Model Creation

April 29, 2023

1 Counterfactual Model Development for Energy Consumption Estimation

[ ]:
```
%load_ext autoreload
%autoreload 2
```

[ ]:
```
import math
import numpy as np
import pandas as pd
import lightgbm as lgb

from tqdm import tqdm
from xgboost import XGBRegressor
from sklearn.preprocessing import normalize
from sklearn.preprocessing import MinMaxScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import mean_squared_error, mean_absolute_error,␣
↪mean_absolute_percentage_error, r2_score, mean_squared_log_error

from utils import styled_print
```

[ ]:
```
train_df = pd.read_feather('../data/x_train.ftr')
validation_df = pd.read_feather('../data/x_validation.ftr')
```

[ ]:
```
styled_print("Training Dataset Summary", header=True)
styled_print(f"The shape of train_df is {train_df.shape}")
styled_print(f"The columns in train_df are {list(train_df.columns)}")
```

[ ]:
```
styled_print("Validation Dataset Summary", header=True)
styled_print(f"The shape of validation_df is {validation_df.shape}")
styled_print(f"The columns in validation_df are {list(validation_df.columns)}")
```

```
[ ]: def evaluate_model(y_true, y_pred, model_desc="ASHRAE Model", antilog=True):
         if antilog:
             y_true = np.exp(y_true)
             y_pred = np.exp(y_pred)
             rmsle = math.sqrt(mean_squared_log_error(y_true, y_pred))
         else:
             mlse = mean_squared_error(y_true, y_pred)
             rmsle = math.sqrt(mlse)

         mae = mean_absolute_error(y_true, y_pred)
         mape = mean_absolute_percentage_error(y_true, y_pred)
         r2 = r2_score(y_true, y_pred)
         styled_print(f"Evaluation of {model_desc}", header=True)
         styled_print(f"R2 Score: {r2}")
         styled_print(f"Mean Absolute Error: {mae}")
         styled_print(f"Mean Absolute Percentage Error: {mape}")
         styled_print(f"Root Mean Square Logarithmic Error: {rmsle}")
```

1.1 Baseline Model

As first step we create a baseline model, where we predict the `mean` value based on group by `primary_use` and `meter_type`.

```
[ ]: y_pred_baseline = train_df.groupby(['primary_use',␣
      ↪'meter_type'])['log_meter_reading'].mean().reset_index()
     y_pred_baseline.rename(columns={"log_meter_reading": "y_pred_baseline"},␣
      ↪inplace=True)
```

```
[ ]: temp_train_df = train_df.copy()
     temp_validation_df = validation_df.copy()
```

```
[ ]: temp_train_df = temp_train_df.merge(y_pred_baseline, on=['primary_use',␣
      ↪'meter_type'], how='left')
     temp_validation_df = temp_validation_df.merge(y_pred_baseline,␣
      ↪on=['primary_use', 'meter_type'], how='left')
```

```
[ ]: evaluate_model(
         temp_train_df['log_meter_reading'],
         temp_train_df['y_pred_baseline'],
         model_desc="Baseline Model - Training Set",
         antilog=True
     )
```

```
[ ]: evaluate_model(
         temp_validation_df['log_meter_reading'],
         temp_validation_df['y_pred_baseline'],
         model_desc="Baseline Model - Validation Set",
```

```
    antilog=True
)
```

As expected our baseline model does very poor on training and validation set. Let's try Decision Tree model as next step.

1.2 Prepare Dataset

```
[ ]: y_train = train_df['log_meter_reading']
     y_validation = validation_df['log_meter_reading']

     x_train = train_df.drop(['log_meter_reading', 'index'], axis=1)
     x_validation = validation_df.drop(['log_meter_reading', 'index'], axis=1)
```

```
[ ]: primary_use_enc = LabelEncoder().fit(x_train['primary_use'])
     season_enc = LabelEncoder().fit(x_train['season'])
     meter_type_enc = LabelEncoder().fit(x_train['meter_type'])
```

```
[ ]: x_train['season'] = season_enc.transform(x_train['season'])
     x_validation['season'] = season_enc.transform(x_validation['season'])
```

```
[ ]: x_train['primary_use'] = primary_use_enc.transform(x_train['primary_use'])
     x_validation['primary_use'] = primary_use_enc.
      ↪transform(x_validation['primary_use'])
```

```
[ ]: x_train['meter_type'] = meter_type_enc.transform(x_train['meter_type'])
     x_validation['meter_type'] = meter_type_enc.
      ↪transform(x_validation['meter_type'])
```

```
[ ]: scaler = MinMaxScaler()
     scaler.fit(x_train)
     x_train = pd.DataFrame(scaler.transform(x_train), columns = x_train.columns)
     x_validation = pd.DataFrame(scaler.transform(x_validation), columns =␣
      ↪x_validation.columns)
```

```
[ ]: x_train.head()
```

```
[ ]: x_validation.head()
```

