Project - Classification

In this project, I will perform a classification task. For this project, I use the **In vehicle coupon recommendation** dataset . This dataset can be found online https://archive.ics.uci.edu/ml/datasets/in-vehicle+coupon+recommendation.

Objective of the analysis

The objective of the analysis for this course peer-review project is to use the "**In vehicle coupon reccomandation**" dataset and perform classification on it. The first step will be to perform EDA and data cleaning on the dataset, and after use the cleaned dataset for classification purposes. The main goal is to use classification on the "**In vehicle coupon reccomandation**" dataset for **prediction**.

Brief description of the dataset and summary attributes

To analyze the dataset, the first step is that to import the necessary Python libraries which are the following:

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(context="notebook")
```

Now I specify the path of the file on my computer to import it as a Pandas dataframe

In [ ]:
```
PATH = "/..../in-vehicle-coupon-recommendation.csv"

df = pd.read_csv(PATH, delimiter=',', header='infer')

df
```

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CoffeeHouse	CarryAway
0	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Female	21	Unmarried partner	...	never	NaN
1	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Female	21	Unmarried partner	...	never	NaN
2	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Female	21	Unmarried partner	...	never	NaN
3	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	2h	Female	21	Unmarried partner	...	never	NaN
4	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Female	21	Unmarried partner	...	never	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	never	1~3
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	never	1~3
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	never	1~3
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	never	1~3
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	never	1~3

12684 rows × 26 columns

Fig.1. Display of a part of the in-vehicle-coupon recommendation dataset. Not all columns are shown.

As one can see, the dataset has 12684 rows and 26 columns. It is a medium size dataset. One can get this information by using the methods df.shape[0] and df.shape[1]. At this point is very helpful to have general information about the dataset. This can be obtained by using the following code:

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12684 entries, 0 to 12683
Data columns (total 26 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   destination           12684 non-null  object
 1   passanger             12684 non-null  object
 2   weather               12684 non-null  object
 3   temperature           12684 non-null  int64 
 4   time                  12684 non-null  object
 5   coupon                12684 non-null  object
 6   expiration            12684 non-null  object
 7   gender                12684 non-null  object
 8   age                   12684 non-null  object
 9   maritalStatus         12684 non-null  object
 10  has_children          12684 non-null  int64 
 11  education             12684 non-null  object
 12  occupation            12684 non-null  object
 13  income                12684 non-null  object
 14  car                   108 non-null    object
 15  Bar                   12577 non-null  object
 16  CoffeeHouse           12467 non-null  object
 17  CarryAway             12533 non-null  object
 18  RestaurantLessThan20  12554 non-null  object
 19  Restaurant20To50      12495 non-null  object
 20  toCoupon_GEQ5min      12684 non-null  int64 
 21  toCoupon_GEQ15min     12684 non-null  int64 
 22  toCoupon_GEQ25min     12684 non-null  int64 
 23  direction_same        12684 non-null  int64 
 24  direction_opp         12684 non-null  int64 
 25  Y                     12684 non-null  int64 
dtypes: int64(8), object(18)
memory usage: 2.5+ MB
```

As one can see, the dataset has in total 25 features where the majority of them are object type, and the rest is int64. So, as one can see here I have to deal with a mixed (data) dataset. Among the features, there is the feature that will select as the target variable. As one can see from the link above regarding the in-vehicle-coupon recommandation dataset, here we are dealing with a situation where a person will either accept or decline a coupon based on several weather conditions. So, the passanger reply will be either yes or no and this answer is encoded in the feature with name **Y**.

It is useful at this stage to have some information about the target variable Y with the following command

In [ ]:
```
df['Y'].value_counts()
```

```
1    7210
0    5474
Name: Y, dtype: int64
```

As one can see, the chosen feature Y as the target variable has 7210 values of 1 and 5474 values of 0. This simple information tells that the target variable is binary, already encoded, and has balanced values of the binary variables. There is not necessary to do anything with this variable. On the other hand it is useful to have some descriptive statistics for the numerical variables:

In [ ]:
```
df.describe()
```

	temperature	has_children	toCoupon_GEQ5min	toCoupon_GEQ15min	toCoupon_GEQ25min	direction_same
count	12684.000000	12684.000000	12684.0	12684.000000	12684.000000	12684.000000
mean	63.301798	0.414144	1.0	0.561495	0.119126	0.214759
std	19.154486	0.492593	0.0	0.496224	0.323950	0.410671
min	30.000000	0.000000	1.0	0.000000	0.000000	0.000000
25%	55.000000	0.000000	1.0	0.000000	0.000000	0.000000
50%	80.000000	0.000000	1.0	1.000000	0.000000	0.000000
75%	80.000000	1.000000	1.0	1.000000	0.000000	0.000000
max	80.000000	1.000000	1.0	1.000000	1.000000	1.000000

Fig. 2. A descriptive statistics table of the numerical values of the in-vehicle-coupon recommendation dataset.

As briefly mentioned above, in this peer-review course, I will use the **in vehicle coupon reccommandation** to perform classification. The target variable y will be the binary feature Y and the remaining feature will be the independent variables.

