HOTEL RESERVATION CANCELLATIONS PREDICTION

COMP 542 - FALL 2022
PROF. MANSOUREH LORD

By -
SREE DIVYA SUDAGONI
PRATHYUSHA DIWAKARLA

PROJECT DESCRIPTION

The main idea of the hotel booking cancellation prediction project was to find the best classification model for predicting booking cancellations and the best explanatory variables for customer cancellations.

All the work done was made in Python using Jupyter Notebook and open python sourced libraries.

HOTEL BOOKING DEMAND DATASET - https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand

DATA SET DESCRIPTION

- The data contains 32 features on 2 hotels in Portugal.
- Each observation represents hotel booking between July,2015 and August,2017.
- The target variable is ‘Is Cancelled’
- Data for 119390 bookings with 32 features (20 categorical and 12 numerical columns)

In [3]:

```
#UNDERSTANDING THE DATA
df.head()
#resulting first five rows in the data set
```

Out[3]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month
0	Resort Hotel	0	342	2015	July
1	Resort Hotel	0	737	2015	July
2	Resort Hotel	0	7	2015	July
3	Resort Hotel	0	13	2015	July
4	Resort Hotel	0	14	2015	July

5 rows × 32 columns

PREPROCESSING OF DATA

FEATURE EXTRACTION

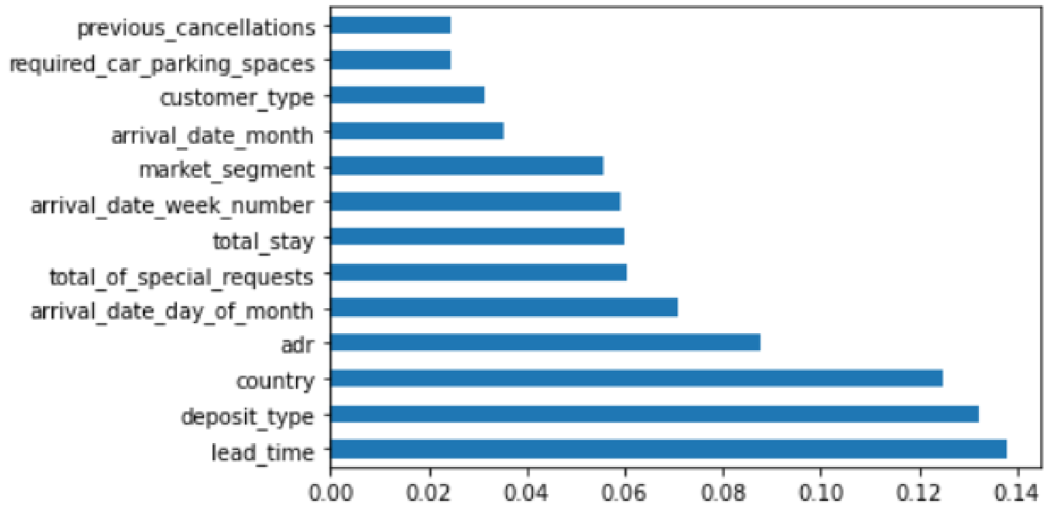
- Now that we have extracted the required features, we have 14 features in total excluding target class.

```
# perform feature selection using Feature Importance
model = ExtraTreesClassifier()
model.fit(x_feat, y_feat)

print(model.feature_importances_)

feat_importances = pd.Series(model.feature_importances_, index = x_fe
feat_importances.nlargest(13).plot.barh()
plt.show()
```

```
[0.01524793 0.13794892 0.03514585 0.05910397 0.07067619 0.01353106
 0.12503737 0.05563193 0.01775188 0.00523049 0.02450703 0.0023263
 0.02342737 0.13203541 0.0313052  0.08788761 0.02459407 0.06044288
 0.01837892 0.05978961]
```



ENCODING THE DATA

Label Encoding refers to converting the labels into a numeric form to convert them into the machine-readable form

Involves converting each value in a column to a number

```python
#Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form
df_le = df.copy()
le = LabelEncoder()
categoricals = [
    'arrival_date_month',
    'meal',
    'country',
    'market_segment',
    'distribution_channel',
    'reserved_room_type',
    'assigned_room_type',
    'deposit_type',
    'customer_type',
    'reservation_status',
]
for col in categoricals:
    df_le[col] = le.fit_transform(df_le[col])

plt.figure(figsize=(20, 15))
sns.heatmap(df_le.corr(), annot=True, fmt='.2f');
```

```python
df_le
```

