수상작 리뷰 보고서

태그

주제

- 홍수 예측 데이터 셋을 이용한 특정 지역의 홍수 발생 확률 예측
- 링크: https://www.kaggle.com/competitions/playground-series-s4e5

데이터

- train: 홍수 발생 확률 예측하기 위한 다양한 환경적 및 인위적 요인을 포함
  - MonsoonIntensity: 계절의 강우의 강도, 높은 값은 강력한 몬순을 나타냄.
  - TopographyDrainage: 지형의 배수 특성. 높은 값은 더 나은 배수 성능을 나타낼 수 있음.
  - RiverManagement: 강 관리 수준. 높은 값은 더 나은 관리 상태를 나타냄.
  - Deforestation: 삼림 벌채 수준. 높은 값은 삼림의 심각한 손실을 나타냄.
  - Urbanization: 도시화 정도
  - ClimateChange: 기후 변화의 영향 정도
  - DamsQuality: 댐의 품질
  - Siltation: 하천이나 댐의 토사 퇴적 수준
  - AgriculturalPractices: 농업 관행의 홍수에 대한 영향도.
  - DrainageSystems: 배수 시스템의 효율성.
  - CoastalVulnerability: 해안 지역의 취약성.
  - Landslides: 산사태 발생 가능성.
  - Watersheds: 유역 관리 상태.
  - DeterioratingInfrastructure: 악화되는 인프라 수준.
  - PopulationScore: 인구 점수.

    - WetlandLoss: 습지 손실의 정도.
    - InadequatePlanning: 부적절한 계획 수준.
    - PoliticalFactors: 정치적 요인. 정책이나 정부의 대응 등을 포함.
    - FloodProbability: 예측되는 홍수 발생 확률 (목표 변수).
- test: 홍수 발생 확률 예측하기 위한 다양한 환경적 및 인위적 요인을 포함.

코드 흐름

(1) 특징 엔지니어링

```
unique_vals = []
for df in [train_data, test_data]:
    for col in initial_features:
        unique_vals += list(df[col].unique())
unique_vals = list(set(unique_vals))
```

- `unique_vals` 리스트를 생성하여 `train_data`와 `test_data`의 `initial_features` 열에 포함된 고유값을 모두 수집하는 과정으로, 이 고유값들은 이후에 특정 값의 개수를 세는 데 사용.

```
for df in [train_data, test_data]:
    df['fsum'] = df[initial_features].sum(axis=1)
    df['fstd'] = df[initial_features].std(axis=1)
    df['special1'] = df['fsum'].isin(np.arange(72, 76))
    df['fskew'] = df[initial_features].skew(axis=1)
    df['fkurtosis'] = df[initial_features].kurtosis(axis=1)

    for i in [0.0, 0.2, 0.4, 0.5, 0.6, 0.8, 1.0]:
        df['q_{}'.format(int(i*100))] = df[initial_features].

    for v in unique_vals:
        df['cnt_{}'.format(v)] = (df[initial_features] == v).
```

- `train_data`, `test_data`에 여러 새로운 특징을 추가하는 작업이 진행됨. `fsum`은 각 행의 `initial_features` 열 값의 합계를 저장하고, `fstd`는 표준 편차를 계산하여 저장함.

`special1`은 `fsum`이 72에서 76 사이인지 확인하는 값이며, `fskew`와 `fkurtosis`는 각 행의 왜도와 첨도를 계산해 저장함. 추가로, 다양한 분위수 값과 고유값에 대한 카운트를 계산해 새로운 열에 저장하여 데이터의 통계적 특성을 파악함.

```
X_train, X_test = train_test_split(
     train_data.drop(columns=["id"]), test_size=0.1, random_s
```

- `train_data`에서 `id` 열을 제외한 나머지 데이터를 훈련 세트와 테스트 세트로 분할함. 이때 데이터의 10%를 테스트 세트로 사용하며, `FloodProbability` 열의 비율을 유지하도록 설정
- 이후 predictor 모델을 사용해 모델의 성능을 평가하고 다양한 모델의 예측 성능을 비교할 수 있도록 진행함.
- 가장 좋은 예측 값을 새로운 열로 할당.

⏩위의 코드를 통해 홍수 발생 확률을 예측하기 위한 머신러닝 파이프 라인을 구성함

- 이후, 사용하지 않는, 각 행의 왜도, 첨도, 중복된 피처를 제거하고 흥미로운 결과를 보여주는 `train.groupby("sum")["FloodProbability"].std()`을 통해 `train["sum"]`은 `train[test.columns].sum(axis=1)` 각 행의 합계를 나타냄.