Brief summary EDA, data cleaning and feature engineering

In this section, I will be performing some EDA, data cleaning and feature engineering for the classification task. Since, the dataframe in Fig.1 has mixed types of features, is very important to categorize each of the features to have a general idea before performing any other action.

The first step is to clean the data as much as possible from NaN values. As one can see from the general information about the dataframe df, the **car** column has only 108 non-null values. So based on this fact one can safely remove this column from the database completely because there is almost nothing one can do to fill the NaN values because of their large number. On the other hand, the other features have a very small number of NaN values and their removal is not going to have a significant impact on the classification study.

By removing the **car** column and removing the NaN values in the remaining features (I do not show these steps because they are trivial), one gets the following cleaned dataframe:

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	CoffeeHouse	CarryAway	RestaurantLessThan20	Restaurant20To50
22	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Male	21	Single	...	less1	4~8	4~8	less1
23	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Male	21	Single	...	less1	4~8	4~8	less1
24	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar	1d	Male	21	Single	...	less1	4~8	4~8	less1
25	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Male	21	Single	...	less1	4~8	4~8	less1
26	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Male	21	Single	...	less1	4~8	4~8	less1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	never	1~3	4~8	1~3
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	never	1~3	4~8	1~3
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	never	1~3	4~8	1~3
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	never	1~3	4~8	1~3
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	never	1~3	4~8	1~3

12079 rows × 25 columns

Fig. 3. The cleaned dataset where NaN values and car column have been removed. The dataframe now has 25 columns, including the target feature.

Now by running an appropriate Python code, it is possible to show the unique values for each column as a list as follows

Variable	Unique Values
destination	3
passanger	4
weather	3
temperature	3
time	5
coupon	5
expiration	2
gender	2
age	8
maritalStatus	5
has_children	2
education	6
occupation	25
income	9
Bar	5
CoffeeHouse	5
CarryAway	5
RestaurantLessThan20	5
Restaurant20To50	5
toCoupon_GEQ5min	1
toCoupon_GEQ15min	2
toCoupon_GEQ25min	2
direction_same	2
direction_opp	2
Y	2

One interesting thing to note about the dataframe above is the feature '**toCoupon_GEQ5min**'. This feature has only one unique value which is equal to 1. Even this feature needs to be removed because has only one value and cannot be used in the analysis since it is highly skewed and it has not any predictive power. Therefore by removing this feature, the final dataset for analysis is the following:

	destination	passanger	weather	temperature	time	coupon	expiration	gender	age	maritalStatus	...	Bar	CoffeeHouse	C
22	No Urgent Place	Alone	Sunny	55	2PM	Restaurant(<20)	1d	Male	21	Single	...	never	less1	
23	No Urgent Place	Friend(s)	Sunny	80	10AM	Coffee House	2h	Male	21	Single	...	never	less1	
24	No Urgent Place	Friend(s)	Sunny	80	10AM	Bar	1d	Male	21	Single	...	never	less1	
25	No Urgent Place	Friend(s)	Sunny	80	10AM	Carry out & Take away	2h	Male	21	Single	...	never	less1	
26	No Urgent Place	Friend(s)	Sunny	80	2PM	Coffee House	1d	Male	21	Single	...	never	less1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
12679	Home	Partner	Rainy	55	6PM	Carry out & Take away	1d	Male	26	Single	...	never	never	
12680	Work	Alone	Rainy	55	7AM	Carry out & Take away	1d	Male	26	Single	...	never	never	
12681	Work	Alone	Snowy	30	7AM	Coffee House	1d	Male	26	Single	...	never	never	
12682	Work	Alone	Snowy	30	7AM	Bar	1d	Male	26	Single	...	never	never	
12683	Work	Alone	Sunny	80	7AM	Restaurant(20-50)	2h	Male	26	Single	...	never	never	

12079 rows × 24 columns

Fig. 4. Cleaned dataset with the columns car and toCoupon_GEQ5min columns removed. The dataset has 24 columns including the target column.

Data categorization

Data can be of different types and in the case study of this project, the data in the dataframe in Fig.4 has different formats. As I discussed above, the data are either as 'object' or 'int64'. It is very important to categorize each type of data present in the cleaned dataset of Fig.4 before doing any type of analysis.

In my case, one can see that the data are either **quantitative** or **qualitative**. In the latter case is important to separate qualitative data into **ordinal data** and **nominal data**.

The first step is to find which data are binary. One can see from the Pandas dataframe above that many features have just two values, namely, binary variables. By running the following code on the above Pandas dataframe, I get a list of the binary value features:

In [ ]:
```
binary_variables = list(df[df['Unique Values'] == 2].index)
binary_variables
```

```
['expiration',
 'gender',
 'has_children',
 'toCoupon_GEQ15min',
 'toCoupon_GEQ25min',
 'direction_same',
 'direction_opp',
 'Y']
```

The next step is to find which data are nominal (or categorical) and which are ordinal. Nominal data do not have a specific order while nominal data have a specific order. To find each type of this data is not a trivial thing and some ordinal data can be seen as quantitative data as well, so, careful attention must be paid to this fact.

