# Estimation of Suspended Sediment Concentration using Remote Sensing Data

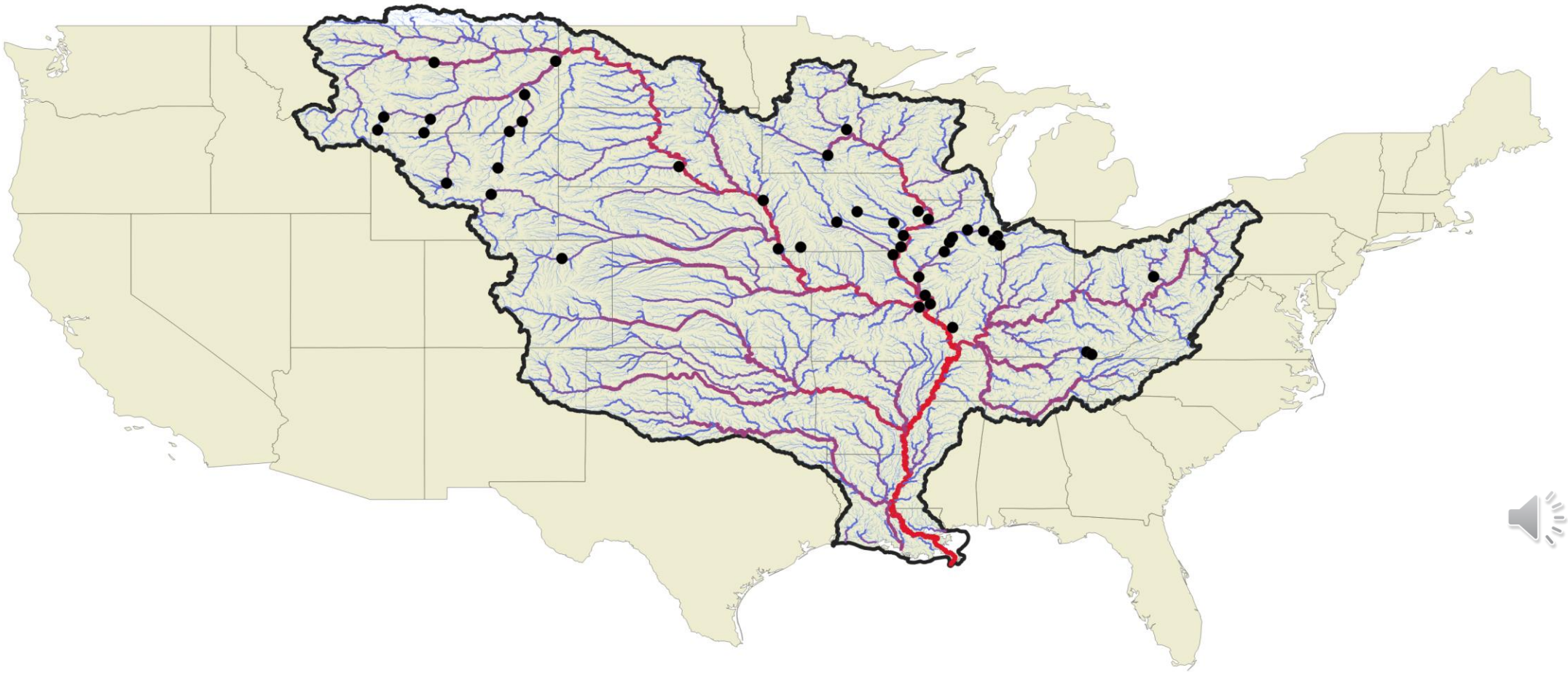
**Abhinav Gupta**

**18 May 2025**

**Make sure to download the data (dataLSTM_SR.txt) and the Jupyter notebook (Xgboost_tutorial.ipynb) for this tutorial**

# Suspended Sediment Concentration (SSC)

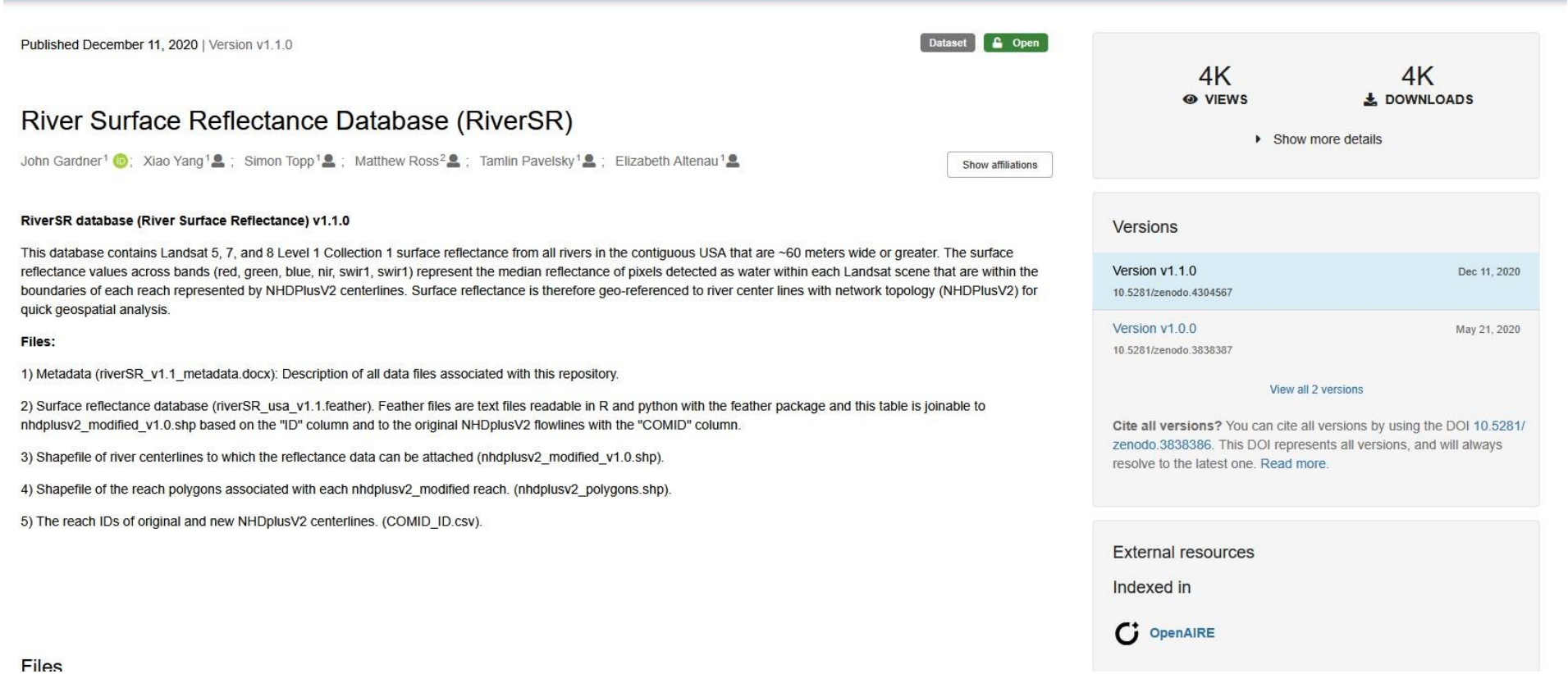
**Mississippi River Basin - 43 USGS stations**



# Remote Sensing (RS) data

Landsat data

https://zenodo.org/records/3838387



Published December 11, 2020 | Version v1.1.0

Dataset Open

## River Surface Reflectance Database (RiverSR)

John Gardner[1]; Xiao Yang[1]; Simon Topp[1]; Matthew Ross[2]; Tamlin Pavelsky[1]; Elizabeth Altenau[1]

Show affiliations

**RiverSR database (River Surface Reflectance) v1.1.0**

This database contains Landsat 5, 7, and 8 Level 1 Collection 1 surface reflectance from all rivers in the contiguous USA that are ~60 meters wide or greater. The surface reflectance values across bands (red, green, blue, nir, swir1, swir1) represent the median reflectance of pixels detected as water within each Landsat scene that are within the boundaries of each reach represented by NHDPlusV2 centerlines. Surface reflectance is therefore geo-referenced to river center lines with network topology (NHDPlusV2) for quick geospatial analysis.

**Files:**

1) Metadata (riverSR_v1.1_metadata.docx): Description of all data files associated with this repository.

2) Surface reflectance database (riverSR_usa_v1.1.feather). Feather files are text files readable in R and python with the feather package and this table is joinable to nhdplusv2_modified_v1.0.shp based on the "ID" column and to the original NHDplusV2 flowlines with the "COMID" column.

