گزارش تمرین عملی سری دوم داده کاوی

یاسمن گودرزی 9931100

در ابتدا تمرین از ما خواسته شده تا دیتا ست مربوط را لود کنیم . برای این کار قطعه کد زیر را می نویسیم که از کتابخانه pandas برای این کار استفاده میکنیم.

```
[3]: ## TODO
data_set = pd.read_excel(r"./worldcities.xlsx")
data_set
```

[3]:

	ville	ville_ascii	lat	lng	pays	iso2	iso3	admin_nom	capital	population	id
0	A Coruña	A Coruna	43.3667	-8.3833	Spain	ES	ESP	Galicia	minor	245468.0	1.724417e+09
1	A Yun Pa	A Yun Pa	13.3939	108.4408	Vietnam	VN	VNM	Gia Lai	minor	53720.0	1.704946e+09
2	Aabenraa	Aabenraa	55.0444	9.4181	Denmark	DK	DNK	Syddanmark	minor	16401.0	1.208000e+09
3	Aachen	Aachen	50.7756	6.0836	Germany	DE	DEU	North Rhine-Westphalia	minor	249070.0	1.276806e+09
4	Aadorf	Aadorf	47.4939	8.8975	Switzerland	CH	CHE	Thurgau	NaN	9036.0	1.756023e+09
...	...	...	...	...	...	...	...	...	...	...	...
44662	Żychlin	Zychlin	52.2453	19.6236	Poland	PL	POL	Łódzkie	NaN	9021.0	1.616509e+09
44663	Żyrardów	Zyrardow	52.0500	20.4333	Poland	PL	POL	Mazowieckie	minor	39374.0	1.616146e+09
44664	Zyryanka	Zyryanka	65.7360	150.8900	Russia	RU	RUS	Sakha (Yakutiya)	NaN	3627.0	1.643202e+09
44665	Zyryanovsk	Zyryanovsk	49.7453	84.2548	Kazakhstan	KZ	KAZ	NaN	minor	49658.0	1.398361e+09
44666	Żywiec	Zywiec	49.6892	19.2058	Poland	PL	POL	Śląskie	minor	30334.0	1.616870e+09

در مرحله بعدی از ما خواسته شده است تا یک ستون جدید با عنوان population_level اضافه کنیم. سپس مقدار دهی این ستون را با استفاده از مقدار ستونpopulation باید پر کنیم. برای این کار از متدquantile موجود در pandas استفاده میکنیم. در این متد با توجه به مقادیری که بهش به عنوان ورودی میدهیم محدود ها را به ما میدهد مثلا 0.25 چارک اول داده ها است. سپس در یک حلقه شرط ها را بررسی میکنیم و ستون جدید را پر میکنیم.

```
[4]: ## TODO
# add new column population_level with vlue low
data_set['population_level'] = 'Low'
# Finding boundaries
boundaries =data_set['population'].quantile([0.25, 0.5, 0.7 , 1])
for item in data_set.index:
    if  0 <=data_set['population'][item]< boundaries [0.25]:
        pass
    elif boundaries [0.25] <= data_set['population'][item] < boundaries [0.5]:
        data_set.loc[item, 'population_level']  = "Mid"
    elif boundaries [0.5] <= data_set['population'][item] < boundaries [0.7]:
        data_set.loc[item, 'population_level']  = "High"
    elif boundaries [0.7] <= data_set['population'][item]<= boundaries [1] :
       data_set.loc[item, 'population_level']  = "Over"
data_set
```

[4]:

	ville	ville_ascii	lat	lng	pays	iso2	iso3	admin_nom	capital	population	id	population_level
0	A Coruña	A Coruna	43.3667	-8.3833	Spain	ES	ESP	Galicia	minor	245468.0	1.724417e+09	Over
1	A Yun Pa	A Yun Pa	13.3939	108.4408	Vietnam	VN	VNM	Gia Lai	minor	53720.0	1.704946e+09	Over
2	Aabenraa	Aabenraa	55.0444	9.4181	Denmark	DK	DNK	Syddanmark	minor	16401.0	1.208000e+09	Mid
3	Aachen	Aachen	50.7756	6.0836	Germany	DE	DEU	North Rhine-Westphalia	minor	249070.0	1.276806e+09	Over
4	Aadorf	Aadorf	47.4939	8.8975	Switzerland	CH	CHE	Thurgau	NaN	9036.0	1.756023e+09	Low
...	...	...	...	...	...	...	...	...	...	...	...	...
44662	Żychlin	Zychlin	52.2453	19.6236	Poland	PL	POL	Łódzkie	NaN	9021.0	1.616509e+09	Low
44663	Żyrardów	Zyrardow	52.0500	20.4333	Poland	PL	POL	Mazowieckie	minor	39374.0	1.616146e+09	Over
44664	Zyryanka	Zyryanka	65.7360	150.8900	Russia	RU	RUS	Sakha (Yakutiya)	NaN	3627.0	1.643202e+09	Low
44665	Zyryanovsk	Zyryanovsk	49.7453	84.2548	Kazakhstan	KZ	KAZ	NaN	minor	49658.0	1.398361e+09	Over
44666	Żywiec	Zywiec	49.6892	19.2058	Poland	PL	POL	Śląskie	minor	30334.0	1.616870e+09	High

44667 rows × 12 columns

در مرحله بعدی از ما خواسته شده تا سطر های که یک ویژگی خالی دارند را حذف کنیم. در واقع از ما خواسته شده تا دیتا کلینینگ انجام دهیم این کار را با متد dropna انجام میدهیم . در مرحله بعد از ما خواسته شده تا دیتا های حروفی را به عددی تبدیل کنیم . برای اینکار میتوان از الگوریتم labeling method استفاده کرد که در این الگوریتم ما به هر مقدار موجود در دیتاست مون برای یک ویژگی یک عدد منحصر به فرد را اختصاص می دهیم . مثلا اگر فیچر ما شامل سه مقدار A,B,C باشد به هر کدام عدد A=1,B=2,C=3 را نسبت می دهیم و به این صورت دیتا های حرفی را به عددی تبدیل کردیم.

برای این کار در پایتون از کتابخانه sklearn استفاده می کنیم . در این کتابخانه یک فایل preprocessing که یک کلاس label Encoder دارد . این کلاس شامل متد fit_transform است که الگوریتم را برای ما روی دیتا اجرا میکند. پس برای فیلد های حرفی این کلاس و متد را صدا میزنیم.

[9]:
```
# TODO
# drop empty rows
data_set.dropna(inplace=True)

label_encoder = preprocessing.LabelEncoder()
data_set['ville']= label_encoder.fit_transform(data_set['ville'])
data_set['ville_ascii']= label_encoder.fit_transform(data_set['ville_ascii'])
data_set['pays']= label_encoder.fit_transform(data_set['pays'])
data_set['iso2']= label_encoder.fit_transform(data_set['iso2'])
data_set['iso3']= label_encoder.fit_transform(data_set['iso3'])
data_set['admin_nom']= label_encoder.fit_transform(data_set['admin_nom'])
data_set['capital']= label_encoder.fit_transform(data_set['capital'])
data_set
```

[9]:

	ville	ville_ascii	lat	lng	pays	iso2	iso3	admin_nom	capital	population	id	population_level
0	0	0	43.3667	-8.3833	163	54	54	936	1	245468.0	1.724417e+09	Over
1	1	1	13.3939	108.4408	193	189	189	966	1	53720.0	1.704946e+09	Over
2	2	2	55.0444	9.4181	47	46	48	2866	1	16401.0	1.208000e+09	Mid
3	3	3	50.7756	6.0836	66	44	45	2109	1	249070.0	1.276806e+09	Over
5	4	4	57.0500	9.9167	47	46	48	2096	0	143598.0	1.208789e+09	Over
...	...	...	...	...	...	...	...	...	...	...	...	...
44656	10097	10357	50.7167	12.5000	66	44	45	2665	1	89540.0	1.276684e+09	Over
44658	10098	10358	51.8167	4.6500	121	130	129	3343	1	44775.0	1.528524e+09	Over
44660	10099	10359	52.5167	6.1000	121	130	129	2233	0	129840.0	1.528690e+09	Over
44663	10353	10360	52.0500	20.4333	137	141	140	1850	1	39374.0	1.616146e+09	Over
44666	10354	10361	49.6892	19.2058	137	141	140	3396	1	30334.0	1.616870e+09	High

در مرحله بعد از ما خواسته شده تا برای ویژگی های قسمت که قبل که حرف را به عدد تبدیل کردیم، نرمال سازی انجام دهیم. میدانیم که نرمال سازی یعنی هر مقدار را منهای میانگین کل و تقسیم برا انحراف معیار کنیم. برای اینکار ویژگی های که در مرحله قبلی داشتیم را در یک لیست مینویسیم و بعد با یک حلقه for هر دفعه میانگین و انحراف معیار را به دست آورده و بعد از انجام عملیات ریاضی به عنوان مقدار جدید در ستون اضافه میکنیم.

