4000)
# Accuracy: 0.6811397557666214
# F1: 0.4035532994923857

# AdaBoostClassifier(algorithm='SAMME',
#                    base_estimator=DecisionTreeClassifier(max_depth=1),
#                    learning_rate=0.05, n_estimators=6000)
# Accuracy: 0.6534873389604621
# F1: 0.32055749128919864

# 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
```

In [37]:

```
clf_ada = AdaBoostClassifier(algorithm = 'SAMME',
                             base_estimator = clf_dt,
                             learning_rate = 0.2,
                             n_estimators = 1400)
clf_ada.fit(X_train, y_train)
y_pred = clf_ada.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_ada, '\nAccuracy:', clf_ada.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))


# AdaBoostClassifier(algorithm='SAMME',
#                    base_estimator=DecisionTreeClassifier(max_depth=1),
#                    learning_rate=0.05, n_estimators=8000)
# Accuracy: 0.6534873389604621
# F1: 0.3193717277486911

# AdaBoostClassifier(algorithm='SAMME',
#                    base_estimator=DecisionTreeClassifier(max_depth=1),
#                    learning_rate=0.005, n_estimators=8000)
# Accuracy: 0.6521545979564638
# F1: 0.30646589902568644

# AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(max_depth=1,
#                    max_leaf_nodes=10), learning_rate=0.05, n_estimators=8000)
# Accuracy: 0.6633906633906634
# F1: 0.3824855119124276
```

```
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
```

In [38]:

```
clf_ada = AdaBoostClassifier(algorithm = 'SAMME',
                             base_estimator = clf_dt,
                             learning_rate = 0.9,
                             n_estimators = 4000)
clf_ada.fit(X_train, y_train)
y_pred = clf_ada.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_ada, '\nAccuracy:', clf_ada.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))

# AdaBoostClassifier(algorithm='SAMME',
#                    base_estimator=DecisionTreeClassifier(max_depth=1, max_leaf_nodes=10),
#                    learning_rate=0.09, n_estimators=4000)
# Accuracy: 0.6633906633906634
# F1: 0.3832797427652734
```

```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=DecisionTreeClassifier(max_depth=1,
                                                         max_leaf_nodes=10),
                   learning_rate=0.9, n_estimators=4000)
Accuracy: 0.663039663039663
F1: 0.415347137637028
```

In [39]:

```
score_ = cross_val_score(clf_ada, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)

# Cross Validation Score: 0.6665930447650759
# Cross Validation Score: 0.6644157694499638
```

```
Cross Validation Score: 0.6629240792939328
```

4.5 VotingClassifier

In [40]:

```
from sklearn.ensemble import VotingClassifier

pipeline_vc = VotingClassifier(
                        [
                            ('vc_classifier', VotingClassifier(
                                                                    estimators = [
                                                                                    ('ada', clf_ada),
                                                                                    ('gnb', GaussianNB())
                                                                                ]
                                                                    )
                            )
                        ]
                      )
```

In [41]:

```
# creating a pipeline object
pipe = Pipeline([('classifier', VotingClassifier(estimators = [
                                                                ('ada', AdaBoostClassifier()),
                                                                ('gnb', GaussianNB())
                                                            ]))])

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [VotingClassifier(estimators = [
                                                                ('ada', AdaBoostClassifier()),
                                                                ('gnb', GaussianNB())
                                                            ])],
                        'classifier__voting': ['hard', 'soft']
                    }
                ]


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))


# VotingClassifier(estimators=[('ada', AdaBoostClassifier()),
#                              ('gnb', GaussianNB())])
# Accuracy 0.664179104477612
# F1 0.32653061224489793
```

```
Pipeline(steps=[('classifier',
                 VotingClassifier(estimators=[('ada', AdaBoostClassifier()),
                                              ('gnb', GaussianNB())]))])
The mean accuracy of the model is:  0.6616356616356617
```

In [42]:

```
clf_vc = VotingClassifier(estimators=[('ada', clf_ada), ('gnb', GaussianNB())])
clf_vc.fit(X_train, y_train)
y_pred = clf_vc.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_vc, '\nAccuracy', clf_vc.score(X_test, y_test), '\nF1', f1_score(y_test, y_pred))
```

