Performance evaluation of Decison Tree and Random Forest algorithm using Online Shoppers Purchasing Intention Dataset

This is a school project. The dataset was downloaded from UCI Repository, https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+

Exercise 1

Ex. 1.1: Describe the relation between Random Forests and Decision Trees (for classification).

A Decision tree is built on an entire dataset, using all the features of interest, whereas a Random forest randomly selects observations and specific features to build multiple decision trees from and then select a class recieving the majority votes.

A decision tree combines some decisions, whereas a random forest combines several decision trees.

Ex 1.2: Compare the Random Forest and the Decision Tree classifier in sklearn by discussing the parameters n_estimators, criterion, and max_depth. Explain what the parameters control and why they are applicable to both algorithms or just the one.

```
                   Random Forest     Decison Tree

    n_estimators      Yes               No

    criterion         Yes               Yes

    max_depth         Yes               Yes
```

n_estimators: This is the number of trees in the forest. Default is 100

criterion: This is function used to measure the quality of the split. Default is 'gini' for Gini Impurities

max_depth: This is the maximum depth of the tree.

Ex. 1.3: Compare the two algorithms with respect to their application: Which are immediate advantages and disadvantages of Random Forest over Decision Trees?

Decison Tree

Advantage: Fast and easy to interpret

Disadvantages: It can lead to overfitting on large dataset. Again, it may not give predictive accuracy

Random Forest

Advantage: It uses averaging to improve predictive accuracy and reduce overfitting even on large datasets

Disadvantage :Difficult to interpret

Exercise 2

Ex. 2.1: How many numerical features can we use for predicting whether revenue is true or false.

In [101…

```
import pandas as pd

df_shoppers = pd.read_csv("online_shoppers_intention.csv")
```

In [6]:

```
df_shoppers.head()
```

Out[6]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	P
0	0	0.0	0	0.0	1	
1	0	0.0	0	0.0	2	
2	0	0.0	0	0.0	1	
3	0	0.0	0	0.0	2	
4	0	0.0	0	0.0	10	

10 numerical features can be used for predicting whether revenue is True or False. They includes: Administrative, Administrative_Duration, Informational, Informational_Duration, ProductRelated, ProductRelated_Duration, BounceRates, ExitRates, PageValues and SpecialDay

2.2: Describe class distribution of the dataset

Class distribution is the Revenue which checks whether the visit has been finalized with a transaction. The dataset has a class imbalance of 10422(False): 1908(True), with a ratio of 5.5:1

In [40]:

```
df_shoppers["Revenue"].value_counts()
```

Out[40]:

```
0    10422
1     1908
Name: Revenue, dtype: int64
```

Exercise 3

• Splitting the target attribute from the data, converting it into a form suitable for sklearn classifiers and preparing the numerical attributes as features,
• selecting 30% of the data as test data (choose random seed 42),
• scaling the data such that the features have similar average and standard distribution.

In [155…

```
df_shoppers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
```

```
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

3.1: Convert the categorical variables to numerical variables

In [8]:
```
df_shoppers.Revenue = df_shoppers.Revenue.astype('int')
df_shoppers.Weekend = df_shoppers.Weekend.astype('int')
```

In [9]:
```
df_shoppers.head()
```

Out[9]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	P
0	0	0.0	0	0.0	1	
1	0	0.0	0	0.0	2	
2	0	0.0	0	0.0	1	
3	0	0.0	0	0.0	2	
4	0	0.0	0	0.0	10	

In [156…
```
y = df_shoppers['Revenue'].values
```

In [159…
```
X = df_shoppers.iloc[:,:10].values
```

3.2: selecting 30% of the test data(random state = 42)

In [161…
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_sta
```

In [162…
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[162…
```
((8631, 10), (3699, 10), (8631,), (3699,))
```

3.3: Scaling the data such that the features have similar average and standard distribution

In [163…
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

4.

Use a combination of the classes GridSearchCV and RepeatedStratifiedKFold to setup a cross validation procedure for hyper parameter optimization.