[10]:
```
# TODO
item_list = ['ville', 'ville_ascii', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital' ]
for item  in item_list:
    mean = data_set[item].mean()
    std = data_set[item].std()
    data_set[item] = (data_set[item] - mean) / std
data_set
```

[10]:

	ville	ville_ascii	lat	lng	pays	iso2	iso3	admin_nom	capital	population	id	population_level
0	-1.738599	-1.738571	43.3667	-8.3833	1.135248	-0.819122	-0.799908	-0.775360	0.585217	245468.0	1.724417e+09	Over
1	-1.738265	-1.738236	13.3939	108.4408	1.671727	1.591842	1.595013	-0.745788	0.585217	53720.0	1.704946e+09	Over
2	-1.737931	-1.737902	55.0444	9.4181	-0.939138	-0.961994	-0.906349	1.127115	0.585217	16401.0	1.208000e+09	Mid
3	-1.737597	-1.737568	50.7756	6.0836	-0.599368	-0.997712	-0.959569	0.380911	0.585217	249070.0	1.276806e+09	Over
5	-1.737262	-1.737233	57.0500	9.9167	-0.939138	-0.961994	-0.906349	0.368096	-1.436681	143598.0	1.208789e+09	Over
...	...	...	...	...	...	...	...	...	...	...	...	...
44656	1.636043	1.724102	50.7167	12.5000	-0.599368	-0.997712	-0.959569	0.928981	0.585217	89540.0	1.276684e+09	Over
44658	1.636378	1.724436	51.8167	4.6500	0.384177	0.538161	0.530604	1.597312	0.585217	44775.0	1.528524e+09	Over
44660	1.636712	1.724771	52.5167	6.1000	0.384177	0.538161	0.530604	0.503143	-1.436681	129840.0	1.528690e+09	Over
44663	1.721604	1.725105	52.0500	20.4333	0.670299	0.734610	0.725745	0.125605	0.585217	39374.0	1.616146e+09	Over
44666	1.721938	1.725439	49.6892	19.2058	0.670299	0.734610	0.725745	1.649556	0.585217	30334.0	1.616870e+09	High

10738 rows × 12 columns

در مرحله بعد از ما خواسته شده تا داده های تست و آموزش را به نسبت 20 به 80 جدا کنیم. برا این کار از متد train_test_split استفاده می کنیم.در این متد سایز دیتا ست تست مون رو 0.2 میدهیم تا 20 درصد داده ها برای تست در نظر بگیرد (با ورودی test_size =0.2) و همچنین برای اینکه رندوم داده ها پخش شود و اطمینان از تکرار پذیری از random_state =42 استفاده می کنیم.