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights
0	1	0	342	2015	5	27	1	0
1	1	0	737	2015	5	27	1	0
2	1	0	7	2015	5	27	1	0
3	1	0	13	2015	5	27	1	0
4	1	0	14	2015	5	27	1	0
...	...	...	...	...	...	...	...	...
119385	0	0	23	2017	1	35	30	2
119386	0	0	102	2017	1	35	31	2
119387	0	0	34	2017	1	35	31	2
119388	0	0	109	2017	1	35	31	2
119389	0	0	205	2017	1	35	29	2

SCALING THE FEATURES

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range

```
clean_data_scal = df_le.drop('is_canceled', axis = 1)

robust = RobustScaler()
robust.fit(clean_data_scal)
scaled_df = robust.transform(clean_data_scal)
scaled_df = pd.DataFrame(scaled_df, columns = clean_data_scal.columns)
scaled_df.head()
```

	hotel	lead_time	arrival_date_month	arrival_date_week_number	arrival_date_day_of_month	adults	country	market_segment	distribution_channel	is_repea
0	1.0	1.922535	-0.166667	-0.045455	-1.0	0.0	0.683544	-1.0	-2.0	
1	1.0	4.704225	-0.166667	-0.045455	-1.0	0.0	0.683544	-1.0	-2.0	
2	1.0	-0.436620	-0.166667	-0.045455	-1.0	-1.0	-0.278481	-1.0	-2.0	
3	1.0	-0.394366	-0.166667	-0.045455	-1.0	-1.0	-0.278481	-1.5	-3.0	
4	1.0	-0.387324	-0.166667	-0.045455	-1.0	0.0	-0.278481	0.5	0.0	

HANDLING IMBALANCED DATASET

- When the data is imbalanced there is a chance that the model will be biased towards majority class.
- Synthetic Minority Oversampling Technique or SMOTE is another technique to oversample the minority class.
- Simply adding duplicate records of minority class often don’t add any new information to the model.
- In SMOTE new instances are synthesized from the existing data. If we explain it in simple words, SMOTE looks into minority class instances and use k nearest neighbor to select a random nearest neighbor, and a synthetic instance is created randomly in feature space.
- We still haven't balanced the data set as SMOTE tends to create a large no. of noisy data points in feature space.

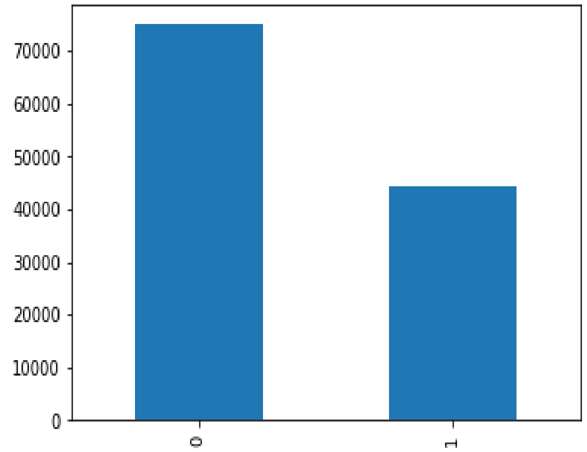
In [30]:
```
df['is_canceled'].value_counts()
```

