tugas-3-dmc-2013-t1-kelompok-9

November 22, 2024

1 Tugas 3: Data Mining Cup 2013 - Task 1

1.0.1 Kelompok 9

- Ahmad Rayki Pahlevi 2301596
- Arya Jagadditha 2312239
- Muhammad Daffa Rizmawan Harahap 2310083
- Rasendriya Andhika 2305309
- Yattaqi Ahmad Faza 2311216

1.0.2 Pendahuluan

Pada tugas ini, kami melakukan analisis terhadap data transaksi dalam konteks *Data Mining Cup 2013*, yang berfokus pada pengembangan model klasifikasi untuk mengidentifikasi pola dari data transaksi. Data yang digunakan terdiri dari dua set: `transact_train.txt` sebagai data pelatihan untuk melatih model, dan `transact_class.txt` sebagai data klasifikasi untuk pengujian model. Setiap file data memiliki format khusus dengan pemisah `|` dan mengandung nilai hilang yang ditandai dengan simbol `?`, yang digantikan dengan nilai `NaN` agar lebih mudah diolah. Dengan menggunakan *library* Python seperti `pandas`, kami bertujuan untuk mengolah data ini secara efektif, melakukan pra-pemrosesan, dan membangun model yang mampu memberikan klasifikasi yang akurat.

1.1 Import dan Cleaning Dataset

[1]:
```
# import library
import pandas as pd
import numpy as np
```

[2]:
```
# training data
dt = pd.read_csv('transact_train.txt', sep='|', na_values='?')

# classification data
dc = pd.read_csv('transact_class.txt', sep='|', na_values='?')
```

Jadi, di kode pertama, saya membaca file transact_train.txt yang berisi data pelatihan. Data di dalam file ini dipisahkan dengan tanda | (bukan koma), dan saya juga mengganti semua tanda ? yang ada di dalamnya dengan NaN, agar bisa dianggap sebagai data yang hilang.

Di kode kedua, saya melakukan hal yang sama, cuma file yang saya baca adalah transact_class.txt, yang berisi data klasifikasi. Jadi intinya, kedua kode ini digunakan untuk membaca file teks dan mengganti tanda ? dengan nilai yang hilang.

[3]:
```
dt.head(10)
```

[3]:

	sessionNo	startHour	startWeekday	duration	cCount	cMinPrice	cMaxPrice	\
0	1	6	5	0.000	1	59.99	59.99	
1	1	6	5	11.940	1	59.99	59.99	
2	1	6	5	39.887	1	59.99	59.99	
3	2	6	5	0.000	0	NaN	NaN	
4	2	6	5	15.633	0	NaN	NaN	
5	2	6	5	26.235	0	NaN	NaN	
6	2	6	5	71.200	0	NaN	NaN	
7	2	6	5	94.469	0	NaN	NaN	
8	3	6	5	181.477	9	29.99	29.99	
9	3	6	5	297.018	11	9.99	29.99	

	cSumPrice	bCount	bMinPrice	…	availability	customerNo	\
0	59.99	1	59.99	…	NaN	1.0	
1	59.99	1	59.99	…	completely orderable	1.0	
2	59.99	1	59.99	…	completely orderable	1.0	
3	NaN	0	NaN	…	completely orderable	NaN	
4	NaN	0	NaN	…	completely orderable	NaN	
5	NaN	0	NaN	…	completely orderable	NaN	
6	NaN	0	NaN	…	completely orderable	NaN	
7	NaN	0	NaN	…	completely orderable	NaN	
8	89.97	1	29.99	…	NaN	3.0	
9	109.95	2	9.99	…	NaN	3.0	

	maxVal	customerScore	accountLifetime	payments	age	address	lastOrder	\
0	600.0	70.0	21.0	1.0	43.0	1.0	49.0	
1	600.0	70.0	21.0	1.0	43.0	1.0	49.0	
2	600.0	70.0	21.0	1.0	43.0	1.0	49.0	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
8	1800.0	475.0	302.0	12.0	45.0	1.0	11.0	
9	1800.0	475.0	302.0	12.0	45.0	1.0	11.0	

	order
0	y
1	y
2	y
3	y

```
4      y
5      y
6      y
7      y
8      y
9      y

[10 rows x 24 columns]
```

