머신러닝 리포트 - 분류1

> 로지스틱 회귀(이진 분류, 2개의 카테고리)

목차

1. 시나리오

- 파형의 특성에 따라 라벨을 적용할 수 있을 때 머신러닝 모델을 이용하여 분류를 할 수 있다.
- 예제 데이터는 엘리베이터 UP/DOWN 라벨이 있는 데이터를 이용하여 분류를 실행 한다.

2. 목표

- 기간과 아이템을 선택하고 데이터를 파형별로 구분하고 통계값을 계산한다.
- 모델에 적용하기 위해 원하는 형태로 전처리 한다.
- 정규 분포 형태의 표준스케일링 적용 후 학습 / 테스트 데이터로 나눈다.
- 로지스틱 회귀 모델로 학습 및 예측을 수행한다.
- 정확도와 ROC-AUC 값 구한다.
- 하이퍼파라미터 최적화를 수행한다.

3. 프로세스

3-1. 흐름도

3-2. Load Data

- Select Period & Item

1. Period: 2020-05-01 ~ 2020-05-14
2. Item: ElevatorMain MaxCurrent

- Indexing & Extraction Statistics

1. Indexing

- EndCount: 2초(파형의 시작과 종료를 구분하기 위한 시간)

2. Extraction Statistics(17개)

- max
- average
- area
- median
- var
- std
- skew
- kurtosis
- Q1
- Q3
- iqr
- percentile10
- percentile40
- percentile60
- percentile90
- trim_mean10
- trim_mean20

