2021 데이콘 전력사용량 예측 AI 경진대회 : 1위

- 특징
  - 다른 대회의 수상작들을 참고해서 방향성을 정함(코드, 알고리즘 아이디어 참고)
  - 사용할 알고리즘의 document 참고
  - 다른 지표는 잘 나오지 않더라도 무조건 대회에서 지정하는 평가 지표를 기준으로 진행

데이터 & 평가 소개

전력사용량을 과대추정 하는 경우 → 과도한 비용 산정

전력사용량을 과소추정 하는 경우 → 전력 공급에 큰 차질

Preprocessing

데이터는 대체로 시계열 성격을 띄는 듯 했음.

그러나, 시계열(autoregression, ARIMA, ARIMAX, RNN 등) 기법에서는 좋은 성능을 보이지 못했음.

진행 방향

1. 건물별로 모델링하여 총 60개의 모델 만들기

   건물별 train 데이터는 2,000 여개의 데이터에 불과.

   → 딥러닝 기반의 모델보다는 boosting 계열의 모델이 적합하다고 판단.

2. 시계열 데이터를 일반 회귀 문제로 변환

   일반 회귀 문제로 변환하기 위해 데이터에서 시간 관련 변수 추가

   참고 링크 : https://www.analyticsvidhya.com/blog/2019/12/6-powerful-feature-engineering-techniques-time-series/

전처리

train set 시각화 결과, 평일/주말, 요일, 시간대에 따라 사용량이 크게 차이 나는 것을 확인.

→ 시간 관련 변수 중 월, 요일, 시간, 공휴일 여부 등을 추가해주는 것이 적합하다는 판단

- 파생 변수 만들기 : 통계
  - 건물별, 요일별, 시간대별 전력사용량 평균
  - 건물별, 시간대별 전력사용량 평균
  - 건물별, 시간대별 전력사용량 표준편차 등

  관련 통계량을 feature로 추가

  → 실제로 이 과정을 통한 성능 향상이 있었음
- 시간대의 순환적 성격 반영하기 (아이디어 : 데이콘 태양광 발전량 예측 대회)

  시간 변수(0시 ~ 23시)는 0시와 23시가 비슷한 시간대임에도 불구하고, 숫자 상으로는 제일 먼 시간대로 받아들일 수 있기 때문에 sin, cosine 함수 활용

  해당 대회 링크 : https://dacon.io/competitions/official/235680/codeshare/2366?page=1&dtype=recent

  시간의 순환적 성격 반영하기 링크 : https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/
- 불쾌 지수 & 시간적 오차 고려하기(아이디어 : 전력 사용량 예측 대회)

- 불쾌 지수 : 기온, 습도 고려
- 시간적 오차 : 기온이 실제 냉방 가동에 이르기까지

→ cooling degree hour

참고 링크 : https://dacon.io/competitions/official/235736/codeshare/2743?page=1&dtype=recent

- '태양광발전시설 보유 여부', '비전기냉방시설 보유 여부' 제거

  건물별 모델링을 진행할 것이기 때문에 건물별로 동일하게 주어진 변수 제거

모델링

*카테고리 변수(month, week 등)을 더미로 변환하지 않음. → tree 기반의 부스팅 모델(XGBoost) 사용

tree 기반의 부스팅 모델은 변수 중 일부를 선택하여 해당 tree를 키워나가면서 학습.

만약 month가 선택된다면 6, 7, 8월이 트리에 의해 잘 분류되어 카테고리 변수처럼 사용될 수 있음

하지만, 더미로 처리할 경우 month 관련 2개의 더미 중 하나만 선택되어 학습하면 일부 범주가 누락되는 경우가 발생할 수 있음(이 문제는 범주가 많을수록 더 심해짐)

XGBoost를 선택한 이유

부스팅 모델 중 시계열 관련 예측에 일반적으로 가장 좋은 성능을 보이는 특징을 가짐

Catboost나 LGBM 등 다양한 부스팅 모델을 같이 실험해보았으나, XGBoost 단일 모델이 제일 좋은 성능을 보임

성능 : XGBoost > Catboost > LGBM

*XGB와 Catboost는 앙상블을 시도했을 때 대회 최상위권의 성능을 보임

모델 튜닝

- skicit-learn의 GridSearchCV

  *튜닝 범위를 넓게 했을 때 오히려 성능이 떨어지는 것을 확인

  → 데이터가 작아 과적합이 쉽게 일어나는 것으로 판단

  *validation set : 가까운 날짜일수록 예측할 데이터와 비슷한 양상일 것이라는 가정 하에 train set에서 마지막 일주일에 해당하는 데이터만 사용

- Custom 목적함수

  일반적으로 XGBoost 학습에 사용하는 RMSE 대신 Custom Objective Function 사용

  → 위에서 언급된 과소추정과 과대추정에 대한 차이를 반영하기 위함

  일반적인 MSE를 변형하여 만약 예측값이 실제값보다 작을 경우 높은 가중치를 주도록 목적 함수 정의

```
def weighted_mse(alpha = 1) :
  def weighted_mse_fixed(label, pred):
    residual = (label - pred).astype("float")
    grad = np.where(residual > 0, -2 * alpha * residual, -2 * residual)
    hess = np.where(residual > 0, -2 * alpha, 2.0)
    return grad, hess
  return weighted_mse_fixed
```