[4]:
```
dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 429013 entries, 0 to 429012
Data columns (total 24 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   sessionNo        429013 non-null  int64
 1   startHour        429013 non-null  int64
 2   startWeekday     429013 non-null  int64
 3   duration         429013 non-null  float64
 4   cCount           429013 non-null  int64
 5   cMinPrice        426248 non-null  float64
 6   cMaxPrice        426248 non-null  float64
 7   cSumPrice        426248 non-null  float64
 8   bCount           429013 non-null  int64
 9   bMinPrice        423883 non-null  float64
 10  bMaxPrice        423883 non-null  float64
 11  bSumPrice        423883 non-null  float64
 12  bStep            237680 non-null  float64
 13  onlineStatus     268634 non-null  object
 14  availability     263758 non-null  object
 15  customerNo       277915 non-null  float64
 16  maxVal           275273 non-null  float64
 17  customerScore    275273 non-null  float64
 18  accountLifetime  275273 non-null  float64
 19  payments         277915 non-null  float64
 20  age              277617 non-null  float64
 21  address          277915 non-null  float64
 22  lastOrder        277915 non-null  float64
 23  order            429013 non-null  object
dtypes: float64(16), int64(5), object(3)
memory usage: 78.6+ MB
```

[5]:
```
dc.head()
```

[5]:

	sessionNo	startHour	startWeekday	duration	cCount	cMinPrice	cMaxPrice	\
0	1	18	7	136.833	3	39.99	39.99	

```
1          1         18             7   189.984       3      39.99      39.99
2          1         18             7   342.894       6      16.99      39.99
3          1         18             7   411.051       8      16.99      39.99
4          1         18             7   460.049      10      16.99      39.99

   cSumPrice  bCount  bMinPrice  …  onlineStatus          availability  \
0      79.98       1      39.99  …             y  completely orderable
1      79.98       1      39.99  …             y  completely orderable
2     113.96       2      16.99  …           NaN                   NaN
3     149.94       3      16.99  …           NaN                   NaN
4     189.92       4      16.99  …           NaN                   NaN

   customerNo  maxVal customerScore  accountLifetime  payments   age  address  \
0     25039.0  1300.0         489.0            188.0       5.0  49.0      1.0
1     25039.0  1300.0         489.0            188.0       5.0  49.0      1.0
2     25039.0  1300.0         489.0            188.0       5.0  49.0      1.0
3     25039.0  1300.0         489.0            188.0       5.0  49.0      1.0
4     25039.0  1300.0         489.0            188.0       5.0  49.0      1.0

   lastOrder
0       65.0
1       65.0
2       65.0
3       65.0
4       65.0

[5 rows x 23 columns]
```

[6]:
```
dc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45068 entries, 0 to 45067
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   sessionNo        45068 non-null  int64
 1   startHour        45068 non-null  int64
 2   startWeekday     45068 non-null  int64
 3   duration         45068 non-null  float64
 4   cCount           45068 non-null  int64
 5   cMinPrice        44742 non-null  float64
 6   cMaxPrice        44742 non-null  float64
 7   cSumPrice        44742 non-null  float64
 8   bCount           45068 non-null  int64
 9   bMinPrice        44479 non-null  float64
 10  bMaxPrice        44479 non-null  float64
 11  bSumPrice        44479 non-null  float64
```

```
 12  bStep             24302 non-null  float64
 13  onlineStatus      27713 non-null  object
 14  availability      27311 non-null  object
 15  customerNo        27804 non-null  float64
 16  maxVal            27616 non-null  float64
 17  customerScore     27616 non-null  float64
 18  accountLifetime   27616 non-null  float64
 19  payments          27804 non-null  float64
 20  age               27786 non-null  float64
 21  address           27804 non-null  float64
 22  lastOrder         27804 non-null  float64
dtypes: float64(16), int64(5), object(2)
memory usage: 7.9+ MB
```

[7]:
```
# pembuangan atribut dan instance
dt = dt.drop(['age', 'address', 'customerNo', 'customerScore', 'maxVal'],␣
 ↪axis=1)
dc = dc.drop(['age', 'address', 'customerNo', 'customerScore', 'maxVal'],␣
 ↪axis=1)
```

Jadi, di kode ini, kami menghapus beberapa kolom dari data. Kolom yang kami buang adalah `'age'`, `'address'`, `'customerNo'`, `'customerScore'`, dan `'maxVal'`. Dengan kode ini, kami menghapus kolom-kolom yang kami anggap tidak diperlukan dari data pelatihan.