Index / Statistics

index_date	index_num	defServer	defTable	defColumn	startTime	stopTime	length	timeTaken	basicFeatures
2020-05-14	264	S1	HisItemCurr	Item005	2020-05-14 21:56:52	2020-05-14 21:57:08	165	00:00:16	{"max": 72.95, "average": 30.91, "area": 5100.17, "median": 30.0, "var": 187.61, "std": 13.7, "skew": 0.12, "kurtosis": 1...
2020-05-14	264	S1	HisItemCurr	Item039	2020-05-14 21:56:52	2020-05-14 21:57:07	147	00:00:14	{"max": 19.91, "average": 7.04, "area": 1035.12, "median": 7.59, "var": 15.47, "std": 3.93, "skew": 0.89, "kurtosis": 1.5...
2020-05-14	263	S1	HisItemCurr	Item005	2020-05-14 21:47:44	2020-05-14 21:47:59	157	00:00:15	{"max": 55.57, "average": 28.83, "area": 4525.8, "median": 25.86, "var": 147.61, "std": 12.15, "skew": -0.04, "kurtosis":...
2020-05-14	263	S1	HisItemCurr	Item039	2020-05-14 21:47:44	2020-05-14 21:47:58	145	00:00:14	{"max": 13.93, "average": 3.27, "area": 474.83, "median": 2.25, "var": 7.82, "std": 2.8, "skew": 2.54, "kurtosis": 5.3, "Q...
2020-05-14	262	S1	HisItemCurr	Item039	2020-05-14 21:31:34	2020-05-14 21:31:49	147	00:00:14	{"max": 17.47, "average": 5.56, "area": 817.68, "median": 5.48, "var": 11.55, "std": 3.4, "skew": 1.65, "kurtosis": 3.14, ...
2020-05-14	262	S1	HisItemCurr	Item005	2020-05-14 21:31:33	2020-05-14 21:31:49	160	00:00:15	{"max": 66.07, "average": 30.18, "area": 4828.38, "median": 27.76, "var": 164.8, "std": 12.84, "skew": 0.08, "kurtosis": ...
2020-05-14	261	S1	HisItemCurr	Item005	2020-05-14 21:31:10	2020-05-14 21:31:26	165	00:00:16	{"max": 79.38, "average": 32.96, "area": 5437.67, "median": 32.19, "var": 200.9, "std": 14.17, "skew": 0.03, "kurtosis": ...
2020-05-14	261	S1	HisItemCurr	Item039	2020-05-14 21:31:10	2020-05-14 21:31:25	156	00:00:15	{"max": 21.28, "average": 8.11, "area": 1265.69, "median": 9.7, "var": 21.74, "std": 4.66, "skew": 0.26, "kurtosis": 0.14,...
2020-05-14	260	S1	HisItemCurr	Item039	2020-05-14 21:03:12	2020-05-14 21:03:38	257	00:00:25	{"max": 12.99, "average": 2.47, "area": 634.74, "median": 1.84, "var": 4.41, "std": 2.1, "skew": 3.66, "kurtosis": 12.6, "...
2020-05-14	260	S1	HisItemCurr	Item005	2020-05-14 21:03:11	2020-05-14 21:03:38	269	00:00:26	{"max": 56.89, "average": 27.44, "area": 7380.33, "median": 25.54, "var": 94.26, "std": 9.71, "skew": 0.4, "kurtosis": 2....
2020-05-14	259	S1	HisItemCurr	Item005	2020-05-14 20:26:33	2020-05-14 20:26:45	123	00:00:12	{"max": 65.57, "average": 31.43, "area": 3865.79, "median": 29.66, "var": 227.02, "std": 15.07, "skew": -0.2, "kurtosis":...
2020-05-14	259	S1	HisItemCurr	Item039	2020-05-14 20:26:33	2020-05-14 20:26:44	110	00:00:10	{"max": 19.0, "average": 6.5, "area": 714.99, "median": 7.08, "var": 18.36, "std": 4.29, "skew": 0.99, "kurtosis": 0.8, "Q...
2020-05-14	258	S1	HisItemCurr	Item005	2020-05-14 20:26:11	2020-05-14 20:26:23	121	00:00:12	{"max": 55.22, "average": 29.54, "area": 3573.95, "median": 25.24, "var": 191.93, "std": 13.85, "skew": -0.22, "kurtosis...
2020-05-14	258	S1	HisItemCurr	Item039	2020-05-14 20:26:11	2020-05-14 20:26:22	109	00:00:10	{"max": 12.36, "average": 3.15, "area": 342.81, "median": 1.82, "var": 7.84, "std": 2.8, "skew": 2.09, "kurtosis": 3.01, "...
2020-05-14	257	S1	HisItemCurr	Item039	2020-05-14 20:25:11	2020-05-14 20:25:36	258	00:00:25	{"max": 19.7, "average": 7.35, "area": 1895.38, "median": 7.72, "var": 9.31, "std": 3.05, "skew": 0.8, "kurtosis": 3.94, "...
2020-05-14	257	S1	HisItemCurr	Item005	2020-05-14 20:25:10	2020-05-14 20:25:37	271	00:00:27	{"max": 70.86, "average": 30.83, "area": 8354.3, "median": 30.07, "var": 109.06, "std": 10.44, "skew": 0.14, "kurtosis": ...
2020-05-14	256	S1	HisItemCurr	Item039	2020-05-14 20:24:26	2020-05-14 20:24:52	262	00:00:26	{"max": 12.42, "average": 2.36, "area": 617.61, "median": 1.82, "var": 3.87, "std": 1.97, "skew": 3.62, "kurtosis": 12.59,...
2020-05-14	256	S1	HisItemCurr	Item005	2020-05-14 20:24:25	2020-05-14 20:24:52	273	00:00:27	{"max": 55.51, "average": 26.97, "area": 7364.11, "median": 25.24, "var": 98.79, "std": 9.94, "skew": 0.48, "kurtosis": 1...