Out[30]:
```
0    75010
1    44198
Name: is_canceled, dtype: int64
```

In [31]:
```
df['is_canceled'].value_counts().plot(kind = 'bar')

#If we look at the bar plot of target variable there
#is approximately 2:1 ratio between majority and minority class
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x15283ca4430>

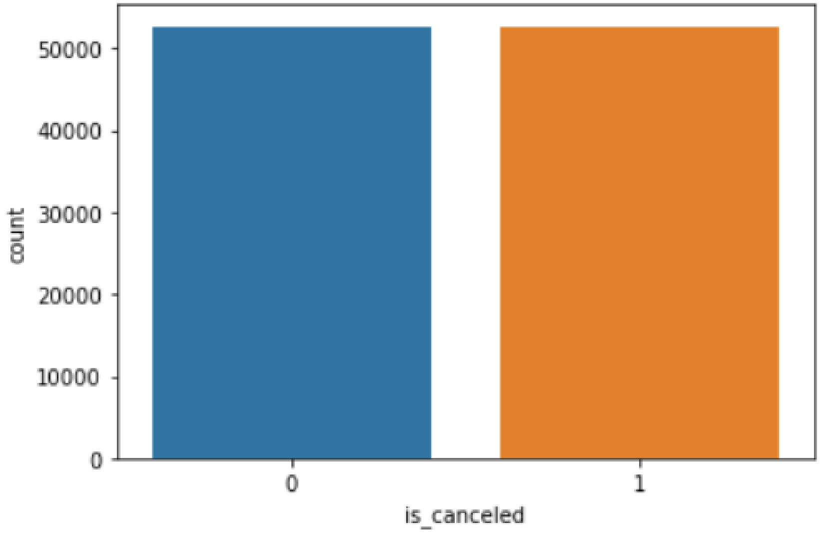


HANDLING IMBALANCED DATASET

```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(y_sm, data = df_sm)

# Show the plot
plt.show()
```



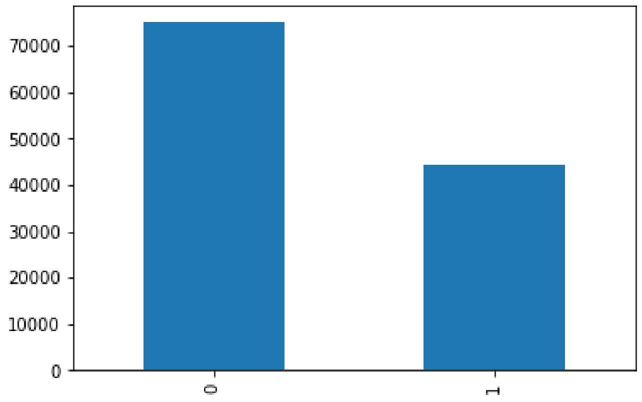

```
In [30]: df['is_canceled'].value_counts()

Out[30]: 0    75010
         1    44198
         Name: is_canceled, dtype: int64
```

```
In [31]: df['is_canceled'].value_counts().plot(kind = 'bar')

         #If we look at the bar plot of target variable there
         #is approximately 2:1 ratio between majority and minority class

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x15283ca4430>
```



MODEL BULDING

- Algorithms used –
  - K-Nearest Neighbors Algorithm
  - Decision Tree Algorithm

- Smote Technique –
  - K-Nearest Neighbors using Smote
  - Decision Tree using Smote

DECISION TREE ALGORITHM

DECISION TREE

```
pipe_DT = Pipeline([
    ("algo", DecisionTreeClassifier())
])
```

```
param_DT = {
    'algo__min_samples_split': [2,1,3,4,6,8,10,],
    'algo__max_depth': [None,1,2,4,8,10,12,14,18, 20],
    'algo__min_samples_leaf':[1,2,4,5,8]
}

model_DT = GridSearchCV(estimator=pipe_DT, param_grid=param_DT, cv = 3, n_jobs = -1, verbose = 1, scoring='accuracy')
model_DT.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 350 candidates, totalling 1050 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:    8.8s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:   13.8s
[Parallel(n_jobs=-1)]: Done 776 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 1050 out of 1050 | elapsed:   40.2s finished
```

```
GridSearchCV(cv=3,
             estimator=Pipeline(steps=[('algo', DecisionTreeClassifier())]),
             n_jobs=-1,
             param_grid={'algo__max_depth': [None, 1, 2, 4, 8, 10, 12, 14, 18,
                                             20],
                         'algo__min_samples_leaf': [1, 2, 4, 5, 8],
                         'algo__min_samples_split': [2, 1, 3, 4, 6, 8, 10]},
             scoring='accuracy', verbose=1)
```

DECISION TREE ALGORITHM

```
DT_tuned = model_DT.best_estimator_
DT_tuned_train = model_DT.best_score_
y_pred_DT_tuned = DT_tuned.predict(X_test)
```