[8]:
```
dt.head(10)
```

[8]:

	sessionNo	startHour	startWeekday	duration	cCount	cMinPrice	cMaxPrice	\
0	1	6	5	0.000	1	59.99	59.99	
1	1	6	5	11.940	1	59.99	59.99	
2	1	6	5	39.887	1	59.99	59.99	
3	2	6	5	0.000	0	NaN	NaN	
4	2	6	5	15.633	0	NaN	NaN	
5	2	6	5	26.235	0	NaN	NaN	
6	2	6	5	71.200	0	NaN	NaN	
7	2	6	5	94.469	0	NaN	NaN	
8	3	6	5	181.477	9	29.99	29.99	
9	3	6	5	297.018	11	9.99	29.99	

	cSumPrice	bCount	bMinPrice	bMaxPrice	bSumPrice	bStep	onlineStatus	\
0	59.99	1	59.99	59.99	59.99	NaN	NaN	
1	59.99	1	59.99	59.99	59.99	2.0	y	
2	59.99	1	59.99	59.99	59.99	NaN	y	
3	NaN	0	NaN	NaN	NaN	2.0	y	
4	NaN	0	NaN	NaN	NaN	NaN	y	
5	NaN	0	NaN	NaN	NaN	4.0	y	
6	NaN	0	NaN	NaN	NaN	4.0	y	
7	NaN	0	NaN	NaN	NaN	NaN	y	
8	89.97	1	29.99	29.99	29.99	NaN	NaN	

```
9     109.95       2       9.99      29.99      39.98    NaN          NaN

            availability  accountLifetime  payments  lastOrder order
0                    NaN             21.0       1.0       49.0     y
1  completely orderable              21.0       1.0       49.0     y
2  completely orderable              21.0       1.0       49.0     y
3  completely orderable               NaN       NaN        NaN     y
4  completely orderable               NaN       NaN        NaN     y
5  completely orderable               NaN       NaN        NaN     y
6  completely orderable               NaN       NaN        NaN     y
7  completely orderable               NaN       NaN        NaN     y
8                    NaN           302.0      12.0       11.0     y
9                    NaN           302.0      12.0       11.0     y
```

1.1.1 Penanganan Data Kosong

[9]:
```python
def handle_missing_data(df):
    # Penanganan data kosong menjadi Mean
    df['cSumPrice'].fillna(df['cSumPrice'].mean(), inplace=True)
    df['bSumPrice'].fillna(df['bSumPrice'].mean(), inplace=True)

    # Penanganan data kosong menjadi Median
    df['cMinPrice'].fillna(df['cMinPrice'].median(), inplace=True)
    df['cMaxPrice'].fillna(df['cMaxPrice'].median(), inplace=True)
    df['bMinPrice'].fillna(df['bMinPrice'].median(), inplace=True)
    df['bMaxPrice'].fillna(df['bMaxPrice'].median(), inplace=True)
    df['accountLifetime'].fillna(df['accountLifetime'].median(), inplace=True)
    df['payments'].fillna(df['payments'].median(), inplace=True)
    df['lastOrder'].fillna(df['lastOrder'].median(), inplace=True)

    # Menangani data NaN di kolom 'onlineStatus' dengan pengisian acak
    df['bStep'] = df['bStep'].apply(lambda x: np.random.choice([1, 2, 3, 4, 5])␣
↪if pd.isna(x) else x)

    # Menangani data NaN di kolom 'onlineStatus' dengan pengisian acak antara␣
↪"y" dan "n"
    df['onlineStatus'] = df['onlineStatus'].apply(lambda x: np.random.
↪choice(['y', 'n']) if pd.isna(x) else x)

    # Menangani data NaN di kolom 'availability' dengan pengisian acak dari␣
↪beberapa opsi yang relevan
    df['availability'] = df['availability'].apply(lambda x: np.random.
↪choice(['completely orderable', 'completely not orderable',
                                                                    ␣
↪ 'mainly orderable', 'mixed',
```

```
    'mainly not orderable', 'completely not determinable',
    'mainly not determinable']) if pd.isna(x) else x)

    return df
```

[10]:
```
dt = handle_missing_data(dt)
dc = handle_missing_data(dc)
```

Di kode ini, kami menangani data yang hilang (missing data) dengan cara mengisinya menggunakan berbagai metode, tergantung jenis kolomnya:

1. **Mengisi dengan Mean (Rata-rata)**:
   - Untuk kolom `'cSumPrice'` dan `'bSumPrice'`, kami mengisi data yang hilang dengan rata-rata (mean) dari nilai-nilai yang ada di kolom tersebut.
2. **Mengisi dengan Median (Nilai Tengah)**:
   - Untuk kolom-kolom seperti `'cMinPrice'`, `'cMaxPrice'`, `'bMinPrice'`, `'bMaxPrice'`, `'accountLifetime'`, `'payments'`, dan `'lastOrder'`, kami mengisi nilai yang hilang dengan median, yaitu nilai tengah dari kolom tersebut.
3. **Mengisi dengan Random Choice**:
   - Kolom `'bStep'`,`'onlineStatus'`, dan `'availability'` diisi dengan random choice, yaitu dipilih secara random dari data yang ada pada kolom tersebut.