(2) 사용 모델

1. 30개 이상의 GBM(Gradient Boosting Machine)을 훈련, 이때 다양한 피처 세트를 사용 (정렬된 원본 피처, 카운터 피처 ["nb_inf4", "nb_inf3", "nb_inf2", "nb_sup6", "nb_sup7", "nb_sup8”]를 사용
2. 타겟 인코딩 이용하거나 타겟 변환 사용 (강한 신로와 노이즈가 있는 신호가 있었기 때문)

```
# 변환된 타겟
train["target_transf"] = train["FloodProbability"] - df[origi
train["target_transf"] = (train["FloodProbability"]*400 - tra
```

3. 하이퍼 파라미터 최적
    - CatBoost와 XGBoost의 depth

- LightGBM의 num_leaves
- XGBoost와 LightGBM의 alpha, lambda, min_child_weight 또는 min_child_samples
- CatBoost의 bagging_temperature, random_strength
- XGBoost와 LightGBM의 subsample, colsample_by_node

⏩ GBM 방법의 기본 grow_policy를 사용했으며, XGBoost, CatBoost 및 LightGBM은 각각 다른 기본 grow_policy 값을 가지고 있어 예측의 다양성을 제공했음. 기본 learning_rate 또는 0.1을 사용했으며, n_iterations에 맞춰 early_stopping(od_wait in CatBoost)을 사용

4. 앙상블
    - LinearRegression을 사용하여 positive=True, fit_intercept = False로 앙상블 훈련했고 이후 Ridge 시도
    - 모델 모델을 3회 반복 K-Fold로 훈련시키키고 CV를 통해 앙상블에 사용할 피처 선택
    - OOF 예측 파일 3개와 더불어 3개의 새로운 피처를 추가하여 총 6개의 OOF 예측 파일을 생성하 최종 앙상블을 적합시킴.

차별 점 및 배울 점

1. 각 행의 합, 표준편차, 최댓값을 축을 따라 정렬함으로써 의미없는 칼럼은 제거하고 의미있는 특징의 값을 추출할 수 있었음.

→ 이러한 특징의 축을 따라 정렬하여 데이터 엔지니어링 하는 방식이 처음 접했고 이를 통해 유의미한 결과를 얻어낼 수 있다는 점이 인상깊었음.

2. K-Fold 뿐만 아니라 OOF 방법도 이용하여 모델의 일반화 성능을 높이고 예측 결과의 신뢰성을 강화하였음.
3. 이 수상작에서는 AutoGluon을 이용하여 여러 모델을 앙상블하여 예측 성능 향상시킴, 이를 통해 여러 모델의 결과를 통합하여 더 나은 예측을 제공할 수 있었음.

→ AutoGluon 머신러닝에 대해 새로 배울 수 있었고 결과를 통합하여 더 나은 결과를 얻을 수 있다는 것이 인상 깊었음.

▶ AutoGluon 이용 예시

1. 모델 필터링 및 새로운 피처 추가

```
models = [m for m in models if m not in ["cb7_9_12", "xgb11_1
for n in range(n_repeats):
    for df in [oofs[n], preds[n]]:
        df["cb7_9_12"] = (df["cb7"] + df["cb9"] + df["cb12"])
        df["xgb12_13"] = (df["xgb12"] + df["xgb13"]) / 2
        df["autog3_5"] = (df["autog3"] + df["autog5"]) / 2
        df["cb2_3"] = (df["cb2"] + df["cb3"]) / 2
        df["lgbmcorr_xgb8_lgbm9"] = (df["bestlgbmcorr"] + df[
```

2. 최종 모델 리스트 업데이트 및 최종 모델 필터링

```
for m in ["cb7_9_12", "xgb12_13", "cb2_3", "autog3_5", "lgbmc
    if m not in models:
        models.append(m)
final_models = [ m for m in models if m not in [ ... ]]
```

3. 앙상블 및 성능 평가

```
params = {"alpha": .001, "random_state": seed, "fit_intercept
all_intercepts, all_coefs, blend_scores = [], [], []

oof_mean = pd.Series(0.0, index = train.index, name= target)

for r in range(n_repeats):
    new_model = {
        "train": oofs[r],
        "test": preds[r],
        "features": final_models,
        "my_model": Ridge(**params),
    }
    res = do_ensemble(**new_model, do_feat_imp = True)
    all_intercepts.extend(res["intercept"])
    all_coefs.append(res["coefs"])
    blend_scores.append(res["oofs_score"])
```

```
    oof_mean += res["oofs"] / n_repeats
print(f"{bold}Final mean RMSE {bold_blue}{np.mean(blend_score
```

4. OOF R2 점수 계산 및 시각화

```
all_scores = {}
ascending = True
for r in range(n_repeats):
    for m in final_models:
        if r == 0:
            all_scores[m] = []
        all_scores[m].append(score_(train[target], oofs[r][m]

all_scores["Ensemble"] = blend_scores

all_scores = pd.DataFrame(all_scores)
_col_order = list(all_scores.mean(axis=0).sort_values(ascendi
all_scores = all_scores[_col_order]
```

5. 최종 모델 학습 및 예측, 시각화

```
model = Ridge(**params)
model.fit(_oofs, train[target])
y_oof = model.predict(_oofs)
y_pred = pd.Series(model.predict(_preds), index = test.index,
y_pred.to_csv(f"submission_with_autogluon.csv")
sns.barplot(x = model.coef_, y = final_models, ax = ax, color
```