3) Shapefile of river centerlines to which the reflectance data can be attached (nhdplusv2_modified_v1.0.shp).

4) Shapefile of the reach polygons associated with each nhdplusv2_modified reach. (nhdplusv2_polygons.shp).

5) The reach IDs of original and new NHDplusV2 centerlines. (COMID_ID.csv).

Files

4K VIEWS 4K DOWNLOADS

Show more details

Versions

Version v1.1.0 Dec 11, 2020
10.5281/zenodo.4304567

Version v1.0.0 May 21, 2020
10.5281/zenodo.3838387

View all 2 versions

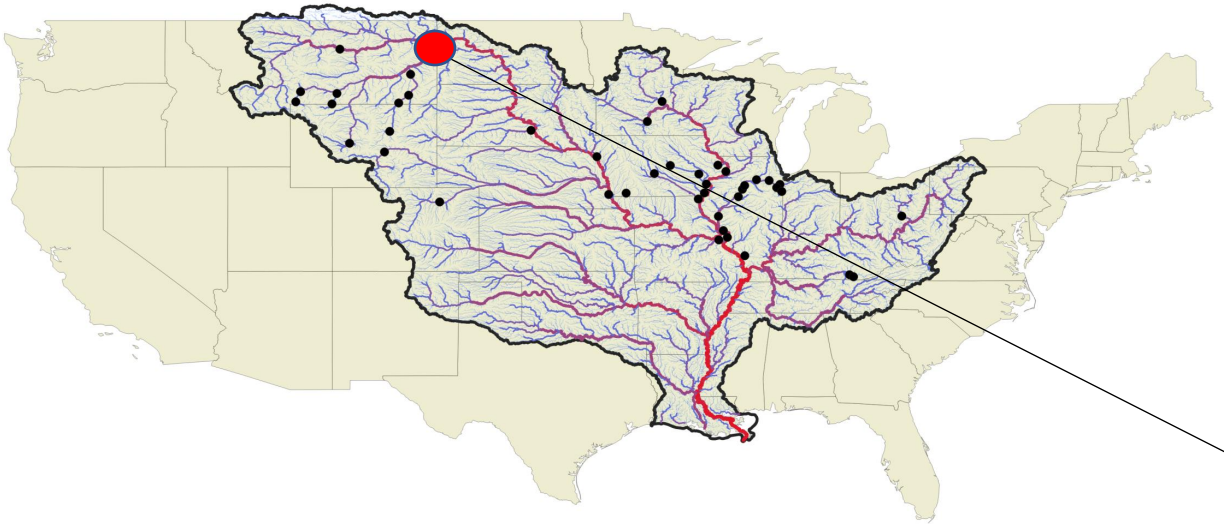
**Cite all versions?** You can cite all versions by using the DOI 10.5281/zenodo.3838386. This DOI represents all versions, and will always resolve to the latest one. Read more.