Jadi, secara keseluruhan, kode ini digunakan untuk mengisi nilai kosong di berbagai kolom dengan cara yang paling sesuai, agar data menjadi lengkap dan bisa digunakan untuk analisis lebih lanjut.

[11]:
```
dt.head(10)
```

[11]:

	sessionNo	startHour	startWeekday	duration	cCount	cMinPrice	cMaxPrice	\
0	1	6	5	0.000	1	59.99	59.99	
1	1	6	5	11.940	1	59.99	59.99	
2	1	6	5	39.887	1	59.99	59.99	
3	2	6	5	0.000	0	12.00	49.99	
4	2	6	5	15.633	0	12.00	49.99	
5	2	6	5	26.235	0	12.00	49.99	
6	2	6	5	71.200	0	12.00	49.99	
7	2	6	5	94.469	0	12.00	49.99	
8	3	6	5	181.477	9	29.99	29.99	
9	3	6	5	297.018	11	9.99	29.99	

	cSumPrice	bCount	bMinPrice	bMaxPrice	bSumPrice	bStep	onlineStatus	\
0	59.990000	1	59.99	59.99	59.990000	2.0	n	
1	59.990000	1	59.99	59.99	59.990000	2.0	y	
2	59.990000	1	59.99	59.99	59.990000	5.0	y	
3	1189.248209	0	14.99	39.99	213.260809	2.0	y	

```
4  1189.248209       0      14.99      39.99  213.260809    2.0           y
5  1189.248209       0      14.99      39.99  213.260809    4.0           y
6  1189.248209       0      14.99      39.99  213.260809    4.0           y
7  1189.248209       0      14.99      39.99  213.260809    4.0           y
8    89.970000       1      29.99      29.99   29.990000    2.0           n
9   109.950000       2       9.99      29.99   39.980000    1.0           y

           availability  accountLifetime  payments  lastOrder order
0                 mixed             21.0       1.0       49.0     y
1  completely orderable             21.0       1.0       49.0     y
2  completely orderable             21.0       1.0       49.0     y
3  completely orderable            109.0       8.0       34.0     y
4  completely orderable            109.0       8.0       34.0     y
5  completely orderable            109.0       8.0       34.0     y
6  completely orderable            109.0       8.0       34.0     y
7  completely orderable            109.0       8.0       34.0     y
8      mainly orderable            302.0      12.0       11.0     y
9  mainly not orderable            302.0      12.0       11.0     y
```

[12]:
```
dc.head(10)
```

[12]:
```
   sessionNo  startHour  startWeekday  duration  cCount  cMinPrice  cMaxPrice  \
0          1         18             7   136.833       3      39.99      39.99
1          1         18             7   189.984       3      39.99      39.99
2          1         18             7   342.894       6      16.99      39.99
3          1         18             7   411.051       8      16.99      39.99
4          1         18             7   460.049      10      16.99      39.99
5          1         18             7   471.502      10      16.99      39.99
6          1         18             7   560.026      11      16.99      39.99
7          1         18             7   564.597      11      16.99      39.99
8          1         18             7   624.606      11      16.99      39.99
9          2         18             7   133.321       7      34.99      34.99

   cSumPrice  bCount  bMinPrice  bMaxPrice  bSumPrice  bStep onlineStatus  \
0      79.98       1      39.99      39.99      39.99    2.0            y
1      79.98       1      39.99      39.99      39.99    2.0            y
2     113.96       2      16.99      39.99      56.98    1.0            n
3     149.94       3      16.99      39.99      74.97    3.0            n
4     189.92       4      16.99      39.99      94.96    4.0            y
5     189.92       4      16.99      39.99      94.96    1.0            y
6     207.91       5      16.99      39.99     112.95    3.0            n
7     207.91       5      16.99      39.99     112.95    1.0            y
8     207.91       5      16.99      39.99     112.95    4.0            y
9      69.98       1      34.99      34.99      34.99    5.0            y

                availability  accountLifetime  payments  lastOrder
0      completely orderable            188.0       5.0       65.0
```

```
1      completely orderable            188.0       5.0       65.0
2                     mixed            188.0       5.0       65.0
3  completely not orderable            188.0       5.0       65.0
4           mainly orderable            188.0       5.0       65.0
5      completely orderable            188.0       5.0       65.0
6      mainly not orderable            188.0       5.0       65.0
7      completely orderable            188.0       5.0       65.0
8      completely orderable            188.0       5.0       65.0
9      mainly not orderable             43.0       5.0      184.0
```