- Labeling

- 저장된 UP/DOWN 신호를 이용하여 라벨링

Label

index_date	index_num	defServer	defTable	defColumn	startTime	stopTime	labels
2020-05-14	1	S1	HisItemCurr	Item039	2020-05-14 05:01:29	2020-05-14 05:01:44	{"up1down0": 0}
2020-05-14	2	S1	HisItemCurr	Item039	2020-05-14 05:02:03	2020-05-14 05:02:14	{"up1down0": 1}
2020-05-14	3	S1	HisItemCurr	Item039	2020-05-14 05:02:22	2020-05-14 05:02:29	{"up1down0": 0}
2020-05-14	4	S1	HisItemCurr	Item039	2020-05-14 07:38:16	2020-05-14 07:38:26	{"up1down0": 0}
2020-05-14	5	S1	HisItemCurr	Item039	2020-05-14 07:38:38	2020-05-14 07:38:52	{"up1down0": 1}
2020-05-14	6	S1	HisItemCurr	Item039	2020-05-14 07:38:59	2020-05-14 07:39:10	{"up1down0": 0}
2020-05-14	7	S1	HisItemCurr	Item039	2020-05-14 08:07:10	2020-05-14 08:07:21	{"up1down0": 1}
2020-05-14	8	S1	HisItemCurr	Item039	2020-05-14 08:15:43	2020-05-14 08:15:54	{"up1down0": 0}
2020-05-14	9	S1	HisItemCurr	Item039	2020-05-14 08:16:14	2020-05-14 08:16:39	{"up1down0": 1}
2020-05-14	10	S1	HisItemCurr	Item039	2020-05-14 08:20:11	2020-05-14 08:20:37	{"up1down0": 0}
2020-05-14	11	S1	HisItemCurr	Item039	2020-05-14 08:20:46	2020-05-14 08:21:11	{"up1down0": 1}
2020-05-14	12	S1	HisItemCurr	Item039	2020-05-14 08:21:23	2020-05-14 08:21:49	{"up1down0": 0}
2020-05-14	13	S1	HisItemCurr	Item039	2020-05-14 08:22:08	2020-05-14 08:22:34	{"up1down0": 1}
2020-05-14	14	S1	HisItemCurr	Item039	2020-05-14 08:25:19	2020-05-14 08:25:45	{"up1down0": 0}
2020-05-14	15	S1	HisItemCurr	Item039	2020-05-14 08:26:04	2020-05-14 08:26:31	{"up1down0": 1}
2020-05-14	16	S1	HisItemCurr	Item039	2020-05-14 08:26:42	2020-05-14 08:27:08	{"up1down0": 0}
2020-05-14	17	S1	HisItemCurr	Item039	2020-05-14 08:27:23	2020-05-14 08:27:48	{"up1down0": 1}

3-3. Pre-Process

- Pre-Processing

- 모델에 적용하기 위해 원하는 형태로 전처리

Features

```
# DATA
[['14.29' '3.56' '519.52' ... '7.8' '2.83' '2.68']
 ['21.3' '8.57' '1268.47' ... '11.9' '8.28' '8.9']
 ['13.87' '3.66' '406.17' ... '8.51' '2.87' '2.42']
 ...
 ['17.47' '5.56' '817.68' ... '9.56' '4.98' '5.14']
 ['13.93' '3.27' '474.83' ... '7.41' '2.51' '2.28']
 ['19.91' '7.04' '1035.12' ... '10.65' '6.6' '7.02']]

 # LENGTH
2963

# TYPE
<class 'numpy.ndarray'>
```

Target

```
# DATA
[1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0,
...
1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1,
0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0]
```

```
# LENGTH
2963

# TYPE
<class 'list'>
```

- Scaling(평균 0, 분산 1로 데이터 분포도 변환)

- 정규 분포 형태의 표준 스케일링 적용

> Features

```
# DATA
[[-0.68371451 -1.00025274 -0.44138566 ... -1.37114    -0.94833775
  -0.88346797]
 [-1.15891996 -1.18069918 -0.62345674 ... -1.47398456 -1.10423386
  -1.03055082]
 [ 1.04456246  0.97206883  1.09280422 ...  0.21753789  1.01367185
   1.06281441]
 ...
 [-0.76008682 -0.87435987 -0.88429488 ... -0.10452798 -0.92552368
  -0.94845807]
 [-0.99486094 -1.12614561 -0.5699796  ... -1.48751674 -1.07761745
  -1.00318657]
 [-1.10517649 -1.13453846 -0.70996628 ... -1.28453405 -1.08902448
  -1.02028923]]
```

