XGBoost

In [ ]:
```
# lets import all the dependencies
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
plt.style.use('ggplot')
import pandas as pd
import numpy as np

import seaborn as sns
sns.set()

from sklearn.model_selection import train_test_split, KFold
from sklearn.model_selection import GridSearchCV, cross_val_score, learning_curv
from sklearn.model_selection import validation_curve

import xgboost as xgb

from sklearn.metrics import mean_absolute_error, r2_score
from joblib import load, dump
```

In [ ]:
```
# loading the data
df = pd.read_csv("./Data/data.csv",sep=",")
df.drop(['Unnamed: 0'], axis=1, inplace=True) # There were some formatting issue
                                              # writing the csv
```

In [ ]:
```
df.head()
```

Out[ ]:

	DISTRICT	UPAZILA	STATION_ID	STATION_NAME	DATE	RAIN_FALL(mm)	LATITUDE	LONGITUDI
0	Bandarban	Lama	CL317	Lama	01-jan-2017	0.0	21.81	92.1
1	Bandarban	Lama	CL317	Lama	02-jan-2017	0.0	21.81	92.1
2	Bandarban	Lama	CL317	Lama	03-jan-2017	0.0	21.81	92.1
3	Bandarban	Lama	CL317	Lama	04-jan-2017	0.0	21.81	92.1
4	Bandarban	Lama	CL317	Lama	05-jan-2017	0.0	21.81	92.1

Defining our X and y

In [ ]:
```
```

```
X = df['RAIN_FALL(mm)'].values.reshape(-1,1) # input feature
y = df['WATER_LEVEL(m)'].values.reshape(-1,1) # target feature
```

In [ ]:
```
X.shape, y.shape
```

Out[ ]:
```
((1826, 1), (1826, 1))
```

Making the train test split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X,y, test_size=0.2, random_state=17, shuffle=True
)
```

Model Building

Initialize the CV

In [ ]:
```
kfold = KFold(n_splits=5, shuffle=True, random_state=17)
```

Initialize the XGB with default parameters!

The values of default parameters can be found here:
https://xgboost.readthedocs.io/en/stable/parameter.html

Im using Xgboost sklearn wrapper here so that code structure doesnt change much!

In [ ]:
```
xgb_model = xgb.XGBRegressor()
```

As usual check the CV score first

In [ ]:
```
results = cross_val_score(
    xgb_model,
    X_train,
    y_train,
    cv=kfold,
    scoring='neg_mean_absolute_error'
)
-results.mean()
```

Out[ ]:
```
0.46924084368144003
```

checking for train-test accuracy

In [ ]:
```
xgb_model.fit(X_train,y_train)
```

Out[ ]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=6,
```

```
                num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
                reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
                validate_parameters=1, verbosity=None)
```

In [ ]:
```
# accuracy on the train set
knn_pred = xgb_model.predict(X_train)
mean_absolute_error(y_train,knn_pred)
```

Out[ ]:
```
0.35532010421230364
```

In [ ]:
```
# accuracy on the test set
knn_pred = xgb_model.predict(X_test)
mean_absolute_error(y_test,knn_pred)
```

Out[ ]:
```
0.4707051377530957
```

Note:

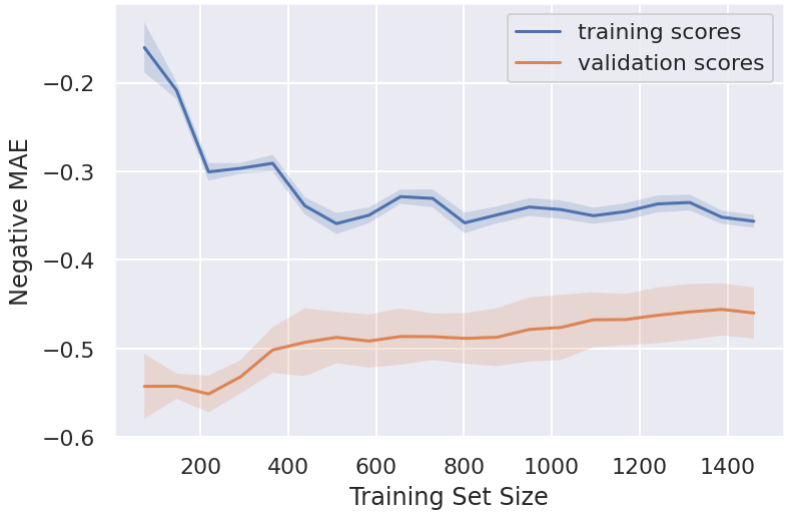
- seems like theres significant overfitting happening here
- Lets plot the learning curve first

