

빅콘테스트

퓨처스리그 : 홍수ZERO

GodBoost

고경수 : star77sa@naver.com
문우혁 : lunanalyze@gmail.com
황산하 : hsh6449@jbnu.ac.kr

INDEX

서론

- 1)요약
- 2)변수

분석 및 전처리


- 1)IDEA
- 2)분석과정

결론

- 1)성능
- 2)결과

서론

1) 요약



“CatBoost를 사용하여 유입량을 예측”

서론 2) 변수

사용 변수		Dtype
평균유역강수		Float
A~D지역 강우		Float
D,E지역 수위		Float
홍수사상 별 시간 INDEX		Float



변수제거 이유

홍수사상번호는 예측해야하는 Data Set의 홍수사상번호가 전부 '26'이므로 불필요하다고 생각

년,월,일,시간은 강우데이터에 시간이 포함되어 있으므로 불필요해서 생략

서론 2) 변수

집단 1~6

사용 변수	Dtype
평균유역강수	Float
A~D지역 강우	Float
D,E지역 수위	Float
홍수사상 별 시간 INDEX	Float

강우데이터는 계속 더해지는 경향을 보이는 것으로 보아 누적강수로 사전에 처리된 것으로 보임
D지역 강우는 상관계수가 0.1로 낮게 나왔으나, 모델성능을 비교해 봤을 때 포함한게 더 좋았음

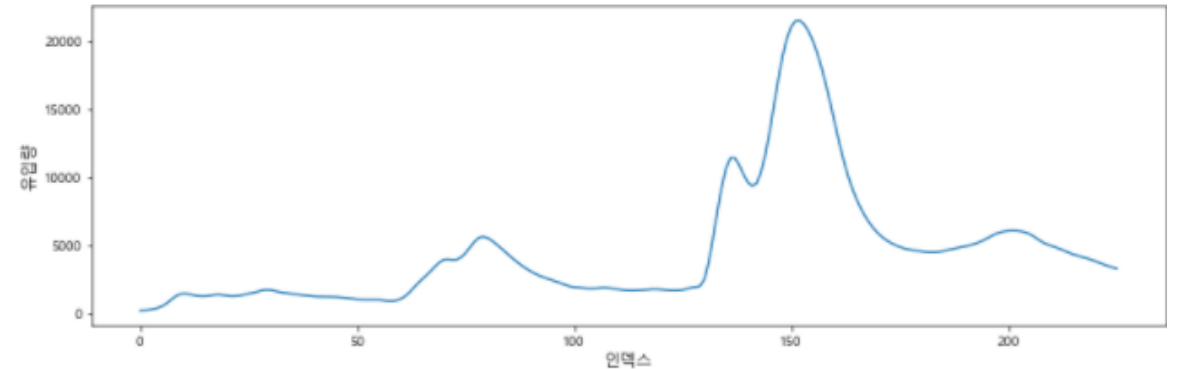
따라서 주어진 데이터를 그대로 사용

서론 2) 변수

새로 추가한 파생변수

사용 변수	Dtype
평균유역강수	Float
A~D지역 강우	Float
D,E지역 수위	Float
홍수사상 별 시간 INDEX	Float

그림 : 인덱스에 따른 사상번호 1 유입량



그림에 보이는 것처럼 사상번호
내에서 일정한 **경향성**을 가짐

강우데이터에 시간데이터가 포함되어 있다고 판단,
홍수사상 별 시간 INDEX를 순서대로 주었음

분석 및 전처리 1) IDEA

①집단 6개의 처리

집단 6개를 각각 학습시키고 가장 **RMSE가 낮게 나온 집단을 이용**

WHY?

집단을 무시하고 전부다 학습시키면 같은 변수간 다중공선성 발생
Train Set을 크게 증가시키기 위해 아래로 이어 붙이면 과적합 발생



[집단 1 과 집단 2 상관계수]

Pycaret의 compares_model()으로 비교해본 결과
train set의 Shuffle 유무로 성능이 크게 차이남

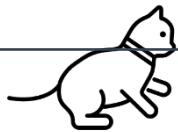
→ 시간의 정보를 담고 있을 것 같다는 의심의 계기

분석 및 전처리 1) IDEA

②사용 모델 : CatBoost

의사결정 나무의 Gradient Boosting 알고리즘을 이용하는 머신러닝 모델. 예측에 뛰어난 성능을 보이고 과적합을 줄여줌

CatBoost 장점



특별한 파라미터의 튜닝없이도 우수한 성능과 정확도를 보임
빈도에 따른 가중치를 두어 인코딩하는 방식을 사용함으로써 범주형 변수가 있을 때 강함
모델 자체적으로 Scailing이 가능해서 편리함

분석 및 전처리 1) IDEA

③Train-Test Split

사상번호 별로 행의 개수가 다르고 시간의 경향을 담고 있다는
가정하에 사상번호를 기준으로 Train과 Test Set을 나누고
Shuffle을 하지 않았음