By looking at the data carefully, the following features(columns) can be classified as **ordinal data**:

```
['weather',
 'temperature',
 'time',
 'age',
 'education',
 'income',
 'Bar',
 'CoffeeHouse',
 'CarryAway',
 'RestaurantLessThan20',
 'Restaurant20To50']
```

By running an appropriate code, each of the **ordinal features** have the following values:

```
[['weather', ['Sunny', 'Rainy', 'Snowy']],
 ['temperature', [55, 80, 30]],
 ['time', ['2PM', '10AM', '6PM', '7AM', '10PM']],
 ['age', ['21', '46', '26', '31', '41', '50plus', '36', 'below21']],
 ['education',
  ['Bachelors degree',
   'Some college - no degree',
   'Associates degree',
   'High School Graduate',
   'Graduate degree (Masters or Doctorate)',
   'Some High School']],
 ['income',
  ['$62500 - $74999',
   '$12500 - $24999',
   '$75000 - $87499',
   '$50000 - $62499',
   '$37500 - $49999',
   '$25000 - $37499',
   '$100000 or More',
   '$87500 - $99999',
   'Less than $12500']],
 ['Bar', ['never', 'less1', '1~3', 'gt8', '4~8']],
 ['CoffeeHouse', ['less1', '4~8', '1~3', 'gt8', 'never']],
 ['CarryAway', ['4~8', '1~3', 'gt8', 'less1', 'never']],
 ['RestaurantLessThan20', ['4~8', '1~3', 'less1', 'gt8', 'never']],
 ['Restaurant20To50', ['less1', 'never', '1~3', 'gt8', '4~8']]]
```

On the other hand the following features(columns) can be classified as **nominal data**

['destination', 'passanger', 'coupon', 'maritalStatus', 'occupation']

Again, by running an appropriate code, each of the **nominal features** has the following values:

```
[['destination', ['No Urgent Place', 'Home', 'Work']],
 ['passanger', ['Alone', 'Friend(s)', 'Kid(s)', 'Partner']],
 ['coupon',
  ['Restaurant(<20)',
   'Coffee House',
   'Bar',
   'Carry out & Take away',
   'Restaurant(20-50)']],
 ['maritalStatus',
  ['Single', 'Married partner', 'Unmarried partner', 'Divorced', 'Widowed']],
 ['occupation',
  ['Architecture & Engineering',
   'Student',
   'Education&Training&Library',
   'Unemployed',
   'Healthcare Support',
   'Healthcare Practitioners & Technical',
   'Sales & Related',
   'Management',
   'Arts Design Entertainment Sports & Media',
   'Computer & Mathematical',
   'Life Physical Social Science',
   'Personal Care & Service',
   'Office & Administrative Support',
   'Construction & Extraction',
   'Legal',
   'Retired',
   'Community & Social Services',
   'Installation Maintenance & Repair',
   'Transportation & Material Moving',
   'Business & Financial',
   'Protective Service',
   'Food Preparation & Serving Related',
   'Production Occupations',
   'Building & Grounds Cleaning & Maintenance',
   'Farming Fishing & Forestry']]]
```

While it is easy to understand why some features are **nominal**, it might be not that easy why some features are considered as **ordinal**. For example, the features **temperature** and **time** have been labeled as ordinal. The first reason why these features have been considered as ordinal is because they have few unique values and their value difference is not easy to interpret. On the other hand, the features **age** and **income** are interval data that can be considered as ordinal data as well.

Feature encoding

Now that each feature has been properly classified, it is time to encode them with the appropriate encoding methods. Here I use scikit-learn and Pandas to encode each feature. **Binary** features can be encoded by using the **LabelBinarizer** sklearn method, while **ordinal** features can be encoded by using the **OrdinalEncoder** sklearn method. On the other hand, **nominal** features can be encoded by using either **OneHotEncoder** sklearn method or the Pandas **get_dummies** method.

I import the following sklearn modules:

In [ ]:
```
from sklearn.preprocessing import LabelBinarizer
```

In [ ]:
```
bin_enc = LabelBinarizer()
```

While it is easy to use encoding for binary features, it is not that easy for ordinal features. The reason is because usually one has to specify manually the encoding order for each value. The **OrdinalEncoder** method usually encodes ordinal features automatically and very often the order of encoding does not correspond to that what the user wants.