```
VotingClassifier(estimators=[('ada',
                              AdaBoostClassifier(algorithm='SAMME',
                                                 base_estimator=DecisionTreeClassifier(max_depth=1,
                                                                                       max_leaf_nodes=1
0),
                                                 learning_rate=0.9,
                                                 n_estimators=4000)),
                             ('gnb', GaussianNB())])
Accuracy 0.663039663039663
F1 0.415347137637028
```

In [43]:
```
score_ = cross_val_score(clf_vc, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)
```

```
Cross Validation Score: 0.6629240792939328
```

4.6 SVM Classifier

In [44]:
```
from sklearn.svm import SVC

pipeline_svm = Pipeline(
                        [
                            ('svm_classifier', SVC(gamma = 'auto'))
                        ]
                      )



# kernel{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}
```

In [45]:
```
# creating a pipeline object
pipe = Pipeline([('classifier', SVC())])

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [SVC()],
                        'classifier__kernel': ['linear', 'poly', 'rbf', 'sigmoid']
                    }
                ]



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))

# SVC()
# Accuracy 0.6648575305291723
# F1 0.31955922865013775
```

```
Pipeline(steps=[('classifier', SVC(kernel='linear'))])
The mean accuracy of the model is:  0.6612846612846612
```

In [46]:
```
clf_svc = SVC()
clf_svc.fit(X_train, y_train)
y_pred = clf_svc.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_svc, '\nAccuracy', clf_svc.score(X_test, y_test), '\nF1', f1_score(y_test, y_pred))
```

```
SVC()
Accuracy 0.6612846612846612
F1 0.4197233914612147
```

In [47]:

```
score_ = cross_val_score(clf_svc, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)
```

```
Cross Validation Score: 0.6640642761425073
```

4.7 KNN Classifier

In [48]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier

pipeline_knn = Pipeline(
                        [
                            ('knn_classifier',
                             BaggingClassifier(KNeighborsClassifier(),
                                               max_samples = 0.5,
                                               max_features = 0.5))
                        ]
                      )
```

In [49]:

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

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [BaggingClassifier()],
                        'classifier__warm_start': [True, False]
                    }
                ]


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))

# BaggingClassifier(warm_start=True)
# Accuracy 0.495251017639077 36
# F1 0.2927756653992396
```

```
Pipeline(steps=[('classifier', BaggingClassifier(warm_start=True))])
The mean accuracy of the model is:  0.4984204984204984
```

In [50]:

```
clf_knn = BaggingClassifier()
BaggingClassifier(KNeighborsClassifier(), warm_start = True, max_samples = 5, max_features = 5)
clf_knn.fit(X_train, y_train)
y_pred = clf_knn.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_knn, '\nAccuracy', clf_knn.score(X_test, y_test), '\nF1', f1_score(y_test, y_pred))
```

```
BaggingClassifier()
Accuracy 0.4871884871884872
F1 0.2869692532942899
```

In [51]:

```
score_ = cross_val_score(clf_knn, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)
```

```
Cross Validation Score: 0.5056188099748934
```

4.8 Multi-Layer Perceptron

In [52]:

```
from sklearn.neural_network import MLPClassifier


pipeline_mlp = Pipeline(
                        [
                            ('mlp_classifier', MLPClassifier()
                            )
                        ]
                      )
```

In [53]:

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

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [MLPClassifier()],
                        'classifier__hidden_layer_sizes': [(100, ), (400, ), (600, )],
                        'classifier__activation': ['tanh', 'relu'],
                        'classifier__solver': ['sgd', 'adam'],
                        'classifier__learning_rate': ['constant', 'invscaling', 'adaptive'],
                        'classifier__max_iter': [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))



# MLPClassifier(activation='tanh', hidden_layer_sizes=(400,),
#               learning_rate='adaptive', max_iter=1000, solver='sgd')
# Accuracy: 0.666214382632293
# F1: 0.3031161473087819
```

```
Pipeline(steps=[('classifier',
                 MLPClassifier(activation='tanh', hidden_layer_sizes=(600,),
                               learning_rate='adaptive', max_iter=1000))])
The mean accuracy of the model is:  0.6581256581256582
```