Ex 4.1: Create a cross validation setting in which the data is split into 10 folds, where all experiments are repeated 10 times, and where algorithms are evaluated using balanced accuracy.

In [65]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold

dt_param_grid = {'criterion':['gini','entropy'],'max_depth':[3,5,10]}
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state = 42)
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state =42), dt_param_grid, scor
dt_grid.fit(X_train, y_train)
```

Out[65]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=10, n_splits=10, random_state=42),
             estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 10]},
             scoring='balanced_accuracy')
```

In [66]:
```
dt_grid.best_params_
```

Out[66]:
```
{'criterion': 'gini', 'max_depth': 3}
```

In [67]:
```
dt_grid.best_estimator_
```

Out[67]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [31]:
```
#dt_grid.cv_results_
```

In [ ]:
```
```

In [41]:
```
dt_grid.best_score_
```

Out[41]:
```
0.8970331343075404
```

In [25]:
```
#Randomforest estimator
```

In [18]:
```
from sklearn.ensemble import RandomForestClassifier

rf_param_grid = {'n_estimators':[10,20], 'max_features':['auto', 'sqrt', 'log2'],
                 'criterion':['gini','entropy'],'max_depth':[3,5,10]}
rf_grid = GridSearchCV(RandomForestClassifier(random_state =42),rf_param_grid, scori
rf_grid.fit(X_train, y_train)
```

```
Fitting 100 folds for each of 36 candidates, totalling 3600 fits
```

Out[18]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=10, n_splits=10, random_state=1),
             estimator=RandomForestClassifier(random_state=1),
```

```
                         param_grid={'criterion': ['gini', 'entropy'],
                                     'max_depth': [3, 5, 10],
                                     'max_features': ['auto', 'sqrt', 'log2'],
                                     'n_estimators': [10, 20]},
                         verbose=1)
```

In [19]:

```
rf_grid.best_params_
```

Out[19]:

```
{'criterion': 'entropy',
 'max_depth': 5,
 'max_features': 'auto',
 'n_estimators': 20}
```

In [21]:

```
rf_grid.best_estimator_
```

Out[21]:

```
RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=20,
                       random_state=1)
```

In [43]:

```
rf_grid.best_score_
```

Out[43]:

```
0.8984126888330976
```

In [27]:

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion='gini', random_state=1)

# Train the classifier
clf.fit(X_train, y_train)
```

Out[27]:

```
DecisionTreeClassifier(random_state=1)
```

In [31]:

```
 from sklearn.ensemble import RandomForestClassifier

clfr = RandomForestClassifier( n_estimators = 20, max_depth=2, random_state=1)
clfr.fit(X_train, y_train)
```

Out[31]:

```
RandomForestClassifier(max_depth=2, n_estimators=20, random_state=1)
```

Ex 4.2: Which dataset is used in the grid search cross validation (training data, test data, or full dataset)?

Training data was used in grid search cross validation. Specifically, X_train and y_train were used

Ex 4.3: Explain, what happens to that dataset during the grid search procedure!

Grid-search is used to find the optimal hyperparameters of a model which results in the most accurate predictions. During the grid-search procedure, datasets are been transformed by reducing the number of false positives in order to find the optimal parameters that meet the expectation of algorithm. It can be slow and computational expensive.

Ex 4.4: What is the difference between using RepeatedStratifiedKFold and the default cross validation in GridSearchCV?

For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used. These splitters are instantiated with shuffle=False so the splits will be the same across calls. RepeatedStratifiedKFold can be used to repeat Stratified K-Fold n times with different randomization in each repetition.

Ex 4.5: Explain the purpose that justifies repeating experiments on the same dataset and on different folds.

A single run of the k-fold cross-validation procedure may result in a noisy estimate of model performance as different splits of the data may result in very different results.

Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model by repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs. This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error.