Learning Curve

In [ ]:
```
# Helper function
alphas = np.logspace(-2, 0, 20)
def plot_with_err(x, data, **kwargs):
    mu, std = data.mean(1), data.std(1)
    lines = plt.plot(x, mu, "-", **kwargs)
    plt.fill_between(
        x,
        mu - std,
        mu + std,
        edgecolor="none",
        facecolor=lines[0].get_color(),
        alpha=0.2,
    )
```

In [ ]:
```
def plot_learning_curve():
    train_sizes = np.linspace(0.05,1,20)
    N_train, val_train, val_test = learning_curve(
        xgb_model,X,y, train_sizes=train_sizes,cv=kfold,scoring='neg_mean_absolu
    )
    plot_with_err(N_train, val_train, label="training scores")
    plot_with_err(N_train, val_test, label="validation scores")
    plt.xlabel("Training Set Size")
    plt.ylabel("Negative MAE")
    plt.legend()
    plt.grid(True);
```

In [ ]:
```
plot_learning_curve()
```



Note:

- two curves havent converged yet! So theres rooms for improvement!

In [ ]:
```
xgb_model.get_params
```

Out[ ]:
```
<bound method XGBModel.get_params of XGBRegressor(base_score=0.5, booster='gbtre
e', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
             gamma=0, gpu_id=-1, importance_type=None,
             interaction_constraints='', learning_rate=0.300000012,
             max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
             monotone_constraints='()', n_estimators=100, n_jobs=6,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)>
```

Validation Curves

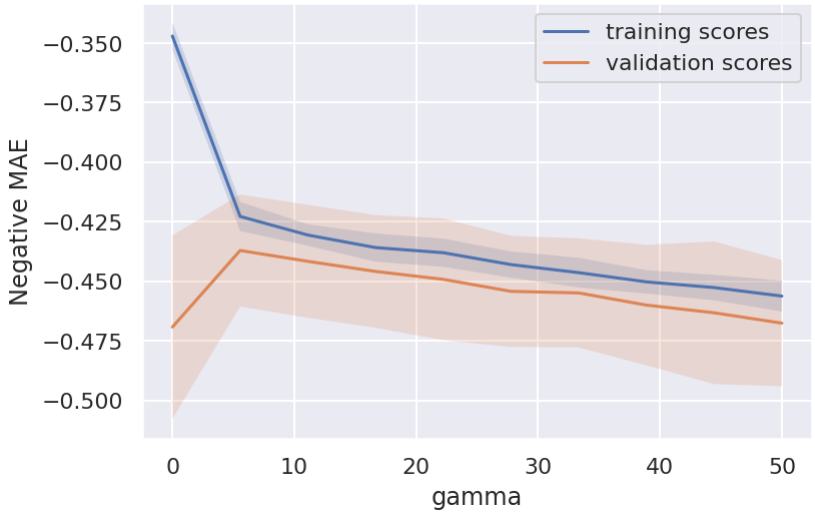
In [ ]:
```
# helper function
def plot_with_err(x, data, **kwargs):
    mu, std = data.mean(1), data.std(1)
    lines = plt.plot(x, mu, "-", **kwargs)
    plt.fill_between(
        x,
        mu - std,
        mu + std,
        edgecolor="none",
        facecolor=lines[0].get_color(),
        alpha=0.2,
    )
```

In [ ]:
```
def plot_validation_curve(param_grid,param,estimator):
    val_train, val_test = validation_curve(
    estimator=estimator,
    X=X_train,
    y=y_train,
    param_name=param,
    param_range=param_grid,
    cv=kfold,
```