In [54]:

```
clf_mlp = MLPClassifier(hidden_layer_sizes = (40,30,20,10),
                        max_iter = 1000,
                        solver = 'sgd',
                        activation = 'tanh',
                        learning_rate = 'adaptive')
clf_mlp.fit(X_train, y_train)
y_pred = clf_mlp.predict(X_test)
y_pred = y_pred.astype(bool)
print(clf_mlp, '\nAccuracy:', clf_mlp.score(X_test, y_test), '\nF1:', f1_score(y_test, y_pred))
```

```
MLPClassifier(activation='tanh', hidden_layer_sizes=(40, 30, 20, 10),
              learning_rate='adaptive', max_iter=1000, solver='sgd')
Accuracy: 0.6588276588276588
F1: 0.3761232349165597
```

In [55]:

```
score_ = cross_val_score(clf_mlp, X_train, y_train, cv = 10).mean()
print('Cross Validation Score:', score_)
```

```
Cross Validation Score: 0.6630987477473315
```

4.9 HistGradientBoostingClassifier

In [56]:

```
from sklearn.ensemble import HistGradientBoostingClassifier

pipeline_hgb = Pipeline(
                        [
                            ('hgb_classifier', HistGradientBoostingClassifier())
                        ]
                      )

# loss='log_loss', *, learning_rate = 0.01, max_iter=100, max_leaf_nodes=31, max_depth=None, min_samples_leaf=
20, l2_regularization=0.0, max_bins=255, categorical_features=None, monotonic_cst=None, interaction_cst=None,
warm_start=False, early_stopping='auto', scoring='loss', validation_fraction=0.1, n_iter_no_change=10, tol=1e-
07, verbose=0, random_state=None, class_weight=None
```

In [57]:

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

# creating a dictionary with candidate learning algorithms and their hyperparameters
grid_parameters = [
                    {
                        'classifier': [HistGradientBoostingClassifier()],
                        'classifier__loss': ['log_loss', 'auto', 'binary_crossentropy', 'categorical_crossen
tropy'],
                        'classifier__learning_rate': np.logspace(0, 4, 10),
                        'classifier__max_iter': [1000, 10000],
                        'classifier__max_depth': [None, 1, 2, 5, 8, 15, 25, 30],
                        'classifier__min_samples_leaf': [None, 2, 5, 10, 15, 100]
                    }
                ]

# loss='log_loss',
# learning_rate = 0.01,
# max_iter=100,
# max_leaf_nodes=31,
# max_depth=None,
# min_samples_leaf=20,
# l2_regularization=0.0,
# max_bins=255,
# categorical_features=None,
# monotonic_cst=None,
# interaction_cst=None,
# warm_start=False,
# early_stopping='auto',
# scoring='loss',
# validation_fraction=0.1,
# n_iter_no_change=10,
# tol=1e-07,
# verbose=0,
# class_weight=None

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))
```

```
/opt/conda/lib/python3.7/site-packages/joblib/externals/loky/process_executor.py:691: UserWarning: A wor
ker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout
or by a memory leak.
  "timeout or by a memory leak.", UserWarning


Pipeline(steps=[('classifier',
                 HistGradientBoostingClassifier(learning_rate=2.7825594022071245,
                                                max_depth=1, max_iter=1000,
                                                min_samples_leaf=2))])
The mean accuracy of the model is:  0.6605826605826606
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
8960 fits failed out of a total of 15360.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
3840 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 680, in _fi
t_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/gradient_boostin
g.py", line 251, in fit
    self._validate_parameters()
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/gradient_boostin
g.py", line 85, in _validate_parameters
    self.loss, self.__class__.__name__, ", ".join(self._VALID_LOSSES)
ValueError: Loss log_loss is not supported for HistGradientBoostingClassifier. Accepted losses: binary_c
rossentropy, categorical_crossentropy, auto.