1.1.2 Grouping berdasarkan sessionNo dan Agregasi

[13]:
```
# Mengelompokkan data berdasarkan 'sessionNo' dengan menjaga semua kolom awal,␣
 ↪setiap kolom diagregasi sesuai dengan datanya
dt = dt.groupby('sessionNo').agg({
    'duration': 'sum',
    'cCount': 'sum',
    'cMinPrice': 'min',
    'cMaxPrice': 'max',
    'cSumPrice': 'sum',
    'bCount': 'sum',
    'bMinPrice': 'min',
    'bMaxPrice': 'max',
    'bSumPrice': 'sum',
    'bStep': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
    'onlineStatus': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
    'availability': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
    'startHour': 'first',
    'startWeekday': 'first',
    'accountLifetime': 'first',
    'payments': 'first',
    'lastOrder': 'first',
    'order': 'first'
}).reset_index()

dc = dc.groupby('sessionNo').agg({
    'duration': 'sum',
    'cCount': 'sum',
    'cMinPrice': 'min',
    'cMaxPrice': 'max',
    'cSumPrice': 'sum',
    'bCount': 'sum',
    'bMinPrice': 'min',
    'bMaxPrice': 'max',
    'bSumPrice': 'sum',
    'bStep': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
    'onlineStatus': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
```

```
    'availability': lambda x: x.mode()[0] if not x.mode().empty else np.nan,
    'startHour': 'first',
    'startWeekday': 'first',
    'accountLifetime': 'first',
    'payments': 'first',
    'lastOrder': 'first'
}).reset_index()
```

[14]:
```
dt.head(10)
```

[14]:

	sessionNo	duration	cCount	cMinPrice	cMaxPrice	cSumPrice	bCount	\
0	1	51.827	3	59.99	59.99	179.970000	3	
1	2	207.537	0	12.00	49.99	5946.241044	0	
2	3	1455.353	53	9.99	29.99	529.770000	9	
3	4	71.683	8	4.99	4.99	39.920000	2	
4	5	12310.330	215	12.99	179.95	4943.150000	26	
5	6	0.000	6	99.99	169.00	758.950000	6	
6	7	8216.285	149	3.00	40.00	1420.000000	29	
7	8	0.000	1	59.99	59.99	59.990000	1	
8	9	1858.220	12	499.99	499.99	5999.880000	6	
9	10	1126.983	22	5.00	24.99	282.110000	3	

	bMinPrice	bMaxPrice	bSumPrice	bStep	onlineStatus	\
0	59.99	59.99	179.970000	2.0	y	
1	14.99	39.99	1066.304047	4.0	y	
2	9.99	29.99	189.910000	1.0	y	
3	4.99	4.99	9.980000	1.0	n	
4	19.99	27.85	684.800000	4.0	y	
5	99.99	169.00	758.950000	4.0	n	
6	3.00	3.00	87.000000	4.0	y	
7	59.99	59.99	59.990000	3.0	n	
8	499.99	499.99	2999.940000	2.0	y	
9	12.00	19.99	43.990000	1.0	n	

	availability	startHour	startWeekday	accountLifetime	\
0	completely orderable	6	5	21.0	
1	completely orderable	6	5	109.0	
2	completely orderable	6	5	302.0	
3	completely not orderable	6	5	109.0	
4	completely orderable	6	5	18.0	
5	mainly not orderable	6	5	109.0	
6	completely orderable	6	5	35.0	
7	completely not orderable	6	5	3.0	
8	completely orderable	6	5	109.0	
9	completely not determinable	6	5	55.0	

	payments	lastOrder	order

```
0       1.0       49.0     y
1       8.0       34.0     y
2      12.0       11.0     y
3       8.0       34.0     n
4       1.0       40.0     y
5       8.0       34.0     n
6      10.0       10.0     y
7      10.0       57.0     n
8       8.0       34.0     y
9      23.0       24.0     n
```