18]:
```
# TODO
train, test = model_selection.train_test_split(data_set, test_size=0.2, random_state=42)
print('data training:', train)
print('data testing:', test)
```

```
data training:           ville  ville_ascii      lat       lng      pays      iso2      iso3  \
32234  0.714258     0.692355  36.9031   50.6583 -0.331129 -0.301211 -0.303183
40366  1.280431     1.364027  58.0333   14.9667  1.224661  0.984636  1.169249
11948 -0.888004    -0.852926 -26.4000  -54.6333 -1.672327 -1.658495 -1.669175
39157  1.163787     1.246342  18.2500  -94.7667  0.223233  0.413148  0.211281
34048  0.761384     0.821072  41.2903   36.3336  1.457136  1.377534  1.399871
...         ...          ...      ...       ...       ...       ...       ...
24500  0.071214     0.124325  21.4225   39.8233  0.920656  0.913200  0.903147
22024 -0.105255    -0.052536  24.8597  -99.5647  0.223233  0.413148  0.211281
22773 -0.044761     0.012324  -0.4069   31.1575  1.510783  1.484688  1.470832
3695  -1.475232    -1.457398  16.0730  102.7362  1.331957  1.270380  1.275690
31456  0.562521     0.624820  15.2653  -83.7744 -0.420542 -0.461942 -0.427364

       admin_nom   capital  population            id population_level
32234   0.258679  0.585217     35997.0  1.364587e+09             High
40366  -0.434295  0.585217     14197.0  1.752722e+09              Mid
11948   0.170949  0.585217     57323.0  1.032056e+09             Over
39157   1.404107  0.585217     15614.0  1.484403e+09              Mid
34048   0.865894 -1.436681   1335716.0  1.792169e+09             Over
...          ...       ...         ...           ...              ...
24500   0.033931 -1.436681   1675368.0  1.682169e+09             Over
22024   0.418369  0.585217     57731.0  1.484505e+09             Over
22773  -0.011413 -1.436681     16300.0  1.800897e+09              Mid
3695   -0.302206  0.585217     28913.0  1.764027e+09             High
31456  -0.714244 -1.436681     47528.0  1.340110e+09             Over

[8590 rows x 12 columns]
data testing:           ville  ville_ascii      lat       lng      pays      iso2      iso3  \
29807  0.435517     0.493093  39.9892   66.8458  1.618079  1.538265  1.541792
16828 -0.518688    -0.477138  20.0767  -74.6519 -1.010669 -1.069148 -1.012789
25855  0.152096     0.211251  48.3667   41.8333  0.759713  0.877482  0.867666
44202  1.584907     1.665594  49.3000   19.9500  0.670299  0.734610  0.725745
36689  1.116662     1.033038  37.1528   49.8708 -0.331129 -0.301211 -0.303183
...         ...          ...      ...       ...       ...       ...       ...
1253  -1.632317    -1.623226  18.6000  -99.3700  0.223233  0.413148  0.211281
1943  -1.601234    -1.589459  30.5000  117.0333 -1.135848 -1.122725 -1.172451
37195  1.000018     1.077839  -8.1324  113.9836 -0.349011 -0.390506 -0.356404
783   -1.673092    -1.662343  31.0158   47.4306 -0.313246 -0.319070 -0.285443
14871 -0.665746    -0.636279  22.8475  -82.0236 -1.010669 -1.069148 -1.012789
```

حالا که داده هامون مشخص شده است میخواهیم مدلمون رو پیاده سازی کنیم.در مرحله اول از ما خواسته شده تا رگرسیون درجه اول و دوم را ابتدا برای داده های اموزش اجرا کنیم و سپس روی داده های تست آن را اجرا کنیم و دقت را در هر مرحله محاسبه کنیم.

میدانیم که در رگرسیون همه ی فیچر های ما باید مقدار عددی داشته باشند. پس در مرحله اول با استفاده از الگوریتم **label encode** که در مرحله های قبلی استفاده کردیم مقدار ویژگی **population level** را عددی می کنیم.

دیتا های اموزش مان را به دیتا های ورودی و خروجی تقسیم میکنیم. **y_train** خروجی واقعی داده های اموزشمون و **x_train** ورودی مدلمون است . سپس دیتا ها را به مدلمون فیت میکنیم و بعد با متد **predict** مدلمون \ایش بینی میکنیم روی داده ها این روند را برای همه مدل ها طی میکنیم.

سپس با استفاده از متد های موجود درپایتون دقت را محاسبه میکنیم.

```python
# TODO
from sklearn import preprocessing , linear_model
from sklearn.metrics import r2_score , mean_squared_error

label_encoder = preprocessing.LabelEncoder()
train['population_level']= label_encoder.fit_transform(train['population_level'])
test['population_level']= label_encoder.fit_transform(test['population_level'])

x_train = train[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id','population_level']]
y_train = train['population']
x_test= test[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id','population_level']]
y_test = test['population']

linear_model = linear_model.LinearRegression()
linear_model.fit(x_train, y_train)
predict_result = linear_model.predict(x_train)

accuracy = r2_score(y_train, predict_result)
mse = mean_squared_error(y_train, predict_result)

print(f"Train accuracy : {accuracy}%")
print(f"Train  MSE: {mse}")

predict_result = linear_model.predict(x_test)

accuracy = r2_score(y_test, predict_result)
mse = mean_squared_error(y_test, predict_result)

print(f"Test accuracy : {accuracy}%")
print(f"Test  MSE: {mse}")
```