3-4. Modeling

- Split

- Train 데이터와 Test 데이터를 7:3으로 분할 구성
- 재현성을 위하여 random state(seed)는 0으로 입력한다.
- 분할된 변수: X_train, X_test, y_train, y_test

```
# X_train
[[-0.68371451 -1.00025274 -0.44138566 ... -1.37114    -0.94833775
  -0.88346797]
 [-1.15891996 -1.18069918 -0.62345674 ... -1.47398456 -1.10423386
  -1.03055082]
 [ 1.04456246  0.97206883  1.09280422 ...  0.21753789  1.01367185
   1.06281441]
 ...
 [-0.99768954 -1.05060989 -0.6584043  ... -1.22499246 -1.02818697
  -0.95871966]
 [-1.1164909  -1.07998489 -0.8433563  ... -0.80549489 -1.07761745
  -1.02370976]
```

```
 [ 0.92010389  0.78322953 -0.35065294 ...  1.53015932  0.64864681
   0.59078107]]

# X_test
[[ 0.84090298  0.90492596  1.02515163 ...  0.17423492  0.95663669
   0.99440378]
 [ 1.29913681  1.47564029  1.52888762 ...  0.60997111  1.54219769
   1.58615572]
 [ 1.2934796   1.38751528  0.97930239 ...  0.63432903  1.39390627
   1.48696031]
 ...
 [-0.76008682 -0.87435987 -0.88429488 ... -0.10452798 -0.92552368
  -0.94845807]
 [-0.99486094 -1.12614561 -0.5699796  ... -1.48751674 -1.07761745
  -1.00318657]
 [-1.10517649 -1.13453846 -0.70996628 ... -1.28453405 -1.08902448
  -1.02028923]]

# y_train
[1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
...
0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0]

# y_test
[0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
1, 1, 0, 0, 0, 1, 0, 0,
...
0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1]
```

- Train / Test

- 로지스틱 회귀로 분류 수행

> Train

```
lr_clf = LogisticRegression()
lr_clf.fit(X_train, y_train)
```

> Test

```
lr_preds = lr_clf.predict(X_test)
```

3-5. Evaluation

- Evaluation

> accuracy & roc_auc

```
# 정확도와 roc_auc 측정
print('accuracy: {:0.3f}'.format(accuracy_score(y_test, lr_preds)))
print('roc_auc: {:0.3f}'.format(roc_auc_score(y_test, lr_preds)))
```

```
accuracy: 0.971
roc_auc: 0.970
```

- 하이퍼파라미터 최적화

- 규제(Regularization)와 규제 강도(alpha)
- C = 1 / alpha
- C값이 작을수록 규제 강도가 큼
- 하이퍼 파라미터 최적화

> Regularization & alpha

```
from sklearn.model_selection import GridSearchCV

params = {'penalty' : ['l2','l1'],
          'C': [0.01, 0.1, 1, 1, 5, 10]}

grid_clf = GridSearchCV(lr_clf, param_grid=params, scoring='accuracy', cv=3)
grid_clf.fit(data_scaled, list_label)
print('최적 하이퍼 파라미터:{0}, 최적 평균 정확도:
{1:.3f}'.format(grid_clf.best_params_, grid_clf.best_score_))
```

```
최적 하이퍼 파라미터:{'C': 0.1, 'penalty': 'l1'}, 최적 평균 정확도:0.972
```

4. 결론

- 엘리베이터 데이터의 통계값을 이용한 분류 예측은 약 97%의 정확도로 높은 편이다.
- 로지스틱 회귀를 이용한 이진 분류 예측 성능이 뛰어나다.