[15]:
```
dc.head(10)
```

[15]:

	sessionNo	duration	cCount	cMinPrice	cMaxPrice	cSumPrice	bCount	\
0	1	3761.542	73	16.99	39.99	1427.43	30	
1	2	14046.089	113	34.99	34.99	1189.66	15	
2	3	107049.958	1815	7.99	59.95	49659.37	201	
3	4	16223.585	868	3.99	239.99	33385.44	16	
4	5	63.261	4	29.99	29.99	119.96	2	
5	6	9203.161	131	7.99	39.99	1160.18	6	
6	7	493.223	11	14.99	19.99	174.89	5	
7	8	3525.261	74	8.99	99.99	1219.54	3	
8	9	68.599	4	59.99	79.99	299.96	1	
9	10	81071.954	2300	5.00	39.99	14038.42	69	

	bMinPrice	bMaxPrice	bSumPrice	bStep	onlineStatus	\
0	16.99	39.99	740.70	1.0	y	
1	34.99	34.99	524.85	2.0	y	
2	12.49	39.95	5855.86	2.0	y	
3	9.99	14.99	179.84	1.0	y	
4	29.99	29.99	59.98	2.0	n	
5	7.99	10.99	60.94	2.0	y	
6	14.99	14.99	74.95	1.0	n	
7	12.99	14.99	42.97	2.0	n	
8	79.99	79.99	79.99	3.0	n	
9	7.99	19.99	843.31	2.0	y	

	availability	startHour	startWeekday	accountLifetime	\
0	completely orderable	18	7	188.0	
1	completely orderable	18	7	43.0	
2	completely orderable	18	7	17.0	
3	completely orderable	18	7	226.0	
4	completely orderable	18	7	39.0	
5	completely not determinable	18	7	352.0	
6	completely not orderable	18	7	102.0	
7	mainly orderable	18	7	102.0	
8	completely not orderable	18	7	73.0	

```
9         completely orderable         18              7            102.0
```

```
   payments  lastOrder
0       5.0       65.0
1       5.0      184.0
2       4.0      107.0
3      19.0       17.0
4       2.0      234.0
5       9.0       28.0
6       7.0       42.0
7       7.0       42.0
8      14.0        4.0
9       7.0       42.0
```

1.1.3 One-hot Encoding

[16]:
```
# one-hot encoding untuk tipe kategori
dt = pd.get_dummies(data=dt, columns=['onlineStatus','availability'])
dc = pd.get_dummies(data=dc, columns=['onlineStatus','availability'])
```

Di kode ini, kami melakukan one-hot encoding pada kolom-kolom kategori, yaitu 'onlineStatus' dan 'availability' yang hasilnya adalah DataFrame yang memiliki kolom-kolom baru yang menggantikan kolom kategori dengan representasi numerik biner.

[17]:
```
dt.shape
```

[17]:
```
(50000, 26)
```

[18]:
```
dc.shape
```

[18]:
```
(5111, 25)
```

1.2 Proses Pembuatan Model

1.2.1 Menggunakan Data Training

[19]:
```
# ubah atribut yang akan menjadi label menggunakan LabelEncoder
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(dt['order'])
Y = le.transform(dt['order'])
```

Di kode ini, kami menggunakan **LabelEncoder** dari pustaka `sklearn` untuk mengubah kolom `'order'`, yang berisi label kategorikal, menjadi angka.

- **`from sklearn import preprocessing`**: Mengimpor modul `preprocessing` dari `sklearn`, yang menyediakan berbagai alat untuk praproses data, termasuk `LabelEncoder`.

- **`le = preprocessing.LabelEncoder()`**: Membuat objek `LabelEncoder` yang akan digunakan untuk mengubah label menjadi angka.
- **`le.fit(dt['order'])`**: Fungsi ini digunakan untuk "melatih" `LabelEncoder` dengan data dari kolom `'order'`. Di sini, `LabelEncoder` mempelajari semua nilai unik yang ada di kolom tersebut.
- **`Y = le.transform(dt['order'])`**: Fungsi ini mengubah nilai kategorikal pada kolom `'order'` menjadi angka, berdasarkan label yang telah dipelajari sebelumnya. Hasilnya adalah array `Y`, yang berisi representasi numerik dari nilai-nilai di kolom `'order'`.

Dengan cara ini, kolom `'order'` yang sebelumnya berisi data kategori kini diubah menjadi data numerik yang bisa digunakan dalam model machine learning.

[20]:
```
# siapkan atribut training dengan membuang kelas label
X = dt.drop("order", axis = 1)
```

Di kode ini, kami mempersiapkan data untuk pelatihan (training) dengan membuang kolom kelas label `'order'`, yang sudah diubah menjadi numerik (dengan `LabelEncoder`).

- **`X = dt.drop("order", axis=1)`**: Fungsi **`drop()`** digunakan untuk menghapus kolom dari DataFrame.
  - **`"order"`**: Ini adalah nama kolom yang akan dihapus, yang berisi label kelas.
  - **`axis=1`**: Menyatakan bahwa kolom yang dihapus (bukan baris) adalah yang dimaksud.