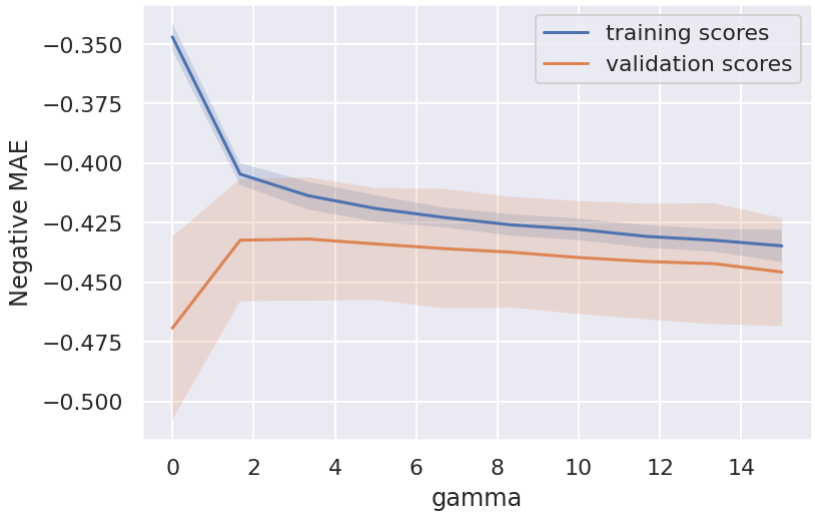
```
    scoring="neg_mean_absolute_error",
    n_jobs=-1
    )

    plot_with_err(param_grid, val_train, label="training scores")
    plot_with_err(param_grid, val_test, label="validation scores")
    plt.xlabel(param)
    plt.ylabel("Negative MAE")
    plt.legend()
    plt.grid(True);
```

- **gamma:**

In [ ]:
```
plot_validation_curve(np.linspace(0,50,10),'gamma',xgb_model)
```



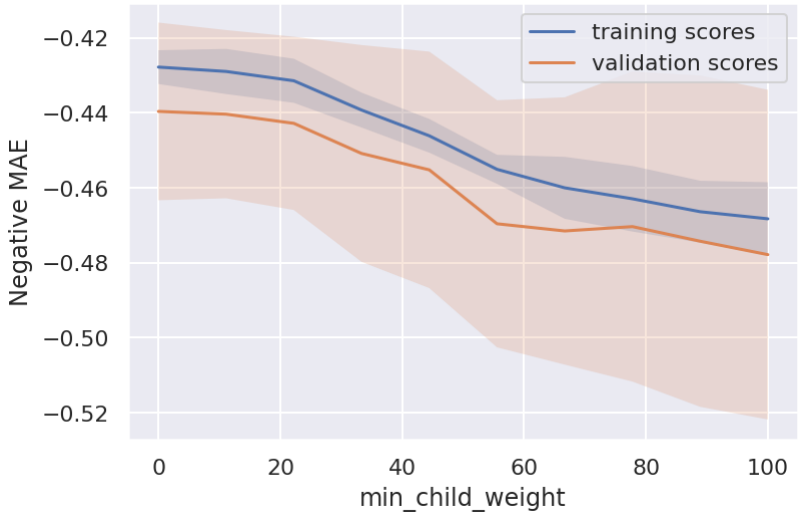
In [ ]:
```
plot_validation_curve(np.linspace(0,15,10),'gamma',xgb_model)
```



- **min_child_weight:**

In [ ]:
```
xgb_model = xgb.XGBRegressor(gamma = 10)
```

In [ ]:
```
plot_validation_curve(np.linspace(0,100,10),'min_child_weight',xgb_model)
```



- **subsample:**

In [ ]:
```
plot_validation_curve([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1],'subsample',xgb_mo
```

HyperParameter Tunings

In [ ]:
```
xgbGrid = {
    'max_depth':[5],
    'gamma':[4,5,6],
    'min_child_weight': [0,1],
    'subsample':[0.1,0.2]
}

xgb_model = xgb.XGBRegressor()
gcv = GridSearchCV(xgb_model,xgbGrid,n_jobs=-1,cv=kfold,verbose=1,\
    scoring='neg_mean_absolute_error')
```

In [ ]:
```
gcv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=17, shuffle=True),
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None,
                                    enable_categorical=False, gamma=None,
                                    gpu_id=None, importance_type=None,
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight...
                                    n_estimators=100, n_jobs=None,
                                    num_parallel_tree=None, predictor=None,
                                    random_state=None, reg_alpha=None,
                                    reg_lambda=None, scale_pos_weight=None,
                                    subsample=None, tree_method=None,
                                    validate_parameters=None, verbosity=None),
             n_jobs=-1,
             param_grid={'gamma': [4, 5, 6], 'max_depth': [5],
                         'min_child_weight': [0, 1], 'subsample': [0.1, 0.2]},
             scoring='neg_mean_absolute_error', verbose=1)
```

In [ ]:
```
gcv.best_score_
```

Out[ ]:
```
-0.4359201454580647
```

In [ ]:
```
gcv.best_params_
```

Out[ ]:
```
{'gamma': 4, 'max_depth': 5, 'min_child_weight': 0, 'subsample': 0.2}
```