Exercise 5: Evaluation of Classifiers

Use the above cross validation setup to optimize and compare tree based learners. Use

Ex 5.1: Decision Trees with the Gini criterion and test parameters 2 through 14 for max_depth.

In [63]:
```
dt_param_grid = {'criterion':['gini'],'max_depth':list(range(2,14))}
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state = 42)
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state =42), dt_param_grid, scor
dt_grid.fit(X_train, y_train)
```

Out[63]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=10, n_splits=10, random_state=42),
             estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini'],
                         'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]},
             scoring='balanced_accuracy')
```

In [64]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,

dt_grid_pred = dt_grid.predict(X_test)

print('Decision Tree Tuned Performance:')
print('-------------------------------')
print('Accuracy         : ', accuracy_score(y_test, dt_grid_pred))
print('F1 Score         : ', f1_score(y_test, dt_grid_pred))
print('Precision        : ', precision_score(y_test, dt_grid_pred))
print('Recall           : ', recall_score(y_test, dt_grid_pred))
print('Confusion Matrix:\n ', confusion_matrix(y_test, dt_grid_pred))
```

```
Decision Tree Tuned Performance:
-------------------------------
Accuracy         :  0.8851040821843742
F1 Score         :  0.5994344957587181
Precision        :  0.654320987654321
Recall           :  0.5530434782608695
Confusion Matrix:
  [[2956  168]
 [ 257  318]]
```

In [51]:
```
dt_grid.best_score_
```

Out[51]:
```
0.8981917755675722
```

Ex 5.2 Random Forests with 1, 10, or 100 trees and 2,3,5, or 10 for max_depth.

In [52]:
```
rf_param_grid = {'n_estimators':[1,10,100],
                 'max_depth':[2,3,5,10]}
rf_grid = GridSearchCV(RandomForestClassifier(random_state =42),rf_param_grid, scori
rf_grid.fit(X_train, y_train)
```

```
Fitting 100 folds for each of 12 candidates, totalling 1200 fits
```

Out[52]:
```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=10, n_splits=10, random_state=42),
             estimator=RandomForestClassifier(random_state=1),
             param_grid={'max_depth': [2, 3, 5, 10],
                         'n_estimators': [1, 10, 100]},
             verbose=1)
```

In [61]:
```
rfm_grid_pred = rf_grid.predict(X_test)

print('Random Forest Tuned Performance:')
print('---------------------------------')
print('Accuracy        : ', accuracy_score(y_test, rfm_grid_pred))
print('F1 Score        : ', f1_score(y_test, rfm_grid_pred))
print('Precision       : ', precision_score(y_test, rfm_grid_pred))
print('Recall          : ', recall_score(y_test, rfm_grid_pred))
print('Confusion Matrix:\n ', confusion_matrix(y_test, rfm_grid_pred))
```

```
Random Forest Tuned Performance:
--------------------------------
Accuracy        :  0.8934847256015139
F1 Score        :  0.5808510638297872
Precision       :  0.7479452054794521
Recall          :  0.4747826086956522
Confusion Matrix:
  [[3032   92]
 [ 302  273]]
```

In [53]:
```
rf_grid.best_score_
```

Out[53]:
```
0.8998486518818933
```

In [143…
```
rf_grid.best_params_
```

Out[143…
```
{'max_depth': 10, 'n_estimators': 100}
```

Exercise 6. Oversampling

Use oversampling to create a balanced training dataset. Look at the class imblearn.over_sampling.RandomOverSampler for that purpose.

Ex. 6.1: What does the above class do?

Class to perform random over-sampling.

Object to over-sample the minority class(es) by picking samples at random with replacement.

Ex. 6.2: Why is oversampling only applied to the training dataset (not to the test data)?

Random undersampling involves randomly selecting examples from the majority class and deleting them from the training dataset

Oversampling is only done on training and not on the test data because test data should contain unseen samples so it would not overfit and give you better evaluation of training process.

Ex.6.3: Optimize Random Forest with the same search grid as before, but trained on a balanced training dataset.