Hasil dari **`X`** adalah DataFrame yang hanya berisi atribut (fitur) atau kolom input, tanpa kolom `'order'` yang menjadi label target.

Dengan kata lain, kami memisahkan atribut (fitur) yang digunakan untuk pelatihan (`X`) dan label kelas (`Y`, yang sudah diubah dengan `LabelEncoder`) untuk digunakan dalam model machine learning.

1.2.2 Split Dataset

[21]:
```
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=123)


# simpan nama kolom untuk keperluan prediksi nanti
# pickle adalah library untuk menyimpan file
import pickle
with(open('transact_train.pickle', 'wb')) as fp:
    pickle.dump(X_train.columns, fp)
```

Di kode ini, kami membagi data menjadi set pelatihan dan set pengujian, serta menyimpan nama kolom dari data pelatihan untuk keperluan prediksi di masa depan.

1. **Membagi data menjadi pelatihan dan pengujian**:
   - **`train_test_split(X, Y, test_size=0.2, random_state=123)`**: Fungsi ini digunakan untuk membagi dataset menjadi dua bagian: data pelatihan (`X_train`, `Y_train`) dan data pengujian (`X_test`, `Y_test`).

- `X` adalah fitur atau atribut, sedangkan `Y` adalah label yang ingin diprediksi.
- `test_size=0.2`: Ini berarti 20% dari data akan digunakan untuk pengujian, sementara 80% lainnya untuk pelatihan.
- `random_state=123`: Menetapkan nilai acak untuk memastikan pembagian data yang konsisten setiap kali kode dijalankan.

2. **Menyimpan nama kolom untuk prediksi**:

- `import pickle`: Mengimpor pustaka `pickle`, yang digunakan untuk menyimpan dan memuat objek Python ke dalam file, seperti nama kolom di sini.
- `with open('transact_train.pickle', 'wb') as fp:`: Membuka file bernama `'transact_train.pickle'` dalam mode tulis biner (`wb`).
- `pickle.dump(X_train.columns, fp)`: Menyimpan nama kolom dari data pelatihan (`X_train.columns`) ke dalam file `'transact_train.pickle'`. Nama kolom ini akan berguna nanti ketika kita perlu melakukan prediksi menggunakan model, agar kita bisa mengetahui kolom mana yang digunakan.

Dengan kode ini, kami memastikan bahwa data telah dibagi untuk pelatihan dan pengujian, serta nama kolom yang digunakan untuk pelatihan disimpan untuk referensi di masa depan.

1.2.3 Random Forest

[22]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
# tampilkan confusion matrix
from sklearn.metrics import confusion_matrix

clf = RandomForestClassifier(n_estimators=50, random_state=123)
clf.fit(X_train, Y_train)

# Make predictions on the test set
Y_pred = clf.predict(X_test)

# Save the trained model for later use
with open('random_forest_model.pickle', 'wb') as f:
    pickle.dump(clf, f)
```

Di kode ini, kami melatih model **RandomForestClassifier** untuk memprediksi kolom `'order'` (yang telah diubah menjadi nilai numerik) dan mengevaluasi kinerjanya.

1. **Mengimpor pustaka yang diperlukan**:

- `RandomForestClassifier` dari `sklearn.ensemble`: Digunakan untuk membuat model Random Forest, yang merupakan salah satu algoritma pembelajaran mesin yang populer untuk klasifikasi.
- `classification_report` dan `accuracy_score` dari `sklearn.metrics`: Digunakan untuk mengevaluasi kinerja model dengan melihat akurasi dan laporan klasifikasi.

2. **Membangun dan melatih model**:

- **`clf = RandomForestClassifier(n_estimators=50, random_state=123)`**: Membuat objek `RandomForestClassifier` dengan 50 pohon keputusan (estimators), yang berarti model akan membuat 50 pohon keputusan dalam proses pelatihan.
- **`clf.fit(X_train, Y_train)`**: Melatih model menggunakan data pelatihan (`X_train` dan `Y_train`).

3. **Memprediksi**:

   - **`Y_pred = clf.predict(X_test)`**: Menggunakan model yang telah dilatih untuk memprediksi label pada data uji (`X_test`).

1.2.4 Menampilkan Akurasi, Classification Report, Confusion Matrix, dan Fitur Penting

[23]:
```
# Calculate accuracy
acc = accuracy_score(Y_test, Y_pred)
print("Akurasi {}".format(acc))

# Print detailed classification report
print(classification_report(Y_test, Y_pred))

print(confusion_matrix(Y_test, Y_pred))

feature_importances = pd.DataFrame(clf.feature_importances_,index = X_train.
  ↪columns, columns=['importance']).sort_values('importance',ascending=False)
print(feature_importances)
```