--------------------------------------------------------------------------------
1280 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 680, in _fi
t_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/gradient_boostin
g.py", line 524, in fit
    n_threads=n_threads,
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/grower.py", line
214, in __init__
    min_hessian_to_split,
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/grower.py", line
338, in _validate_parameters
    if min_samples_leaf < 1:
TypeError: '<' not supported between instances of 'NoneType' and 'int'

--------------------------------------------------------------------------------
3840 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 680, in _fi
t_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/gradient_boostin
g.py", line 274, in fit
    sample_weight=sample_weight, n_threads=n_threads
  File "/opt/conda/lib/python3.7/site-packages/sklearn/ensemble/_hist_gradient_boosting/gradient_boostin
g.py", line 1694, in _get_loss
    "'categorical_crossentropy' is not suitable for "
ValueError: 'categorical_crossentropy' is not suitable for a binary classification problem. Please use '
auto' or 'binary_crossentropy' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_search.py:972: UserWarning: One or more
```

```
of the test scores are non-finite: [nan nan nan ... nan nan nan]
  category=UserWarning,
```

5. Training

5.1 Consolidating Pipelines

In [58]:

```
# pipelines = [pipeline_lr, pipeline_dt, pipeline_rf, pipeline_adaboost, pipeline_mlp]

# pipes = {
#             0: 'Logistic Regression',
#             1: 'Decision Tree',
#             2: 'Random Forest',
#             3: 'ADAboost',
#             4: 'Multi-Layer Perceptron'
# }
```

5.2 Fitting Models

In [59]:

```
# for pipe in pipelines:
#     pipe.fit(X_train, y_train)
```

5.3 Scoring of Models

5.3.1 Accuracy

In [60]:

```
# for _, classifier in enumerate(pipelines):
#     print('{} test accuracy: {}'.format(pipes[_], classifier.score(X_test, y_test)))


# # Logistic Regression test accuracy: 0.6628222523744912
# # Decision Tree test accuracy: 0.4497964721845319
# # Random Forest test accuracy: 0.5210312075983717
# # ADAboost test accuracy: 0.6723202170963365
# # Multi-Layer Perceptron test accuracy: 0.6648575305291723
```

5.3.2 F1

In [61]:

```
# from sklearn.metrics import f1_score


# for _, classifier in enumerate(pipelines):
#     y_pred = classifier.predict(X_test)
#     y_pred = y_pred.astype(bool)
#     print('{} f1-score: {}'.format(pipes[_], f1_score(y_test, y_pred)))


# # Logistic Regression f1-score: 0.28694404591104733
# # Decision Tree f1-score: 0.2453531598513011
# # Random Forest f1-score: 0.28771228771228774
# # ADAboost f1-score: 0.3893805309734514
```

6. Hyperparameter Tuning

6.1 Importing GridSearchCV

In [62]:

```
# from sklearn.model_selection import GridSearchCV
```

6.2 Creating a pipeline fot GridSearchCV

In [63]:

```
# # 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))
```

6.3 Creating GridSearchCV

In [64]:

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

6.4 Fitting best parameters

In [65]:

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

# Pipeline(steps=[('classifier',
#                  RandomForestClassifier(max_depth=5, max_leaf_nodes=10,
#                  min_samples_leaf=15))])
# The mean accuracy of the model is:  0.6709633649932157

# Pipeline(steps=[('classifier',
#                  AdaBoostClassifier(learning_rate=0.01, n_estimators=1000,
#                  random_state=1127))])
# The mean accuracy of the model is:  0.6811397557666214
```

6.5 Calculating scores of best parameters

6.5.1 F1

In [66]:

```
# y_pred = best_model.predict(X_test)
# y_pred = y_pred.astype(bool)
# print('{} f1-score: {}'.format(pipes[_], f1_score(y_test, y_pred)))
# # Random Forest f1-score: 0.3081312410841655
# # ADAboost f1-score: 0.4035532994923857
```

6.5.2 Cross Validation

In [67]:

```
# from sklearn.model_selection import cross_val_score

# score_ = cross_val_score(best_model, X_train, y_train, cv = 3).mean()
# print('{} cross_validation-score: {}'.format(pipes[_], score_))

# # Random Forest cross_validation-score: 0.6551260431050521
```

Submitting as CSV file

In [68]:

```
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")
```

In [69]:

```
# with open('/kaggle/working/submission.csv') as f:
#     ff = f.readlines()
#     print(*ff)
```