```
recall_DT_tuned = recall_score(y_test, y_pred_DT_tuned)
acc_DT_tuned = accuracy_score(y_test, y_pred_DT_tuned)
precision_DT_tuned = precision_score(y_test, y_pred_DT_tuned)
f1_DT_tuned = f1_score(y_test, y_pred_DT_tuned)
acc_DT_tuned_train = DT_tuned_train

print(f"Training Accuracy : {acc_DT_tuned_train}")
print(f"Testing Accuracy : {acc_DT_tuned}")
```

```
Training Accuracy : 0.8330162376901078
Testing Accuracy : 0.8345478404109781
```

```
print(classification_report(y_test, y_pred_DT_tuned))
```

	precision	recall	f1-score	support
0	0.85	0.89	0.87	22455
1	0.80	0.74	0.77	13362
accuracy			0.83	35817
macro avg	0.83	0.82	0.82	35817
weighted avg	0.83	0.83	0.83	35817

DECISION TREE– USING SMOTE

```
#by using SMOTE

print(classification_report(y_test, y_pred_dtsm))
print(confusion_matrix(y_test, y_pred_dtsm))
confus_gbsm = confusion_matrix(y_test, y_pred_dtsm)
sns.heatmap(confus_gbsm, annot = True)
```

```
              precision    recall  f1-score   support

           0       0.87      0.86      0.86     22654
           1       0.76      0.77      0.76     13163

    accuracy                           0.83     35817
   macro avg       0.81      0.81      0.81     35817
weighted avg       0.83      0.83      0.83     35817

[[19383  3271]
 [ 2992 10171]]

<matplotlib.axes._subplots.AxesSubplot at 0x218305bf160>
```


```
sm = SMOTE(random_state = 42)
X_train=np.array(X_train)
X_sm, y_sm = sm.fit_resample(X_train, y_train)
```

```
sm = SMOTE(random_state = 42)
X_sm, y_sm = sm.fit_resample(X_train, y_train)
```

```
# X_train setelah oversampling dengan SMOTE
X_train =  pd.DataFrame(X_train, columns = X.columns)
```

```
# Predict with SMOTE
y_pred_dtsm = dt_model_sm.predict(X_test)
proba_dtsm = dt_model_sm.predict_proba(X_test)
```

```
# Cross Validation score (SMOTE)

cross_val_score(dt_model_sm, X_test, y_test, cv = 10).mean()
```

K-NEAREST NEIGHBOURS

```
pipe_KNN = Pipeline([
    ('algo', KNeighborsClassifier())
])
```

```
param_KNN = {
    'algo__n_neighbors': [5, 10, 15, 20, 30, 40],
    'algo__weights':['uniform', 'distance'],
    'algo__p':[2,1]
}

model_KNN = GridSearchCV(estimator=pipe_KNN, param_grid=param_KNN, cv = 3, n_jobs = -1, verbose = 1)
model_KNN.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:  6.7min finished
```

```
GridSearchCV(cv=3, estimator=Pipeline(steps=[('algo', KNeighborsClassifier())]),
             n_jobs=-1,
             param_grid={'algo__n_neighbors': [5, 10, 15, 20, 30, 40],
                         'algo__p': [2, 1],
                         'algo__weights': ['uniform', 'distance']},
             verbose=1)
```

```
KNN_tuned = model_KNN.best_estimator_
KNN_tuned_train = model_KNN.best_score_
y_pred_KNN_tuned = KNN_tuned.predict(X_test)
```

K-NEAREST NEIGHBOURS

```
recall_KNN_tuned = recall_score(y_test, y_pred_KNN_tuned)
acc_KNN_tuned = accuracy_score(y_test, y_pred_KNN_tuned)
precision_KNN_tuned = precision_score(y_test, y_pred_KNN_tuned)
f1_KNN_tuned = f1_score(y_test, y_pred_KNN_tuned)
acc_KNN_tuned_train = KNN_tuned_train

print(f"Training Accuracy : {acc_KNN_tuned_train}")
print(f"Testing Accuracy : {acc_KNN_tuned}")


# we see now that KNN doesn't have an overfitting condition and KNN have
#a better accuracy score compared to logistic regression
# we also see that after hyperparameter tuning KNN has a better
#testing score compared to it's based model
```