In [ ]:
```
# accuracy on train set
pred = gcv.best_estimator_.predict(X_train)
mean_absolute_error(y_train,pred)
```

Out[ ]:
```
0.4100284295487077
```

In [ ]:
```
# accuracy on the test set
pred = gcv.best_estimator_.predict(X_test)
mean_absolute_error(y_test,pred)
```

Out[ ]:
```
0.43212858450217323
```

Result Dataframe

In [ ]:
```
result_df = pd.DataFrame()
result_df['Actual River Level(m)'] = pd.Series(y_test.ravel())
result_df['Predicted'] = gcv.best_estimator_.predict(pd.Series(X_test.ravel()))
```

In [ ]:
```
result_df.head(50)
```

Out[ ]:

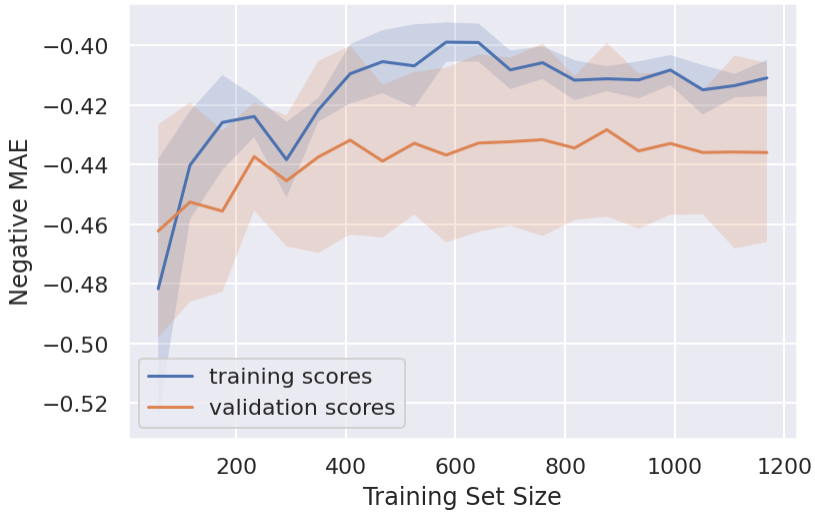
	Actual River Level(m)	Predicted
0	6.020	6.386051
1	6.230	6.386051
2	6.090	6.386051
3	6.210	6.386051
4	6.190	6.386051
5	5.950	6.386051
6	9.772	7.647743
7	6.270	6.386051
8	7.268	7.220532
9	6.180	6.386051
10	6.888	7.057232
11	6.220	6.386051
12	5.990	6.386051
13	7.230	6.386051
14	6.260	6.386051
15	6.280	6.386051
16	6.060	6.386051
17	7.394	6.795314
18	6.444	6.386051
19	7.264	6.957291
20	6.080	6.386051
21	6.260	6.386051
22	5.860	6.386051
23	6.100	6.386051
24	6.252	6.386051
25	5.870	6.386051
26	6.110	6.386051
27	6.402	7.518615
28	6.120	6.386051
29	7.668	7.135021
30	6.420	6.386051
31	6.160	6.386051
32	6.220	6.386051
33	7.156	7.135021
34	6.790	6.846089
35	7.600	7.057232

	Actual River Level(m)	Predicted
36	7.326	7.135021
37	7.316	6.386051
38	5.982	7.135021
39	6.170	6.386051
40	6.380	6.386051
41	6.220	6.386051
42	5.960	6.386051
43	6.200	6.386051
44	5.930	6.386051
45	6.190	6.386051
46	6.876	6.386051
47	7.002	6.795314
48	8.336	7.526973
49	7.762	7.135021

Learning Curve

In [ ]:
```
def plot_learning_curve():
    train_sizes = np.linspace(0.05,1,20)
    N_train, val_train, val_test = learning_curve(
        gcv.best_estimator_,X_train,y_train, train_sizes=train_sizes,cv=kfold,sc
    )
    plot_with_err(N_train, val_train, label="training scores")
    plot_with_err(N_train, val_test, label="validation scores")
    plt.xlabel("Training Set Size")
    plt.ylabel("Negative MAE")
    plt.legend()
    plt.grid(True);
```

In [ ]:
```
plot_learning_curve()
```



Saving the model

In [ ]:
```
dump(gcv.best_estimator_,'./Saved Model/XGBoost.joblib')
```

Out[ ]:
```
['./Saved Model/XGBoost.joblib']
```