[3주차] 1차 과제

생성일 2022년 5월 4일 오전 9:33

태그 공지

날짜 2022년 5월 4일

목차

Assignment 1.
Make the best model for TADA ETA data

- **제출 마감: 2022년 05월 22일 일요일 23시 59분**
- **제출 방법:** [3주차] 1차 과제 제출 페이지(~5/22)
- **과제 설명**

  Chapter 2에서 배운 tree algorithm들과 Chapter 3에서 배운 unsupervised learning을 통한 전처리를 응용해서 가장 좋은 성능을 내는 TADA ETA 예측 모델을 만들어봅시다.

1. 트레이닝셋과 테스트셋의 split은 아래와 같이 고정하도록 하겠습니다. 실습 코드에서는 아래의 random_state=0이 없었어서 데이터셋 생성 시 마다 순서가 달라졌는데, 이번엔 모두가 같은 data split을 가지고 동등하게 평가를 진행하도록 하겠습니다.

   ```
   tada_eta = tada_eta.sample(frac=1,
   random_state=0).reset_index(drop=True)
   train = tada_eta[:12000]
   test = tada_eta[12000:]
   ```

2. 데이터셋에 normalization이나 PCA와 같은 전처리를 하고, 그 후에 아래의 **Tips**과 같이 grid search와 category feature를 활용할 수 있는 방법을 적용해보세요. 참고로 decision tree의 특성과 데이터의 input feature가 많지 않다는 점을 고려했을 때, 전처리가 성능에 큰 도움이 되지 않을 수도 있습니다.

3. 이미 구현된 코드 사용은 sklearn 패키지 안에 있는 tree, ensemble 모델들과 unsupervised learning 알고리즘들로 제한하겠습니다. 다만, 직접 구현하신다면 모델 종류에 제한을 두지 않겠습니다.

4. input feature의 종류는 실습 코드에서는 4개(*api_eta, month, hour, distance*)만 사용했으나, 이번 과제에서는 원 데이터셋에서 자유롭게 변형하여 사용하셔도 됩니다.

5. **[채점 기준]** test set에 대한 MSE(소숫점 둘째 자리까지)를 기준으로 평가하도록 하겠습니다. **test set은 절대 학습에 이용하시면 안됩니다.** 원래 generalization이 되는지 평가하기 위해서는 test set이 주어지지 않는 것이 맞는데, 이 과제에서 주어지는 test set은 model selection을 위한 validation set이라고 생각하시고 과제를 진행해주세요. 그리고 randomness가 들어간 모델의 경우 평가를 위해서 random_state를 고정해주세요.
   - 목표 MSE 8.19 이하 달성: 10점
   - MSE 8.22 이하 달성: 9점
   - MSE 8.25 이하 달성: 8점
   - MSE 8.25 이하 달성: 7점
   - 공식과제 설명에 나온 방법(HW2의 feature set 그대로 사용 및 grid search 사용하지 않은 경우) 사용하지 않고 달성한 경우에 한함
   - MSE 8.25 초과: 6점
   - 제출하였으나 코드 실행 안됨: 5점

과제 Tips

1. 모델의 hyperparamter(*max_depth, min_samples_leaf* 등)를 찾을 때 아래 예시와 같이 sklearn의 GridSearchCV를 이용하면 주어진 모든 조합의 hyperparameter들에 대해서 train, validation을 진행해서 최적의 hyperparameter를 찾을 수 있습니다.

   - 아래 코드에서와 같이 cv=3이라고 하면 3-fold cross validation을 할 수 있습니다.

   ```
   from sklearn import ensemble
   from sklearn.metrics import mean_squared_error
   from sklearn.model_selection import GridSearchCV
   # Hyperparameter 조합 정의
   param_grid = {'n_estimators': [100, 200, 300],
           'max_depth': [2, 3, 4, 5],
            'min_samples_leaf': [1, 20, 100],
           'learning_rate': [0.01, 0.02, 0.05],
           'loss': ['ls']}
   # grid search 모델 정의, 학습 및 model selection
   grid_search =
   GridSearchCV(estimator=ensemble.GradientBoostingRegressor(),
                        param_grid=param_grid,
                        cv=3, n_jobs=-1, scoring='neg_mean_squared_error',
   verbose=2)
   grid_search.fit(x_train, y_train)
   print(grid_search.best_params_)
   reg =
   ensemble.GradientBoostingRegressor(**grid_search.best_params_)
   ```

   - 우리는 validation set이 정해져있으므로 아래와 같이 PredefinedSplit을 통해 바로 validation set을 정해주고 이를 cv에 넘겨주면, validation set에 대해서 가장 성능이 좋은 hyperparameter를 찾을 수 있습니다. 이 방법을 추천 드립니다.