```
Training Accuracy : 0.8505342762441517
Testing Accuracy : 0.8550967417706675
```

```
print(classification_report(y_test, y_pred_KNN_tuned))
```

```
              precision    recall  f1-score   support

           0       0.85      0.93      0.89     22455
           1       0.86      0.72      0.79     13362

    accuracy                           0.86     35817
   macro avg       0.86      0.83      0.84     35817
weighted avg       0.86      0.86      0.85     35817
```

K-NEAREST NEIGHBOURS – USING SMOTE

```
# by using SMOTE

print(classification_report(y_test, y_pred_knnsm))
print(confusion_matrix(y_test, y_pred_knnsm))
confus_knnsm = confusion_matrix(y_test, y_pred_knnsm)
sns.heatmap(confus_knnsm, annot = True)
```

```
              precision    recall  f1-score   support

           0       0.89      0.86      0.87     22654
           1       0.77      0.82      0.79     13163

    accuracy                           0.84     35817
   macro avg       0.83      0.84      0.83     35817
weighted avg       0.85      0.84      0.84     35817

[[19488  3166]
 [ 2420 10743]]

<matplotlib.axes._subplots.AxesSubplot at 0x218389ff3a0>
```


```
sm = SMOTE(random_state = 42)
X_train=np.array(X_train)
X_sm, y_sm = sm.fit_resample(X_train, y_train)
```

```
sm = SMOTE(random_state = 42)
X_sm, y_sm = sm.fit_resample(X_train, y_train)
```

```
# X_train setelah oversampling dengan SMOTE
X_train =  pd.DataFrame(X_train, columns = X.columns)
```

```
# Predict with SMOTE
y_pred_knnsm = KNN_model_sm.predict(X_test)
proba_knnsm = KNN_model_sm.predict_proba(X_test)
```

```
# Cross Validation score (SMOTE)

cross_val_score(KNN_model_sm, X_test, y_test, cv = 10).mean()
```

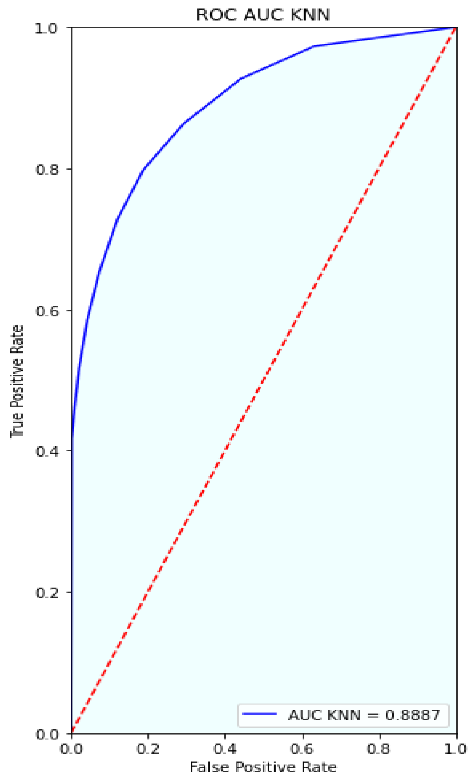
```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
```

K-NEAREST NEIGHBOURS – ROC CURVE

```
def knn_curve():
    fpr_kn, tpr_kn, thres_kn = roc_curve(y_test, proba_kn[:,1])
    roc_auc_kn = auc(fpr_kn, tpr_kn)

    plt.subplot(154)
    plt.title('ROC AUC KNN')
    plt.plot(fpr_kn, tpr_kn, 'blue', label='AUC KNN = {}'.format(round(roc_auc_kn,4)))
    plt.plot([0,1],[0,1], 'r--')
    plt.xlim([0,1])
    plt.ylim([0,1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.fill_between(fpr_kn,tpr_kn, 0, facecolor='azure', alpha=1)
    plt.legend(loc = 'lower right')
```

```
plt.figure(figsize = (25, 10))
knn_curve()
```



CONCLUSION

Tuned K-Nearest Algorithm has the best accuracy when compared with Decision Tree Algorithm.

After using Smote, K-Nearest algorithm has the best accuracy among the two.

FUTURE RESEARCH

- Can a similar result have obtained given any location?
- Can we train the model with more hotels integrated into the model?

Thank you