Here I will show the encoding only for the ordinal features because it is the most complex. For the ordinal features, I create the following dictionary:

```
{'weather': {'Snowy': 0, 'Rainy': 1, 'Sunny': 2},
 'temperature': {30: 0, 55: 1, 80: 2},
 'time': {'2PM': 2, '10AM': 1, '6PM': 3, '7AM': 0, '10PM': 4},
 'age': {'21': 1,
  '46': 6,
  '26': 2,
  '31': 3,
  '41': 5,
  '50plus': 7,
  '36': 4,
  'below21': 0},
 'education': {'Bachelors degree': 3,
  'Some college - no degree': 2,
  'Associates degree': 4,
  'High School Graduate': 1,
  'Graduate degree (Masters or Doctorate)': 5,
  'Some High School': 0},
 'income': {'$62500 - $74999': 5,
  '$12500 - $24999': 1,
  '$75000 - $87499': 6,
  '$50000 - $62499': 4,
  '$37500 - $49999': 3,
  '$25000 - $37499': 2,
  '$100000 or More': 8,
  '$87500 - $99999': 7,
  'Less than $12500': 0},
 'Bar': {'never': 0, 'less1': 1, '1~3': 2, 'gt8': 4, '4~8': 3},
 'CoffeeHouse': {'less1': 1, '4~8': 3, '1~3': 2, 'gt8': 4, 'never': 0},
 'CarryAway': {'4~8': 3, '1~3': 2, 'gt8': 4, 'less1': 1, 'never': 0},
 'RestaurantLessThan20': {'4~8': 3,
  '1~3': 2,
  'less1': 1,
  'gt8': 4,
  'never': 0},
 'Restaurant20To50': {'less1': 1, 'never': 0, '1~3': 2, 'gt8': 4, '4~8': 3}}
```

As one can see, for each feature I wrote a dictionary where each ordinal value has been "mapped" with positive integer numbers. For example, when the weather is Snowy, I labeled this value with 0, when is Rainy I labeled it with 1, and when is Sunny I labeled it with 3. I did similar operations for each feature. Unfortunately, there is no other way to do it in case when the ordinal values are not ordered by default. One has to do it manually.

After creating the dictionary and calling it as "ord_dic", I run the following code to replace each ordinal value with the appropriate numerical label value:

In [ ]:
```
for column in ordinal_variables:
    df[column] = df[column].map(ord_dic[column])
```

In the code above, I used the "map" Pandas method the perform the mapping between feature ordinal values and imputed label values. With the code above, I have already performed the encoding for each ordinal feature because I imputed the value and numerical label correspondence manually. It is not necessary to use **OrdinalEncoder** at this point. If one runs **OrdinalEncoder** nothing will change.

To perform the encoding for binary features, I run the following code:

In [ ]:
```
for column in binary_variables:
    df[column] = bin_enc.fit_transform(df[column])
```

The remaining step is to encode the nominal features. This can be easily done by running the following Pandas method:

In [ ]:
```
df = pd.get_dummies(df, columns = nominal_variables)
```

The latest code essentially completes all the necessary steps of feature encoding. It is useful to see how the final, cleaned and encoded dataset looks like in a Pandas dataframe.

	weather	temperature	time	expiration	gender	age	has_children	education	income	Bar	...	occupation_Management	occupation_Office & Administrative Support
22	2	1	2	0	1	1	0	3	5	0	...	0	0
23	2	2	1	1	1	1	0	3	5	0	...	0	0
24	2	2	1	0	1	1	0	3	5	0	...	0	0
25	2	2	1	1	1	1	0	3	5	0	...	0	0
26	2	2	2	0	1	1	0	3	5	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12679	1	1	3	0	1	2	0	3	6	0	...	0	0
12680	1	1	0	0	1	2	0	3	6	0	...	0	0
12681	0	0	0	0	1	2	0	3	6	0	...	0	0
12682	0	0	0	0	1	2	0	3	6	0	...	0	0
12683	2	2	0	1	1	2	0	3	6	0	...	0	0

12079 rows × 61 columns

Fig. 5. Final dataset after being cleaned and encoded.

As one can see, the final dataframe, after being cleaned and encoded has in total 12079 rows and 61 columns. The number of columns has slightly increased due to the encoding procedure. It is the encoding of nominal features that actually increases their total number.

Feature engineering

Now that data have been encoded, it is important to make some scaling if that is necessary. Since many algorithms use the distance among data points, it is important to use an appropriate scaling for some of the data.

As I have shown above, binary features all have values either 0 or 1. Nominal features after encoding with Pandas "get_dummies" all have values ether 0 or 1. On the other hand, ordinal features have different values that are not necessarily of only 0 or 1. For example, one can see in Fig. 5, the **income** feature has many different values ranging from 0 to 8. In this case, it important to scale the ordinal features only and bring them in the same scale as the binary and nominal features.

To scale the ordinal features only, I first import the **MinMaxScaler** and then apply it to fit and transform the ordinal features. To do these steps, I run the following code:

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
MM = MinMaxScaler()
```

In [ ]:
```
for column in ordinal_variables:
    df[column] = MM.fit_transform(df[column])
```

After applying the MinMaxScaler on the ordinal features, one gets the final Pandas dataframe:

	weather	temperature	time	expiration	gender	age	has_children	education	income	Bar	...	occupation_Management	occupation_Office & Administrative Support
22	1.0	0.5	0.50	0	1	0.142857	0	0.6	0.625	0.0	...	0	0
23	1.0	1.0	0.25	1	1	0.142857	0	0.6	0.625	0.0	...	0	0
24	1.0	1.0	0.25	0	1	0.142857	0	0.6	0.625	0.0	...	0	0
25	1.0	1.0	0.25	1	1	0.142857	0	0.6	0.625	0.0	...	0	0
26	1.0	1.0	0.50	0	1	0.142857	0	0.6	0.625	0.0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12679	0.5	0.5	0.75	0	1	0.285714	0	0.6	0.750	0.0	...	0	0
12680	0.5	0.5	0.00	0	1	0.285714	0	0.6	0.750	0.0	...	0	0
12681	0.0	0.0	0.00	0	1	0.285714	0	0.6	0.750	0.0	...	0	0
12682	0.0	0.0	0.00	0	1	0.285714	0	0.6	0.750	0.0	...	0	0
12683	1.0	1.0	0.00	1	1	0.285714	0	0.6	0.750	0.0	...	0	0

12079 rows × 61 columns

Fig. 6. Final dataset where data have been cleaned, encoded and scaled.

In Fig. 6 is shown the final data to be used below for the training and test split. One can see that the data are in the same range after being scaled with the MinMaxScaler as discussed above. Below is also shown the descriptive properties of the dataset shown in Fig. 6.

	count	mean	std	min	25%	50%	75%	max
weather	12079.0	0.842	0.330	0.0	1.0	1.0	1.00	1.0
temperature	12079.0	0.667	0.383	0.0	0.5	1.0	1.00	1.0
time	12079.0	0.473	0.358	0.0	0.0	0.5	0.75	1.0
expiration	12079.0	0.440	0.496	0.0	0.0	0.0	1.00	1.0
gender	12079.0	0.487	0.500	0.0	0.0	0.0	1.00	1.0
...	...	...	...	...	...	...	...	...
occupation_Retired	12079.0	0.039	0.194	0.0	0.0	0.0	0.00	1.0
occupation_Sales & Related	12079.0	0.089	0.284	0.0	0.0	0.0	0.00	1.0
occupation_Student	12079.0	0.124	0.330	0.0	0.0	0.0	0.00	1.0
occupation_Transportation & Material Moving	12079.0	0.018	0.133	0.0	0.0	0.0	0.00	1.0
occupation_Unemployed	12079.0	0.150	0.357	0.0	0.0	0.0	0.00	1.0

61 rows × 8 columns

Fig. 7. Descriptive properties of the dataset in Fig. 6 obtained by using the Pandas method 'df.describe().T'

Summary of training at least three classification models

Now that the data have been cleaned, encoded and scaled, it is the time to model them and obtain predictions by using at least 3 different algorithms. Independently on the algorithms used below, it is important first to select the data used as features and as target variables. I run the following Python code:

In [ ]:
```
y, X = df2['Y'], df2.drop(columns='Y')
```

In the code above, the target variable has been chosen as the column 'Y' which indicates the cases if the driver accepts the coupon or not. On the other hand, the features are those present in the dataset in Fig. 7 without the 'Y' column.

Classification with KNN

In this section, I use the KNN algorithm to perform classification. I use the follwing modules for my analysis:

In [ ]:
```
from sklearn.model_selection import KFold, GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classificati
from sklearn.pipeline import Pipeline
```

After Importing the modules above, I run the following codes that are self-explanatory:

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
KNN = KNeighborsClassifier()  # Instantiate the classifier
kf = KFold(shuffle=True, random_state=0, n_splits=5)    # KFold splitting r
Pipe = Pipeline([("classification", KNN)])   # Instantiate a pipeline
```

In the code above I used a test size of data splitting of 0.2 which is quite common. I also used the KFold method with 5 splits with shuffling. Now I run the following codes that are also self-explanatory:

In [ ]:
```
parameters = {'classification__algorithm': ["auto",  "ball_tree", "kd_tree
              'classification__metric': ['euclidean', 'manhattan', 'chebysh
grid = GridSearchCV(Pipe, parameters, cv=kf, scoring="f1", n_jobs=-1)

Model = grid.fit(X_train, y_train)
Model.best_score_, Model.best_params_
```

```
(0.7616381871473055,
 {'classification__algorithm': 'auto',
  'classification__metric': 'manhattan',
  'classification__n_neighbors': 27})
```