External resources

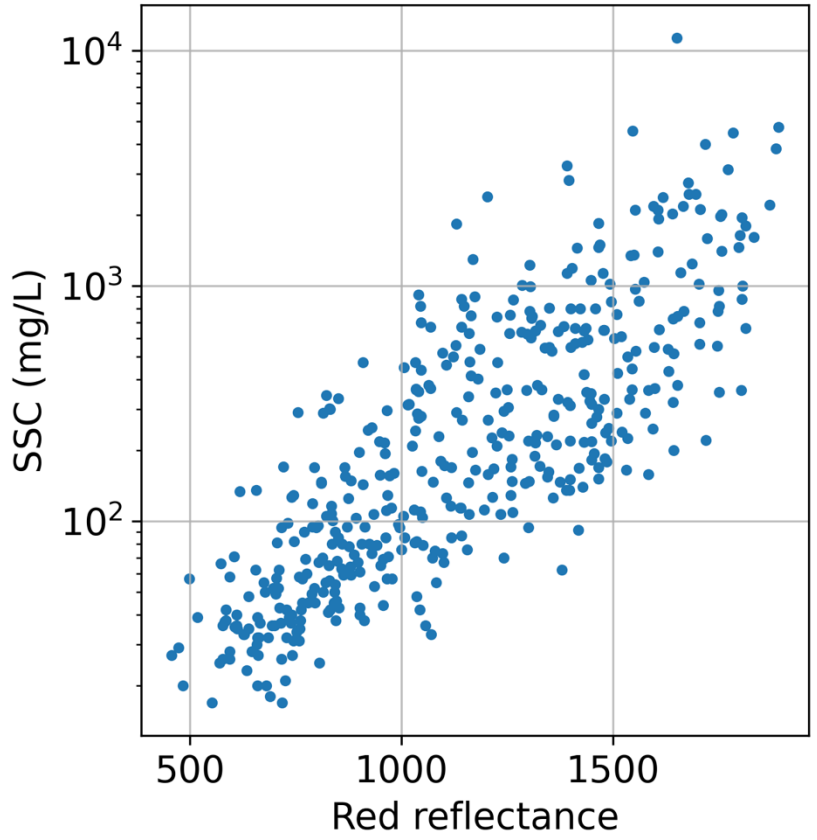
Indexed in

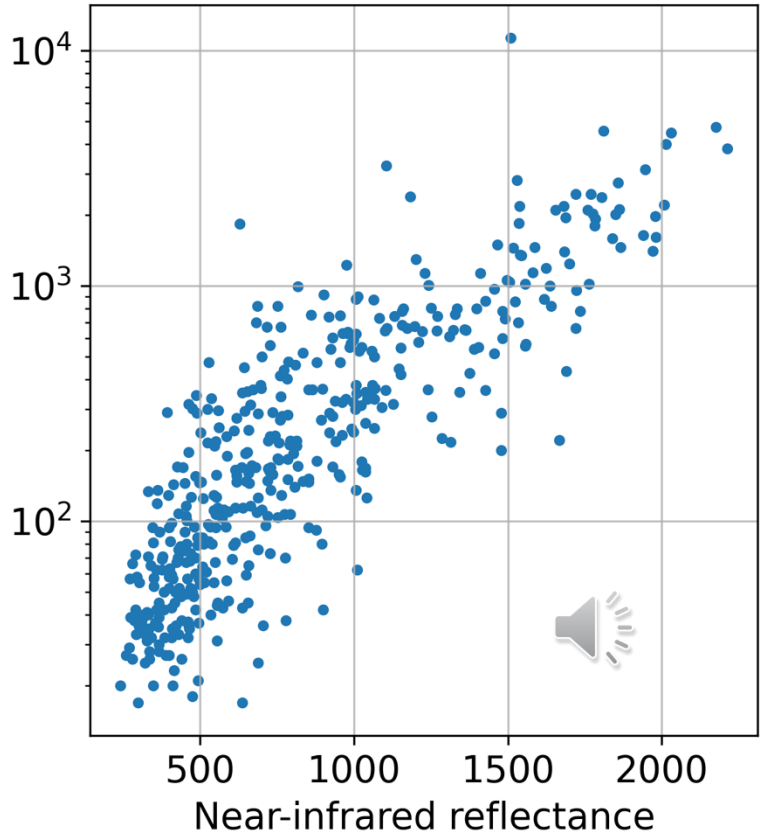
OpenAIRE

# Relationship between RS and SSC data



**Yellowstone River, Montana**



# Extreme gradient boosting (Xgboost)

*The Annals of Statistics*
2001, Vol. 29, No. 5, 1189–1232

## 1999 REITZ LECTURE

## GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE[1]

BY JEROME H. FRIEDMAN

*Stanford University*

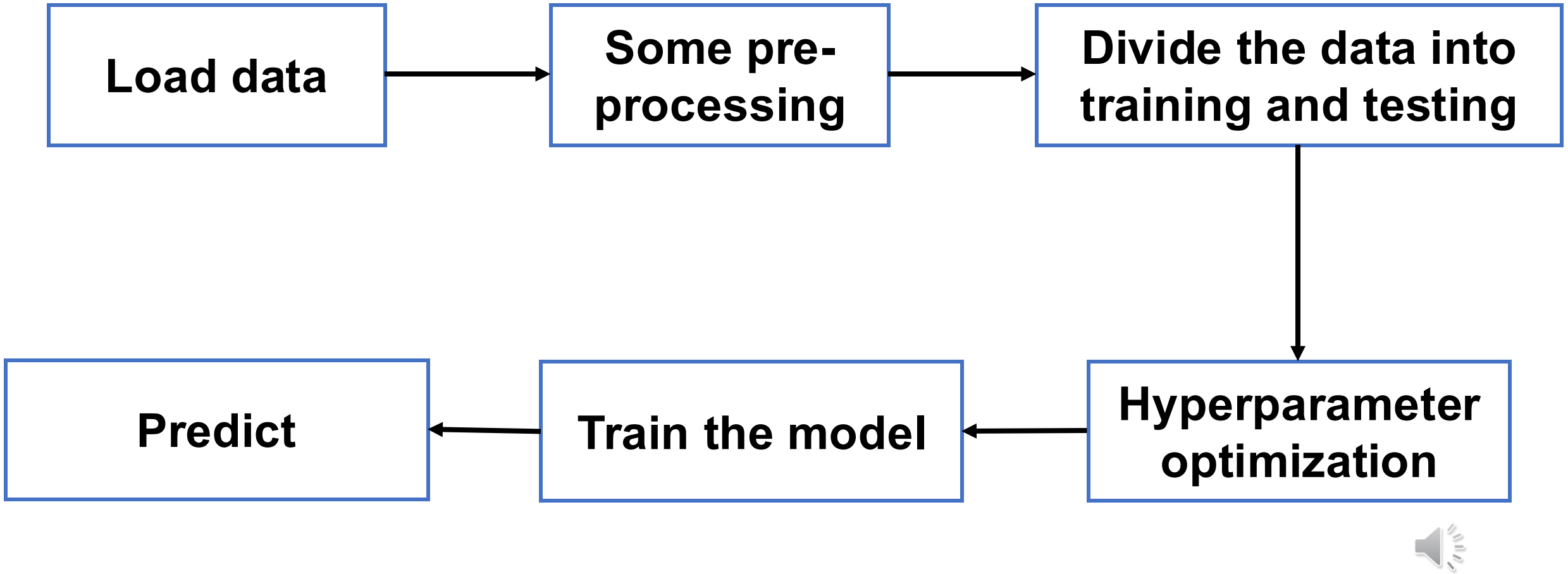
Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest-descent minimization. A general gradient descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least absolute deviation, and Huber-$M$ loss functions for regression, and multiclass logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire and Friedman, Hastie and Tibshirani are discussed.

$$F(x) = f_1(x) + f_2(x) + \cdots + f_N(x)$$

The function $f_i(x)$ are estimated iteratively

Often, a decision tree is defined to estimate $f_i(x)$

# General pipeline



# Determination of hyperparameters

Boosting method = [‘gbtree’, ‘gblinear’]

Maximum tree depth = [2, 4, 6, 8, 10, 15, 20]