در مرحله بعد همین روند را برای رگرسیون درجه دو اجرا میکنیم. برای اینکار دیتا ورودی را باید در درجه 2 فیت کنیم و بعد مراحل قبلی را طی کنیم.

```
# create a polynomial regression with degree 2

x_train_poly = preprocessing.PolynomialFeatures(degree=2).fit_transform(x_train)

linear_model.fit(x_train_poly, y_train)
predict_result = linear_model.predict(x_train_poly)

accuracy = r2_score(y_train, predict_result)
mse = mean_squared_error(y_train, predict_result)

print(f"Train polynomial regression accuracy : {accuracy}%")
print(f"Train polynomial regression  MSE: {mse}")

x_test_poly = preprocessing.PolynomialFeatures(degree=2).fit_transform(x_test)
predict_result = linear_model.predict(x_test_poly)

accuracy = r2_score(y_test, predict_result)
mse = mean_squared_error(y_test, predict_result)

print(f"Test polynomial regression accuracy : {accuracy}%")
print(f"Test polynomial regression MSE: {mse}")
```

نتیجه به صورت زیر است:

```
Train accuracy : 0.058153613542916927%
Train  MSE: 1455344861977.0251
Test accuracy : 0.058008767410671624%
Test  MSE: 2744751322240.912
Train polynomial regression accuracy : 0.062066942750408916%
Train polynomial regression  MSE: 1449297969790.3154
Test polynomial regression accuracy : 0.0647161061462933%
Test polynomial regression MSE: 2725207640488.471
```

با مقایسه رگرسیون درجه اول و دوم میبینیم که دقت در رگرسیون درجه دوم بیشتر است . از طرفی خطا هم کاهش یافته است . پس میتوان نتیجه گرفت که رگرسیون درجه دوم بهتر از درجه اول است.

در قسمت بعدی از ما خواسته شده است تا اینبار **population level** را برچسب کلاس در نظر بگیریم. و بقیه فیچر ها را به عنوان ورودی در نظر بگیریم. برا این کار دیتا ها را مرتبط می کنیم(فیچر **population** را در نظر نمیگیریم)

```
x_train = train[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_train = train['population_level']
x_test= test[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_test = test['population_level']
```

حالا مدل های خواسته شده را با استفاده از کتابخانه های موجود در **sklearn** پیاده سازی میکنیم.

decision tree

Decision Tree (Entropy)

```
# TODO
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

x_train = train[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_train = train['population_level']
x_test= test[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_test = test['population_level']

decision_tree_classifier = DecisionTreeClassifier(criterion='entropy')
decision_tree_classifier.fit(x_train, y_train)
predict_result = decision_tree_classifier.predict(x_train)

report = classification_report(y_train, predict_result, output_dict=True)
report
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")

predict_result = decision_tree_classifier.predict(x_test)

report = classification_report(y_test, predict_result, output_dict=True)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 1.0
Train decision tree recall : 1.0
Train decision tree support : 8590.0
Train decision tree f1-score : 1.0
Test decision tree accuracy : 0.5083798882681564
Test decision tree recall : 0.5083798882681564
Test decision tree support : 2148.0
Test decision tree f1-score : 0.5086949175467569
```

Random forest

در این متد مثل رگرسیون نیاز است تا مقدار همه ویژگی ها عددی باشد

Random Forest (Entropy)

]:
```
# TODO
from sklearn.ensemble import RandomForestClassifier
label_encoder = preprocessing.LabelEncoder()
train['population_level']= label_encoder.fit_transform(train['population_level'])
test['population_level']= label_encoder.fit_transform(test['population_level'])

x_train = train[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_train = train['population_level']
x_test= test[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_test = test['population_level']


random_forest_classifier = RandomForestClassifier(criterion='entropy')
random_forest_classifier.fit(x_train, y_train)
predict_result = random_forest_classifier.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = random_forest_classifier.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 1.0
Train decision tree recall : 1.0
Train decision tree support : 8590.0
Train decision tree f1-score : 1.0
Test decision tree accuracy : 0.5498137802607076
Test decision tree recall : 0.5498137802607076
Test decision tree support : 2148.0
Test decision tree f1-score : 0.5334443483436968
```