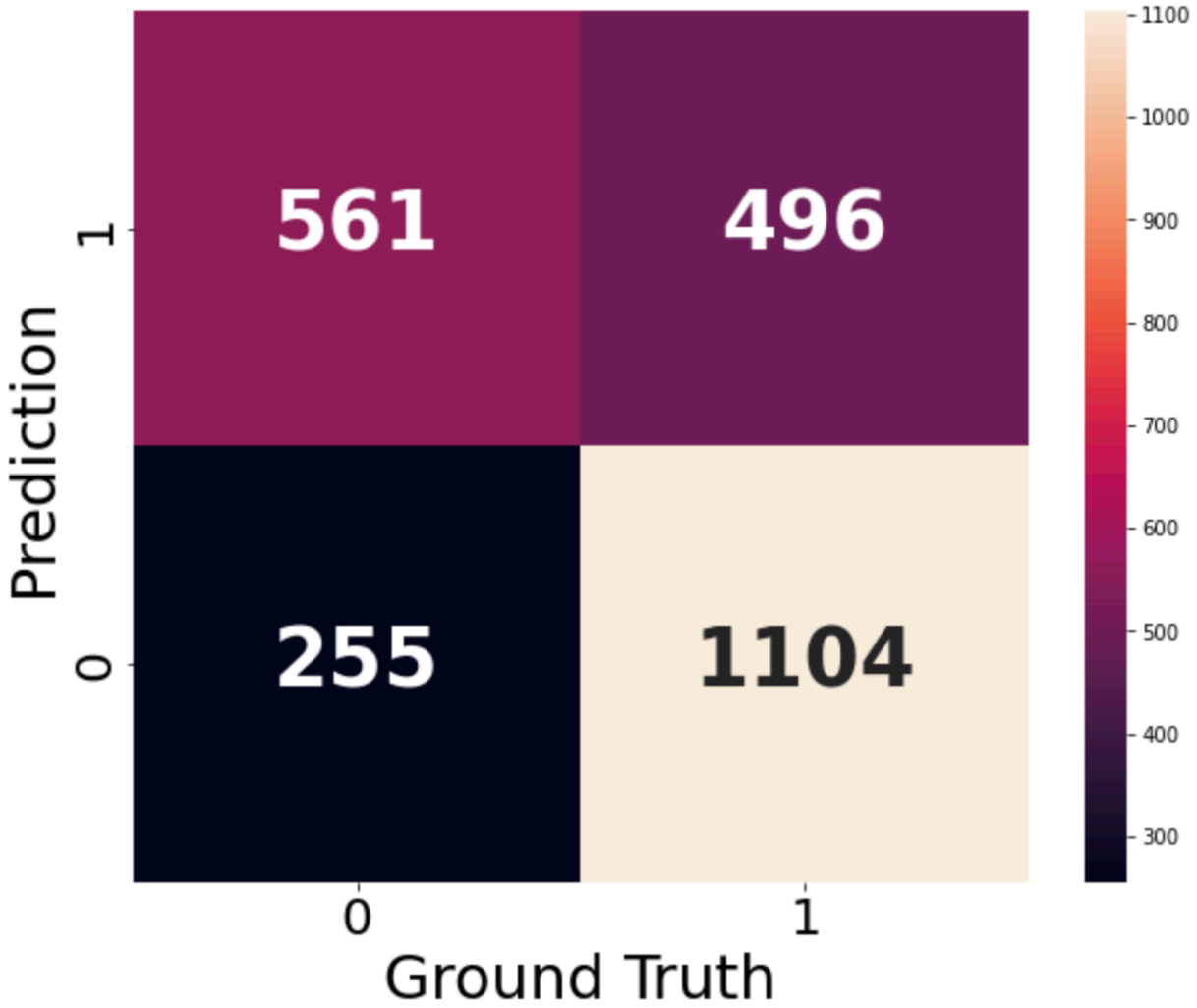
In the code above, I used different KNN algorithms, different distance metrics and a range of K nearest neighbors from 1 to 39 included. By using the GridSearchCV method with scoring method as F1-score and applying it to the train data, the GridSearchCV finds that the best model that fits the training data has an F1-score of 0.76. This score is achieved with the 'auto' algorithm, with the 'manhattan' metric distance and for K=27.

With the best selected model by GridSearcCV, one can apply this model to the test data for prediction. By running the code below on the **test data**, one gets:

In [ ]:
```
y_pred = Model.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.69	0.53	0.60	1057
1	0.69	0.81	0.75	1359
accuracy			0.69	2416
macro avg	0.69	0.67	0.67	2416
weighted avg	0.69	0.69	0.68	2416

In addition to the classification report, one can also produce the following confusion matrix of the **test data**



Classification with SVM

In this section I perform classification with the SVM method. This method has several hyper-parameters that complicate the situation with respect to the KNN method. The first thing to do is to import the SVM classifier with the following code:

In [ ]:
```
from sklearn.svm import SVC
svc = SVC()
Pipe = Pipeline([('classification', svc)])
```

After having imported the classifier and instantiated the method, I use again the same method for the KFold sampling as in the previous section. So again, the train-split size is again the same, cv=5 with random shuffling and the scoring method is f1. I run the following code lines:

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
svc = SVC()  # Instantiate the classifier
kf = KFold(shuffle=True, random_state=0, n_splits=5)    # KFold splitting r
Pipe = Pipeline([("classification", svc)])   # Instantiate a pipeline
```

In [ ]:
```
parameters_svc = {'classification__C': [0.001, 0.01, 1, 10, 20],
                  'classification__kernel': ['linear', 'poly', 'rbf', 'sigr
                  'classification__max_iter': [20000],
                  'classification__random_state': [0],
                  }
grid_svc = GridSearchCV(Pipe, parameters_svc, cv=kf, scoring="f1", n_jobs=-
```