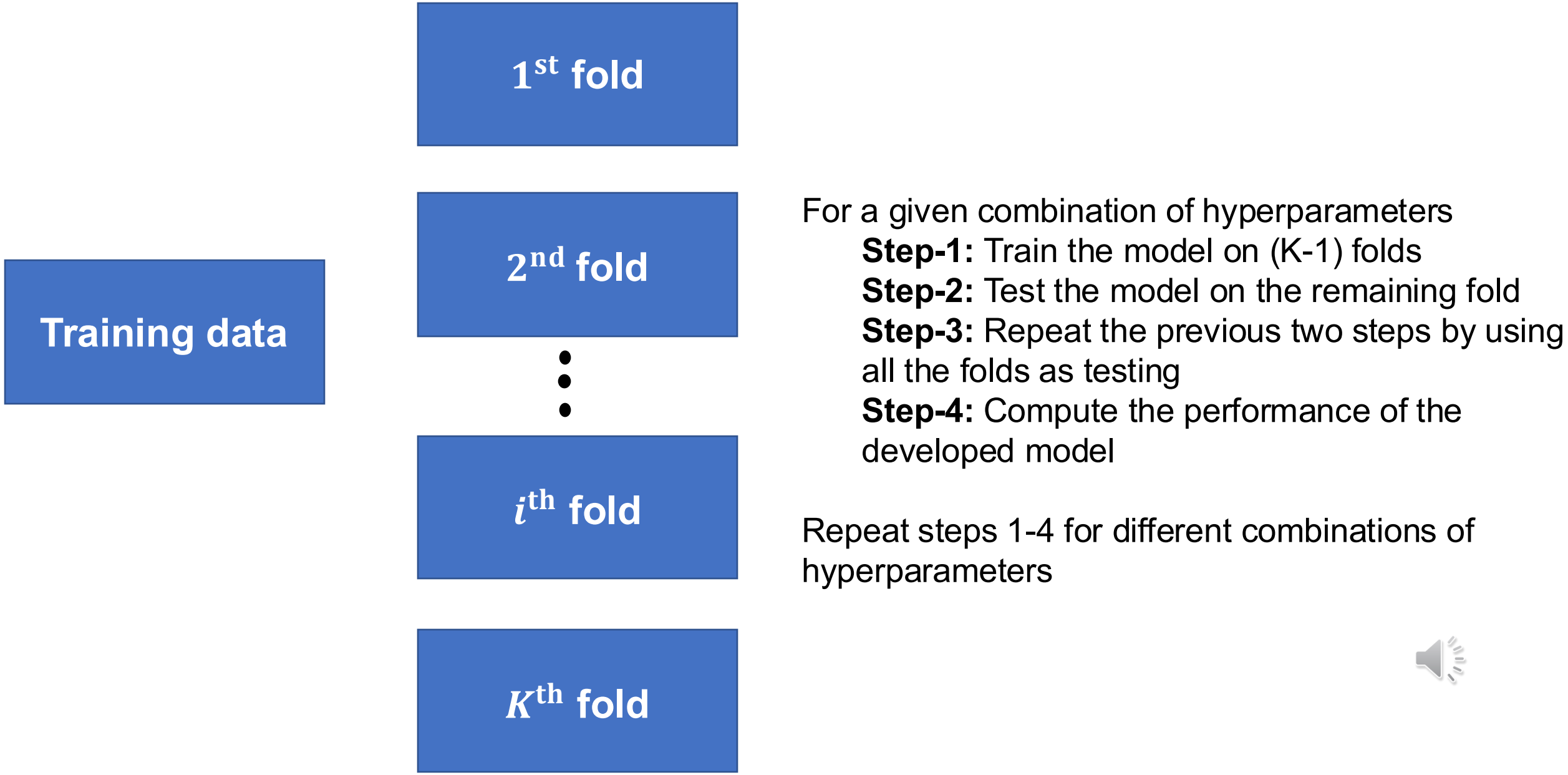
Number of iterations (estimators) = [50, 100, 200]

Shrinkage rate = [0.10, 0.30, 0.50, 0.80, 1.00]

Training subsample = [0.5, 0.75, 1.00]

