TestExam-Live 2024-11-11

November 11, 2024

1 Test Exam Live 2024-11-11

2 Question 3

```
[3]: import pandas as pd
```

```
[4]: X = pd.read_csv("Data01.csv")
```

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

```
[5]:    Unnamed: 0        f1        f2        f3        f4  target
     0           0  2.470518  2.094533  7.464342 -0.898171       1
     1           1 -1.290985  3.834506  3.713694  0.829274       0
     2           2  1.131218  4.681950  6.135816  1.703968      -1
     3           3  2.711527  1.489895  7.714867 -1.508245       1
     4           4  2.442849  5.156339  7.453527  2.162256      -1
```

2.1 Compute Yield

Good == target 0 Bad else

```
[6]: X.target.value_counts()
```

```
[6]: target
      0    6000
      1    3000
     -1    1000
     Name: count, dtype: int64
```

```
[9]: my_yield = 6000/(6000+3000+1000)
```

```
[10]: print(my_yield)
```

```
0.6
```

2.2 Q3a

[11]:

```
X.describe()
```

[11]:

```
        Unnamed: 0            f1            f2            f3            f4  \
count  10000.00000  10000.000000  10000.000000  10000.000000  10000.000000
mean    4999.50000     -0.213018      2.453336      4.787027     -0.546665
std     2886.89568      1.921910      1.475879      1.922073      1.476047
min        0.00000     -5.973536     -2.494636     -0.961878     -5.498820
25%     2499.75000     -1.741135      1.424400      3.259478     -1.574793
50%     4999.50000     -0.613317      2.546318      4.389423     -0.454807
75%     7499.25000      1.426110      3.489117      6.424810      0.490143
max     9999.00000      5.856928      7.063142     10.852296      4.071513

            target
count  10000.00000
mean       0.20000
std        0.60003
min       -1.00000
25%        0.00000
50%        0.00000
75%        1.00000
max        1.00000
```

Result min =

[14]:

```
for f in X.columns:
    print(f)
    print(X[f].min())
    print(X[f].max())
    print(X[f].mean())
    print()
```

```
Unnamed: 0
0
9999
4999.5

f1
-5.9735362545812
5.856927736399993
-0.2130181506933142

f2
-2.494636333484439
7.063142441469973
2.453336311133327
```

```
f3
-0.961878024767998
10.852295817742014
4.7870270709980645

f4
-5.49882016533299
4.071512820262887
-0.5466648786352191

target
-1
1
0.2
```

```
[15]: X.min()
```

```
[15]: Unnamed: 0    0.000000
      f1           -5.973536
      f2           -2.494636
      f3           -0.961878
      f4           -5.498820
      target       -1.000000
      dtype: float64
```

```
[16]: X.max()
```

```
[16]: Unnamed: 0    9999.000000
      f1               5.856928
      f2               7.063142
      f3              10.852296
      f4               4.071513
      target           1.000000
      dtype: float64
```

```
[17]: X.mean()
```

```
[17]: Unnamed: 0    4999.500000
      f1              -0.213018
      f2               2.453336
      f3               4.787027
      f4              -0.546665
      target           0.200000
      dtype: float64
```

Response: We need to scale! Features have very different value ranges.

3 Q3.b

```
[18]: from sklearn.preprocessing import MinMaxScaler
```

```
[22]: scaler = MinMaxScaler(feature_range=(0,1))
```

```
[24]: scaler.fit(X)
      # will also scale targets. but we will ignore scaled targets :-)
```

```
[24]: MinMaxScaler()
```

```
[30]: X_scaled = pd.DataFrame(scaler.transform(X),columns=X.columns)
```

```
[31]: X_scaled.describe()
```

```
[31]:         Unnamed: 0            f1            f2            f3            f4  \
      count  10000.000000  10000.000000  10000.000000  10000.000000  10000.000000
      mean       0.500000      0.486922      0.517691      0.486611      0.517449
      std        0.288718      0.162454      0.154417      0.162692      0.154231
      min        0.000000      0.000000      0.000000      0.000000      0.000000
      25%        0.250000      0.357754      0.410036      0.357313      0.410020
      50%        0.500000      0.453086      0.527419      0.452956      0.527047
      75%        0.750000      0.625474      0.626061      0.625240      0.625784
      max        1.000000      1.000000      1.000000      1.000000      1.000000

                   target
      count  10000.000000
      mean       0.600000
      std        0.300015
      min        0.000000
      25%        0.500000
      50%        0.500000
      75%        1.000000
      max        1.000000
```

3.1 Result

min ….

3.2 Q3e

```
[33]: # stupid predictor always predicts most frequent class

      y_stupid = X.target - X.target
```

```
[36]: from sklearn.metrics import f1_score
```

```python
baseline = f1_score(X.target,y_stupid,average='micro')
print(baseline)
```

```
0.6
```

[ ]:

```python

```