1.3 Decision Tree

```
[ ]: # Create a decision tree classifier object
     dt_reg = DecisionTreeRegressor()

     # Define the hyperparameter grid for the decision tree
     # params = {
     #     'max_depth': [4, 5, 6, 7, 8, 9, 10, 15, None],
```

```
#     'min_samples_split': [2, 3, 4],
#     'min_samples_leaf': [1, 2, 3, 4, 5]
# }

params = {
    'max_depth': [15]
}

# Create a GridSearchCV object and fit it to the training data
grid_search = GridSearchCV(estimator=dt_reg, param_grid=params, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)

# Print the best hyperparameters and the corresponding mean cross-validated␣
 ↪score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

[ ]:
```
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
y_pred_train = best_model.predict(x_train)
y_pred_validation = best_model.predict(x_validation)
```

[ ]:
```
evaluate_model(
    y_train, y_pred_train,
    model_desc="Decision Tree - Training Set",
    antilog=True
)
```

[ ]:
```
evaluate_model(
    y_validation, y_pred_validation,
    model_desc="Decision Tree - Validation Set",
    antilog=True
)
```

1.4 Random Forest

[ ]:
```
# params = {
#     'n_estimators': [50, 100, 200],
#     'max_depth': [3, 5, None],
#     'min_samples_split': [2, 5, 10],
#     'min_samples_leaf': [1, 2, 4],
#     'criterion': ['gini', 'entropy'],
# }

params = {
```

```
    'n_estimators': [200],
    'criterion': ['gini', 'entropy'],
}

# Create a Random Forest classifier
rfc = RandomForestRegressor(random_state=42)

# Create a GridSearchCV object
grid_search = GridSearchCV(rfc, params, cv=5, n_jobs=-1)

# Fit the GridSearchCV object to the data
grid_search.fit(x_train, y_train)

# Print the best hyperparameters and the corresponding mean cross-validated␣
 ↪score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

[ ]:
```
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
y_pred_train = best_model.predict(x_train)
y_pred_validation = best_model.predict(x_validation)
```

[ ]:
```
evaluate_model(
    y_train, y_pred_train,
    model_desc="Random Forest - Training Set",
    antilog=True
)
```

[ ]:
```
evaluate_model(
    y_validation, y_pred_validation,
    model_desc="Random Forest - Validation Set",
    antilog=True
)
```

1.5 Gradient Boosted Machines

[ ]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

[ ]:
```
# Define the parameter grid to search over
params = {
    "learning_rate": [0.01, 0.1, 1],
    "n_estimators": [100, 500, 1000],
    "max_depth": [3, 5, 7],
}
```

```
# Create a gradient boosting regressor
gb_regressor = GradientBoostingRegressor()

# Create a GridSearchCV object
grid_search = GridSearchCV(gb_regressor, params, cv=5, n_jobs=-1)

# Fit the GridSearchCV object to the data
grid_search.fit(x_train, y_train)

# Print the best hyperparameters and the corresponding mean cross-validated␣
↪score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

[ ]:
```
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
y_pred_train = best_model.predict(x_train)
y_pred_validation = best_model.predict(x_validation)
```

[ ]:
```
evaluate_model(
    y_train, y_pred_train,
    model_desc="Gradient Boosted Machines - Training Set",
    antilog=True
)
```

[ ]:
```
evaluate_model(
    y_validation, y_pred_validation,
    model_desc="Gradient Boosted Machines - Validation Set",
    antilog=True
)
```

1.6 Neural Networks

[ ]:
```
from keras.wrappers.scikit_learn import KerasRegressor
from keras.layers import Dense
from keras import Sequential

# Define the model function
def create_model():
    model = Sequential()
    model.add(Dense(64, input_dim=X.shape[1], activation="relu"))
    model.add(Dense(32, activation="relu"))
    model.add(Dense(1, activation="linear"))
    model.compile(loss="mean_squared_error", optimizer="adam")
```

```
    return model

# Create the KerasRegressor model
model = KerasRegressor(build_fn=create_model)

# Define the hyperparameters to tune
parameters = {
    'batch_size': [16, 32],
    'epochs': [50, 100],
    'optimizer': ['adam', 'rmsprop']
}

# Create the GridSearchCV object
grid = GridSearchCV(estimator=model, param_grid=parameters, cv=5, n_jobs=-1)

# Train the model using GridSearchCV
grid.fit(x_train, y_train)

# Print the best hyperparameters and the corresponding mean cross-validated␣
 ↪score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

[ ]:
```
# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict on the test set using the best model
y_pred_train = best_model.predict(x_train)
y_pred_validation = best_model.predict(x_validation)
```

[ ]:
```
evaluate_model(
    y_train, y_pred_train,
    model_desc="Neural Networks - Training Set",
    antilog=True
)
```

[ ]:
```
evaluate_model(
    y_validation, y_pred_validation,
    model_desc="Neural Networks - Validation Set",
    antilog=True
)
```