KNN K=2

141]:
```
# TODO
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
x_train = train[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_train = train['population_level']
x_test= test[['ville', 'ville_ascii','lat','lng', 'pays', 'iso2', 'iso3', 'admin_nom' , 'capital','id']]
y_test = test['population_level']
K_neighbors_classifier = KNeighborsClassifier(n_neighbors=2)
K_neighbors_classifier.fit(x_train, y_train)
predict_result = K_neighbors_classifier.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = K_neighbors_classifier.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 0.7173457508731083
Train decision tree recall : 0.7173457508731083
Train decision tree support : 8590.0
Train decision tree f1-score : 0.7198719069668124
Test decision tree accuracy : 0.38221601489757917
Test decision tree recall : 0.38221601489757917
Test decision tree support : 2148.0
Test decision tree f1-score : 0.41068540150608646
```

KNN K=3

|:
```
# TODO
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
K_neighbors_classifier = KNeighborsClassifier(n_neighbors=3)
K_neighbors_classifier.fit(x_train, y_train)
predict_result = K_neighbors_classifier.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = K_neighbors_classifier.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 0.6757857974388825
Train decision tree recall : 0.6757857974388825
Train decision tree support : 8590.0
Train decision tree f1-score : 0.6763036369210289
Test decision tree accuracy : 0.4557728119180633
Test decision tree recall : 0.4557728119180633
Test decision tree support : 2148.0
Test decision tree f1-score : 0.4601137794846133
```

KNN K=5

43]:
```
# TODO
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
K_neighbors_classifier = KNeighborsClassifier(n_neighbors=5)
K_neighbors_classifier.fit(x_train, y_train)
predict_result = K_neighbors_classifier.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = K_neighbors_classifier.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 0.6321303841676368
Train decision tree recall : 0.6321303841676368
Train decision tree support : 8590.0
Train decision tree f1-score : 0.6306392456540761
Test decision tree accuracy : 0.4743947858472998
Test decision tree recall : 0.4743947858472998
Test decision tree support : 2148.0
Test decision tree f1-score : 0.4727971038241092
```

SVM (Linear)

48]:
```
# TODO
from sklearn.svm import SVC
from sklearn.metrics import classification_report
svm = SVC()
svm.fit(x_train, y_train)
predict_result = svm.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True, zero_division=1)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = svm.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True, zero_division=1)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 0.47939464493597206
Train decision tree recall : 0.47939464493597206
Train decision tree support : 8590.0
Train decision tree f1-score : 0.31069360211619973
Test decision tree accuracy : 0.49068901303538176
Test decision tree recall : 0.49068901303538176
Test decision tree support : 2148.0
Test decision tree f1-score : 0.3230394876572719
```

SVM (Non-linear)

```
# TODO
from sklearn import svm
from sklearn.metrics import classification_report
no_linear_svm = svm.NuSVC(gamma="auto")
no_linear_svm.fit(x_train, y_train)
predict_result = no_linear_svm.predict(x_train)
report = classification_report(y_train, predict_result, output_dict=True, zero_division=1)
print(f"Train decision tree accuracy : {report['accuracy']}")
print(f"Train decision tree recall : {report['weighted avg']['recall']}")
print(f"Train decision tree support : {report['weighted avg']['support']}")
print(f"Train decision tree f1-score : {report['weighted avg']['f1-score']}")
predict_result = no_linear_svm.predict(x_test)
report = classification_report(y_test, predict_result, output_dict=True, zero_division=1)
print(f"Test decision tree accuracy : {report['accuracy']}")
print(f"Test decision tree recall : {report['weighted avg']['recall']}")
print(f"Test decision tree support : {report['weighted avg']['support']}")
print(f"Test decision tree f1-score : {report['weighted avg']['f1-score']}")
```

```
Train decision tree accuracy : 0.99930151338766
Train decision tree recall : 0.99930151338766
Train decision tree support : 8590.0
Train decision tree f1-score : 0.9993014734312083
Test decision tree accuracy : 0.49068901303538176
Test decision tree recall : 0.49068901303538176
Test decision tree support : 2148.0
Test decision tree f1-score : 0.4777027895728581
```

با مشاهده خروجی ها میتوان مدل ها را از لحاظ دقت به صورت زیر دسته بندی کرد:

random forest > decision tree > svm = svm(no linear) > knn=2 > knn=3 > knn=5

از لحاظ خطا:

random forest > decision tree > svm(no linear) > knn=5 > knn=3 > knn=2 > svm