In [130…

```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
rf_res_grid = GridSearchCV(RandomForestClassifier(random_state =42),rf_param_grid, s
X_res, y_res = ros.fit_resample(X_train, y_train)
rf_res_grid.fit(X_res, y_res)
```

Out[130…

```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=10, n_splits=10, random_state=42),
             estimator=RandomForestClassifier(random_state=42),
             param_grid={'max_depth': [2, 3, 5, 10],
                         'n_estimators': [1, 10, 100]},
             scoring='balanced_accuracy')
```

In [131…

```
rfm_res_pred = rf_res_grid.predict(X_test)

print('Random Forest Tuned Performance:')
print('-------------------------------')
print('Accuracy        : ', accuracy_score(y_test, rfm_res_pred))
print('F1 Score        : ', f1_score(y_test, rfm_res_pred))
print('Precision       : ', precision_score(y_test, rfm_res_pred))
print('Recall          : ', recall_score(y_test, rfm_res_pred))
print('Confusion Matrix:\n ', confusion_matrix(y_test, rfm_res_pred))
```

```
Random Forest Tuned Performance:
-------------------------------
Accuracy        :  0.874831035414977
F1 Score        :  0.6583025830258302
Precision       :  0.5717948717948718
Recall          :  0.7756521739130435
Confusion Matrix:
  [[2790  334]
 [ 129  446]]
```

Exercise 7: Interpretation

Evaluate the resulting three algorithms of the three above cross validation experiments (Decision Tree and two versions of Random Forest).

In [107…

```
from sklearn.metrics import classification_report
dt_report = classification_report(y_test, dt_grid_pred, output_dict=True)
```

Ex. 7. 1: Prepare a data frame in which the evaluation results of algorithms can be stored with columns for the algorithm, accuracy, balanced accuracy, confusion matrix and the best hyperparameters of the algorithm.

In [165…

```
#creating a dataframe for evaluation metrics

df_metrics = pd.DataFrame(columns =['classifiers', 'accuracy', 'balanced accuracy',
df_metrics
```

Out[165…

classifiers	accuracy	balanced accuracy	confusion matrix	best_hyperparameters

Ex. 7.2: On which dataset should the performance of algorithms (with already optimized hyper parameters) be compared (training data, test data, full data)?

Ans: Test data

Ex. 7.3: For each algorithm report the best choice of hyperparameters found using the above cross validations.

In [124…

```
dt_grid.best_estimator_
```

Out[124…

```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [132…

```
rf_grid.best_estimator_
```

Out[132…

```
BalancedRandomForestClassifier(max_depth=10, random_state=1)
```

In [134…

```
rf_res_grid.best_estimator_
```

Out[134…

```
RandomForestClassifier(max_depth=10, random_state=42)
```

Ex. 7.4: Compare three classifiers regarding both accuracy and balanced accuracy. Recommend a setting for use in production.

In [141…

```
from sklearn.metrics import balanced_accuracy_score
dt_acc=accuracy_score(y_test,dt_grid_pred)
print(dt_acc)
dt_bal_acc=balanced_accuracy_score(y_test,dt_grid_pred)
print(dt_bal_acc)
rf_acc=accuracy_score(y_test,rfm_grid_pred)
print(rf_acc)
rf_bal_acc=balanced_accuracy_score(y_test,rfm_grid_pred)
print(rf_bal_acc)
rf_res_acc=accuracy_score(y_test,rfm_res_pred)
print(rf_res_acc)
rf_res_bal_acc=balanced_accuracy_score(y_test,rfm_res_pred)
print(rf_res_bal_acc)
```

```
0.8851040821843742
0.7496331347770417
0.8934847256015139
0.7226665924400156
0.874831035414977
0.8343689806825141
```

From the above performance score, we can say that Random Forest with accurcy score 0.8934847256015139 yields the best result. I recommend a Random Forest algorithm with n_estimator = 100 and max_depth = 10