점수 예측을 위한 임시 train(홍수사상번호 1 ~ 21) / test(홍수사상번호 22 ~ 25) 분리

```
X_train, X_test, y_train, y_test = train_test_split(train[['홍수사상번호', '연', '월', '일', '시간', *유역평균강수, *강우A,  
*강우B, *강우C, *강우D, *수위E, *수위D]],  
train['유입량'], test_size=0.2109996, shuffle=False)  
  
Original_train = X_train # 사상번호 1 ~ 21  
Original_test = X_test # 사상번호 22 ~ 25
```

경향성으로 인한 부정확성을 낮추기 위함

데이터의 약 20%정도가 test set

분석 및 전처리

2) 분석과정

①집단 6개의 전처리

```
data.columns = ['홍수사상번호', '연', '월', '일', '시간', '유입량',  
'구역평균강수1', '강우A1', '강우B1', '강우C1', '강우D1', '수위E1', '수위D1',  
'구역평균강수2', '강우A2', '강우B2', '강우C2', '강우D2', '수위E2', '수위D2',  
'구역평균강수3', '강우A3', '강우B3', '강우C3', '강우D3', '수위E3', '수위D3',  
'구역평균강수4', '강우A4', '강우B4', '강우C4', '강우D4', '수위E4', '수위D4',  
'구역평균강수5', '강우A5', '강우B5', '강우C5', '강우D5', '수위E5', '수위D5',  
'구역평균강수6', '강우A6', '강우B6', '강우C6', '강우D6', '수위E6', '수위D6']
```

```
train = data.iloc[1:2892]
```

```
train.head()
```

집단 1

... 집단6

홍수사 상번호	연	월	일	시 간	유입량	구역평균	강우	강우	강우	강우	수위	수위D5	구역평균	강우	강우	강우	강우	수위	수위D6		
						강수1	A1	B1	C1				강수6	A6	B6	C6	D6			E6	
1	1.0	2006.0	7.0	10.0	8.0	189.100000	6.4	7	7	7	...	8	2.54	122.66	6.4	7	7	8	8	2.54	122.61
2	1.0	2006.0	7.0	10.0	9.0	216.951962	6.3	7	8	7	...	10	2.53	122.648	7.3	7	8	10	10	2.53	122.6
3	1.0	2006.0	7.0	10.0	10.0	251.424419	6.4	7	9	7	...	11	2.53	122.636	8.2	7	9	10	11	2.53	122.59
4	1.0	2006.0	7.0	10.0	11.0	302.812199	7.3	7	10	7	...	14	2.53	122.62	11.3	9	10	15	14	2.53	122.585
5	1.0	2006.0	7.0	10.0	12.0	384.783406	8.2	7	12	8	...	16	2.53	122.604	14.4	12	12	18	16	2.53	122.575

집단을 따로따로 이용하기 위해 칼럼명을 재정의
Submmission도 마찬가지로 재정의

분석 및 전처리

2) 분석과정

②시간 INDEX 추가

```
: X_1 = X.iloc[:226]
X_2 = X.iloc[226:326]
X_3 = X.iloc[326:407]
X_4 = X.iloc[407:441]
X_5 = X.iloc[441:535]
X_6 = X.iloc[535:581]
X_7 = X.iloc[581:632]
X_8 = X.iloc[632:673]
X_9 = X.iloc[673:741]
X_10 = X.iloc[741:827]
X_11 = X.iloc[827:937]
X_12 = X.iloc[937:1064]
X_13 = X.iloc[1064:1131]
X_14 = X.iloc[1131:1181]
X_15 = X.iloc[1181:1443]
X_16 = X.iloc[1443:1648]
X_17 = X.iloc[1648:1773]
X_18 = X.iloc[1773:1838]
X_19 = X.iloc[1838:1933]
X_20 = X.iloc[1933:2197]
X_21 = X.iloc[2197:2281]
X_22 = X.iloc[2281:2565]
X_23 = X.iloc[2565:2694]
X_24 = X.iloc[2694:2788]
X_25 = X.iloc[2788:2891]

dtlists = []
index = pd.DataFrame()
for j in range(1, 26):
    eval('dtlists.append(X_'+str(j)+'')')
    dtlists[j-1] = dtlists[j-1].reset_index()
    dtlists[j-1]['인덱스'] = dtlists[j-1].index
    dtlists[j-1]['정규화인덱스'] = dtlists[j-1]['인덱스'] / (len(dtlists[j-1])-1)
    index = pd.concat([index, pd.DataFrame(dtlists[j-1])])

index = index.reset_index()
train['정규화인덱스'] = index['정규화인덱스']
```