  XGB에서 목적함수를 정의하는 방식

  → grad : 1차 미분 함수, hessian : 2차 미분 함수

  residual이 양수일 경우(과소추정), alpha만큼의 가중치가 곱해진다.

  alpha를 인자로 받는 함수로 wrapping 하여 활용성을 높임

```
xgb_reg = XGBRegressor(n_estimators = 10000, eta = xgb_params.iloc[47,1], min_child_weight = xgb_params.iloc[47,2],
                       max_depth = xgb_params.iloc[47,3], colsample_bytree = xgb_params.iloc[47,4],
                       subsample = xgb_params.iloc[47,5], seed=0)

xgb_reg.set_params(**{'objective':weighted_mse(100), 'metrics':'mse'})
```

새 목적함수를 적용했을 때와 적용하지 않았을 때를 비교하면, 거의 모든 건물에서 SMAPE가 개선됨

새로운 목적함수의 하이퍼파라미터는 1, 3, 5, 7, 10, 25, 50, 75, 100의 후보군에서 튜닝

일반 RMSE에서 훈련한 모델과 비교하여 validation set의 SMAPE가 감소한 경우에만 적용

custom 목적함수 관련 링크 : https://xgboost.readthedocs.io/en/latest/tutorials/custom_metric_obj.html

모든 하이퍼파라미터를 한 번에 튜닝할 경우 시간이 매우 오래 걸림

1. n_estimators를 100, eta를 0.01로 고정하고 다른 하이퍼파라미터 튜닝
2. early stopping 기능을 활용해 n_estimators 튜닝
3. 마지막으로 weighted_mse의 alpha 값 튜닝

   이전 단계의 하이퍼파라미터는 주어진 값으로 고정, 각 단계에서 튜닝할 하이퍼파라미터만 변화시키며 튜닝
4. inference

   3번에서 튜닝한 하이퍼파라미터를 활용하여 전체 train set(validation set 포함)에 훈련시키고, test set에 예측 진행

   test set의 전처리 과정은 train set과 완전히 동일, test set은 3시간 또는 6시간 단위로 주어지는 예보 데이터만을 사용하기 때문에 pandas의 interpolate를 활용하여 선형 보간한 후 사용

   (+) XGBoost의 예측값이 seed에 따라 변화하는 것을 파악, seed의 효과를 없애기 위해 0부터 5까지 총 6개의 seed에 대해 각각 예측한 뒤 이들의 평균으로 예측값을 만들었다. (seed ensemble)

Post-processing

성능 향상을 위해 후처리 진행

이는 weighted mse를 적용했던 논리와 마찬가지로 과소추정을 막기 위해, 예측 전 마지막 4주의 건물별, 요일별, 시간대별 최솟값을 계산하여 같은 건물의 같은 요일, 같은 시간대의 예측값과 비교해 만약 최솟값이 예측값보다 크다면 최솟값으로 대체

```
for i in range(60):
    min_data = train_to_post.loc[train_to_post.num == i+1, ].iloc[-28*24:, :] ## 건물별로 직전 28일의 데이터 불러오기
    ## 요일별, 시간대별 최솟값 계산
    min_data = pd.pivot_table(min_data, values = 'power', index = ['day', 'hour'], aggfunc = min).reset_index()
    pred = df.answer[168*i:168*(i+1)].reset_index(drop=True) ## 168개 데이터, 즉 건물별 예측값 불러오기
    day =  test_to_post.day[168*i:168*(i+1)].reset_index(drop=True) ## 예측값 요일 불러오기
    hour = test_to_post.hour[168*i:168*(i+1)].reset_index(drop=True) ## 예측값 시간 불러오기
    df_pred = pd.concat([pred, day, hour], axis = 1)
    df_pred.columns = ['pred', 'day', 'hour']
    for j in range(len(df_pred)):
        min_power = min_data.loc[(min_data.day == df_pred.day[j])&(min_data.hour == df_pred.hour[j]), 'power'].values[0]
        if df_pred.pred[j] < min_power:
            pred_clip.append(min_power)
        else:
            pred_clip.append(df_pred.pred[j])
```

대부분의 건물에서는 큰 변화가 없었지만 일부 건물에서는 모델이 제대로 예측하지 못하는 부분을 후처리 과정을 통해 보완할 수 있었음

후기 : 건물별로 모델링을 진행한 것이 마음에 걸림, 실제로 한국에너지공단이 관리하는 건물은 이보다 훨씬 많은데 건물별로 모든 모델을 만드는 것이 실효성이 떨어진다고 판단. 이를 보완하기 위해선 건물과 관련된 여러 변수들을 만들어주는 것(건물별 클러스터링 등)이 필요