```
Akurasi 0.8316
              precision    recall  f1-score   support

           0       0.89      0.79      0.84      5419
           1       0.78      0.88      0.83      4581

    accuracy                           0.83     10000
   macro avg       0.83      0.84      0.83     10000
weighted avg       0.84      0.83      0.83     10000

[[4271 1148]
 [ 536 4045]]
                                        importance
bCount                                    0.186349
availability_completely orderable         0.095447
duration                                  0.085836
bSumPrice                                 0.076482
cCount                                    0.067239
cSumPrice                                 0.051207
sessionNo                                 0.044632
accountLifetime                           0.042996
payments                                  0.038973
```

```
lastOrder                                    0.038226
bMinPrice                                    0.036700
cMaxPrice                                    0.036303
cMinPrice                                    0.035153
bMaxPrice                                    0.034449
startHour                                    0.031805
onlineStatus_n                               0.025890
bStep                                        0.025715
onlineStatus_y                               0.021424
startWeekday                                 0.008681
availability_completely not orderable        0.007909
availability_completely not determinable     0.006317
availability_mainly not determinable         0.000757
availability_mainly orderable                0.000668
availability_mixed                           0.000450
availability_mainly not orderable            0.000393
```

1.2.5 Hasil yang didapat

- Akurasi: 0.8311, yang berarti model berhasil memprediksi label dengan tingkat akurasi sekitar 83.1%. Ini menunjukkan bahwa model cukup baik dalam memprediksi kelas.
- Precision: Pada kelas 0, precision 0.89 atau 89% prediksi kelas 0 adalah benar. Sementara pada kelas 1, precision 0.78 atau 78% prediksi kelas 1 adalah benar.
- Recall: Pada kelas 0, recall 0.79 atau 79% data kelas 0 berhasil terdeteksi model. Sementara pada kelas 1, recall 0.88 atau 88% data kelas 1 berhasil terdeteksi model.
- F1-Score: Pada kelas 0 adalah 0.84 dan kelas 1 adalah 0.83
- Confusion Matrix: Model berhasil memprediksi kelas 1 dengan benar sebanyak 4029 prediksi dan 4282 prediksi untuk kelas 0. Sementara terdapat 1137 prediksi kelas 0 yang salah dan 552 prediksi kelas 1 yang salah.
- Feature Importance: bCount, availability_completely, orderable, duration adalah beberapa fitur yang memiliki pengaruh tertinggi pada model.

1.3 Prediksi Data Class Menggunakan Trained Model

1.3.1 Import Data Real Class dan Pisahkan Real Prediction

[24]:
```
rc = pd.read_csv('realclass_t1.txt', sep='|', na_values='?')
rc.info()


real_pred = rc['prediction'].values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5111 entries, 0 to 5110
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sessionNo   5111 non-null   int64
 1   prediction  5111 non-null   int64
```

```
dtypes: int64(2)
memory usage: 80.0 KB
```

1.3.2 Load Kembali Trained Model dan Lakukan Prediksi pada Data Class

[25]:
```
# Load the trained model from the pickle file
with open('random_forest_model.pickle', 'rb') as f:
    clf = pickle.load(f)

# Load the column names for features from the pickle file
with open('transact_train.pickle', 'rb') as fp:
    feature_columns = pickle.load(fp)

# Ensure dc (data_class) has the same columns as data_train (excluding the␣
 ↪'order' column)
# Assuming that dc has the same columns as data_train, except for 'order'
X_dc = dc[feature_columns]

# Predict using the model
Y_dc_pred = clf.predict(X_dc)
```

[26]:
```
Y_dc_pred
```

[26]:
```
array([1, 1, 1, …, 0, 0, 0])
```

1.3.3 Tampilkan Akurasi dan Confusion Matrix

[27]:
```
# Calculate accuracy
acc = accuracy_score(real_pred, Y_dc_pred)
print("Akurasi {}".format(acc))

# Print detailed classification report
print(classification_report(real_pred, Y_dc_pred))

# tampilkan confusion matrix
print(confusion_matrix(Y_test, Y_pred))
```

```
Akurasi 0.8215613382899628
              precision    recall  f1-score   support

           0       0.92      0.73      0.82      2786
           1       0.74      0.93      0.83      2325

    accuracy                           0.82      5111
   macro avg       0.83      0.83      0.82      5111
weighted avg       0.84      0.82      0.82      5111
```

```
[[4271 1148]
 [ 536 4045]]
```