   ```
   from sklearn.model_selection import GridSearchCV
   from sklearn.model_selection import PredefinedSplit

   # train set과 test set의 input과 output를 각각 이어 붙여서 X,y를 정의
   X = np.concatenate((x_train,x_test), axis=0)
   y = np.concatenate((y_train,y_test), axis=0)

   # 전체 데이터 X에서 training data에 해당하는 index는 -1, test data에 해당하는 index는 0이 되도록,
   # 여기서는 [-1, -1, ... , -1, 0, 0, ... , 0] 같은 형태의 1차원 배열 test fold와 predefined split을 정의
   pds = PredefinedSplit(test_fold=[-1]len(x_train)+[0]len(x_test))

   # grid search 모델 정의, 학습 및 model selection
   grid_search =
   GridSearchCV(estimator=ensemble.GradientBoostingRegressor(),
   ```

```
                    param_grid=param_grid,
                    cv=pds, n_jobs=-1,
scoring='neg_mean_squared_error', verbose=2)
grid_search.fit(X, y)
print(grid_search.best_params_)
```

2. **sklearn.ensemble.HistGradientBoostingRegressor**은 gradient boosting 모델의 또 다른 구현인데, 이 모델은 input feature로 category feature도 사용 가능합니다. 이를 위해서는 우선 아래 예시와 같이 category feature를 ordinal encoding을 통해서 0부터(category갯수-1)까지의 숫자 중 하나로 변환해줘야 합니다. 그 다음 ordinal encoding을 한 후, HistGradientBoostingRegressor을 정의할 때, input parameter인 categorical_feature를 통해 몇 번째 feature가 categorical feature인지 알려줘야 합니다.

   예를 들어 세 번째(index는 2) feature가 categorical feature라면 categorical_features=[2]가 됩니다. categorical feature를 쓰지 않더라도 이 HistGradientBoostingRegressor를 쓰면 데이터의 갯수가 많을 때 연산 속도가 훨씬 빠르니 GradientBoostingRegressor 대신 사용을 추천 드립니다.

```
tada_eta = pd.read_excel('/content/drive/MyDrive/Colab
Notebooks/data/tada_eta.xlsx')
# Ordianl Encoding
enc = sklearn.preprocessing.OrdinalEncoder(dtype=np.int32)
ordinal = enc.fit_transform(np.asarray(tada_eta['pickup_gu']).reshape(-1,1))
tada_eta['pickup_gu'] = ordinal[:,0]
tada_eta['distance'] = ((tada_eta['pickup_lat']-tada_eta['driver_lat'])**2 +
(tada_eta['pickup_lng']-tada_eta['driver_lng'])**2)*100000
tada_eta = tada_eta.drop(['id', 'created_at_kst', 'driver_id', 'pickup_lng',
'pickup_lat', 'driver_lng','driver_lat'],1)
```

   ⚠️ **주의사항:** category feature에 대해서는 normalization과 PCA 등의 차원 축소를 하시면 안됩니다. 이 같은 전처리를 하시려면 category feature는 제외하고 전처리를 진행하시고, 그 후에 category feature를 다시 붙여주세요.

   - HistGradientBoostingRegressor에서는 tree의 갯수를 정해주기 위해서 'n_estimators' 대신 'max_iter'라는 이름의 parameter를 사용합니다.

3. **[Optional]** 구, 시간대, 월과 같은 feature를 예측에 더욱 더 효과적으로 활용하기 위해서 data.seoul.go.kr에 있는 서울시 공공데이터셋을 활용해도 됩니다. 아래 Reference에 사용 가능한 데이터의 예시들이 있습니다. 다만 예측에 별로 도움이 되지 않는 데이터도 있을 수 있으니 잘 살펴보시고 사용해주세요.

   ※ **[Optional]** 부분들은 과제 평가 항목에서 제외됩니다.

[Reference]

- https://scikit-learn.org/stable/modules/tree.html
- https://scikit-learn.org/stable/modules/ensemble.html
- https://scikit-learn.org/stable/unsupervised_learning.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html
- 서울시 차량통행속도 (구별/월별) 통계
- 서울시 승용차통행속도 (시간대별) 통계
- 서울시 자동차등록 (월별/구별) 통계