시간에 따른 경향성을 학습하기 위해 사상번호마다 행이 다르므로 정규화된 인덱스 추가

분석 및 전처리

2) 분석과정

③CatBoost 모델링

집단 1

```
X_train = Original_train[['유역평균강수[0]', '강우A[0]', '강우B[0]', '강우C[0]', '강우D[0]', '수위E[0]', '수위D[0]']]
X_test = Original_test[['유역평균강수[0]', '강우A[0]', '강우B[0]', '강우C[0]', '강우D[0]', '수위E[0]', '수위D[0]']]

categorical_features_indices1 = np.where(X_train.dtypes == np.object)[0]

model = cat
model.fit(X_train, y_train, cat_features = categorical_features_indices1)
```

집단 별로 각각 모델링 후 성능 비교

분석 및 전처리

2) 분석과정

④집단 별 성능 비교

```
pred = model.predict(X_test)

rmse = (np.sqrt(np.mean(mean_squared_error(y_test, pred))))
rmse
# 1: 774
```

집단	RMSE
집단 1	759.7370959418506
집단 2	803.1710778071422
집단 3	792.0017813608149
집단 4	704.6983613897916
집단 5	670.9822236789764
집단 6	780.2260861941985

RMSE가 가장 낮은 **집단 5** 선택

결론

1) 모델 성능 평가

①모델 별 성능 평가

모델	RMSE
XGBoost	552.0518058209736
LGBM	538.0813745593989
CatBoost	421.1103896772417

최종적으로 선택한 변수들을 동일하게 설정하여 모델간 RMSE를 비교

CatBoost의 RMSE가 가장 낮아 **성능이 가장 좋다**고 판단

결론

1) 모델 성능 평가

②집단 5 학습 후 전체 집단 예측 평균값 사용

```
mean_preds = (pred1 + pred2 + pred3 + pred4 + pred5 + pred6) / 6
```

```
rmse = (np.sqrt(np.mean(mean_squared_error(y_test, mean_preds))))  
rmse
```

```
420.4005862789978
```

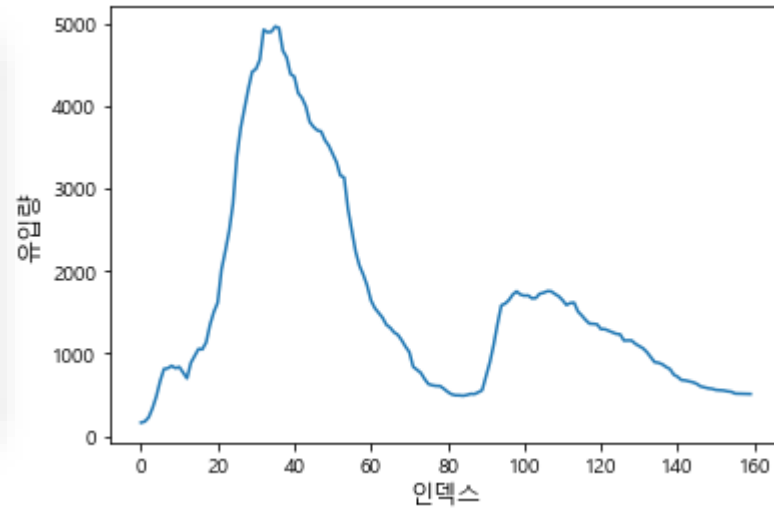
모델	RMSE
집단 5 데이터만을 이용하여 예측	421.1103896772417
집단 1~6 데이터를 이용하여 예측 (평균)	420.4005862789978

집단 5의 데이터만을 이용하여 학습을 시켰지만,
전체 데이터에 적용했을 때에도 거의 비슷한 RMSE를 보이며

일반화가 가능하다고 판단

결론 2) 최종 결과

```
pred
array([ 161.75740327,  175.44486783,  225.56723231,  337.32149606,
        480.61763002,  667.58506526,  812.67223241,  820.89320653,
        850.68861443,  822.52220278,  836.85525907,  769.52858154,
        699.66514232,  887.43973669,  971.98509473, 1054.86761336,
       1050.94368635, 1135.69286645, 1349.64244341, 1504.53720304,
       1617.07042641, 2019.37504007, 2242.88696971, 2487.91334101,
       2817.02510902, 3385.73990386, 3735.07049575, 3976.19739489,
       4210.81115554, 4417.19213033, 4451.68018145, 4562.91294241,
       4928.13856687, 4896.8600933 , 4902.79090969, 4966.06374669,
       4949.93477465, 4675.07083218, 4591.95466854, 4389.95976784,
       4359.1122805 , 4158.72593847, 4099.95333732, 3998.81354324,
       3810.54134161, 3749.43073968, 3707.07746307, 3694.03597581,
       3587.18192124, 3520.51448588, 3424.14225421, 3322.29433828,
       3159.43413736, 3135.5005931 , 2754.44753809, 2489.54206508,
```



시간의 **경향성**이 반영되도록 예측되었음을 볼 수 있다.