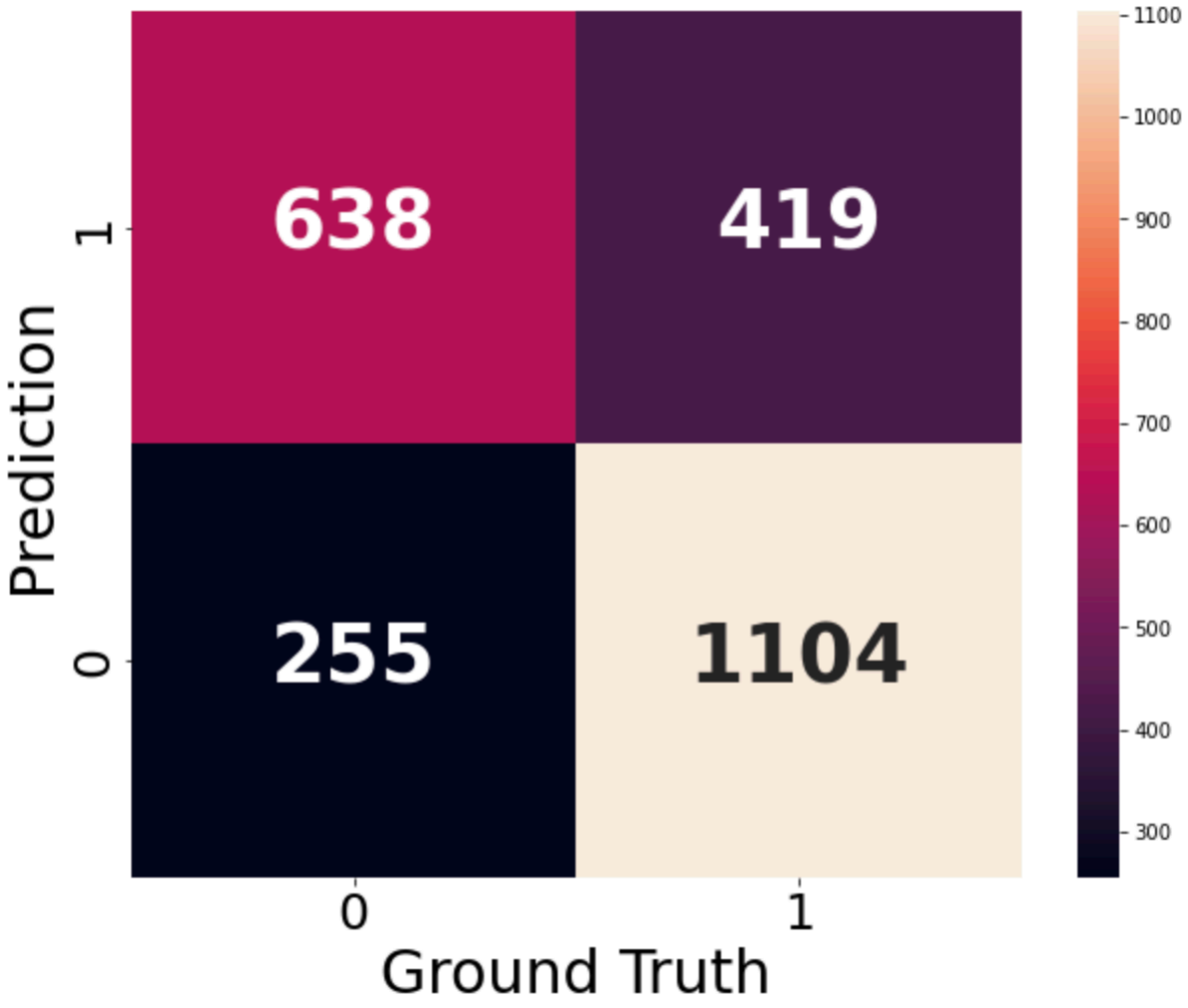
After running the above code on the train data, I get the following best results from the cross validation method:

```
(0.773607386563452,
 {'classification__C': 1,
  'classification__kernel': 'poly',
  'classification__max_iter': 20000,
  'classification__random_state': 0})
```

The classification report for the SVC algorithm applied on the **test data** is given by:

	precision	recall	f1-score	support
0	0.71	0.60	0.65	1057
1	0.72	0.81	0.77	1359
accuracy			0.72	2416
macro avg	0.72	0.71	0.71	2416
weighted avg	0.72	0.72	0.72	2416

The confusion matrix for SVC algorithm applied to the **test data** is the following:

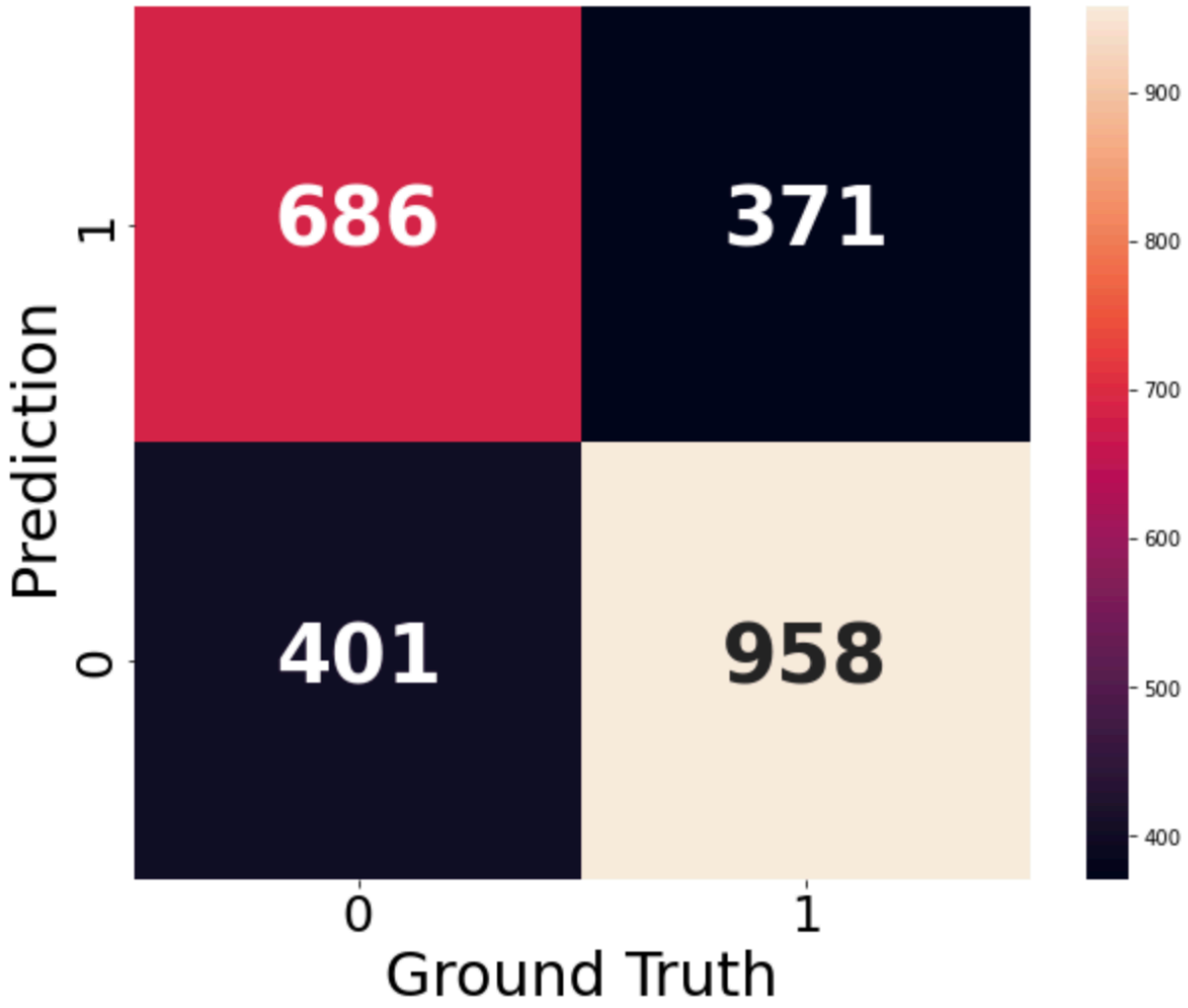


Classification with Decision Trees

In the case of decision trees, the splitting of data and the number of cross validation is the same as in the previous section. It is also the same the scoring method, namely 'f1'. By repeating the same steps as in the previous section, I get the following a result of **0.7180125218236085** of the GridSearchCV on the train data. The classification report of the **test data** is:

	precision	recall	f1-score	support
0	0.63	0.65	0.64	1057
1	0.72	0.70	0.71	1359
accuracy			0.68	2416
macro avg	0.68	0.68	0.68	2416
weighted avg	0.68	0.68	0.68	2416

The confusion matrix of the the **test data** is:



The following pandas dataframe gives the metric quantities for the train and test data for the best model obtained with the GridSearchCV:

	train	test
accuracy	0.998655	0.680464
precision	1.000000	0.720843
recall	0.997644	0.704930
f1	0.998821	0.712798

Fig. 8. Train and test different score methods for decisions trees.

Best Classifier

As seen in the previous section, I used three different algorithms to train and test the data. Each of these classifiers uses different methods to estimate the best parameters etc. Of these algorithms, the SVM classification model gives the best F1 scores for the positive and negative classes. The second model that performs also well is the KNN algorithm and the least performing is the decision tree algorithm.

So, the best model selected if the **SVC model** with $C = 1$ and **polynomial kernel**. This model give an f_1 score of 0.77 for the class 1 and a score of 0.65 for the class 0.

Summary of key findings, insights and possible next steps

As I have showed above, on analyzing the data, the SVC algorithm performed best in the test data. Each of the classification algorithms that have selected are the most used for classification and one cannot choose which is the best in the general case of many different datasets. One can see from Fig. 8, that decision trees tend to overfit on the train data. This is a well known property of decision trees and my analysis validates it further.

The f_1 score for each of these algorithms is good but it might be improved further. Some possible steps to improve it would be to check for a stratisfied splitting of the data, use feature selection like PCA and use wider ranges of the hyper-parameters C and γ for the SVC algorithm. Additional steps by using other algorithms such as random forest, gradient boosting etc. might help improve the final result.