$$\boldsymbol{F(x) = f_1(x) + f_2(x) + \cdots + f_N(x)}$$

# K-fold cross-validation




For a given combination of hyperparameters

**Step-1:** Train the model on (K-1) folds

**Step-2:** Test the model on the remaining fold

**Step-3:** Repeat the previous two steps by using all the folds as testing

**Step-4:** Compute the performance of the developed model

Repeat steps 1-4 for different combinations of hyperparameters

# K-fold cross-validation

Training data

$1^{st}$ fold

$2^{nd}$ fold

$i^{th}$ fold

$K^{th}$ fold

For a given combination of hyperparameters

**Step-1:** Train the model on (K-1) folds

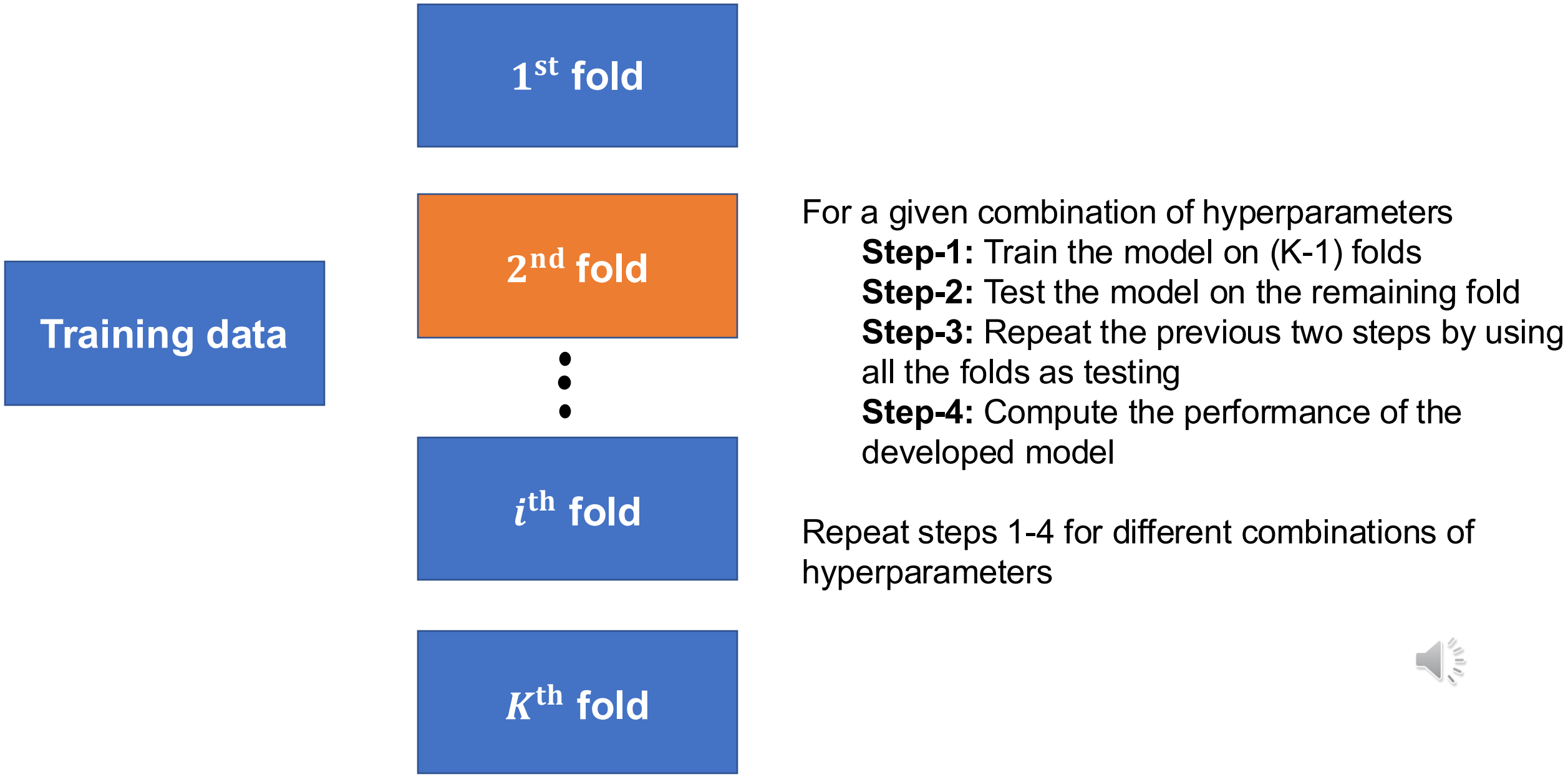
**Step-2:** Test the model on the remaining fold

**Step-3:** Repeat the previous two steps by using all the folds as testing

**Step-4:** Compute the performance of the developed model

Repeat steps 1-4 for different combinations of hyperparameters

# K-fold cross-validation




For a given combination of hyperparameters

**Step-1:** Train the model on (K-1) folds

**Step-2:** Test the model on the remaining fold

**Step-3:** Repeat the previous two steps by using all the folds as testing

**Step-4:** Compute the performance of the developed model

Repeat steps 1-4 for different combinations of hyperparameters

# Import libraries

```
import datetime
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xgboost as xgb
import multiprocessing
from sklearn.model_selection import GridSearchCV
```

**Please install these libraries in your environment in case you do not have any library**

**Example:** pip install sklearn

# Read data

```
main_dir = 'D:/Research/datasets/data_generated/LSTM_data/MS01/SR_data/EWRI_tutorial'
fname = 'dataLSTM_SR.txt'      # file containing data


filename = os.path.join(main_dir, fname)
data = pd.read_csv(filename, delimiter='\t’)

# Keep all the relevant data only
all_data = data[['COMID', 'Date', 'Obs_SSC', 'redR', 'greenR', 'blueR', 'nirR', 'swir1R',
'swir2R']]
```

**‘main_dir’** is the path to the directory containing the data textfile
**‘fname’** is the name of the textfile
**Define these two variable in your system**

**COMID** – Unique river ID
**Date** – Date of obsevrations
**Obs_SSC** - Observed suspended sediment concentration (from USGS)
**redR** – Red band reflectance

# Count the number of RS observations for each river

```
# Number of unique rivers and corresponding number of observations
COMIDs = all_data['COMID'].unique()
nonNaN_counts = all_data.groupby('COMID')['redR'].count()
```

43 rivers (COMIDs)
2-3 Landsat observations within a month

# Select one river and access its data

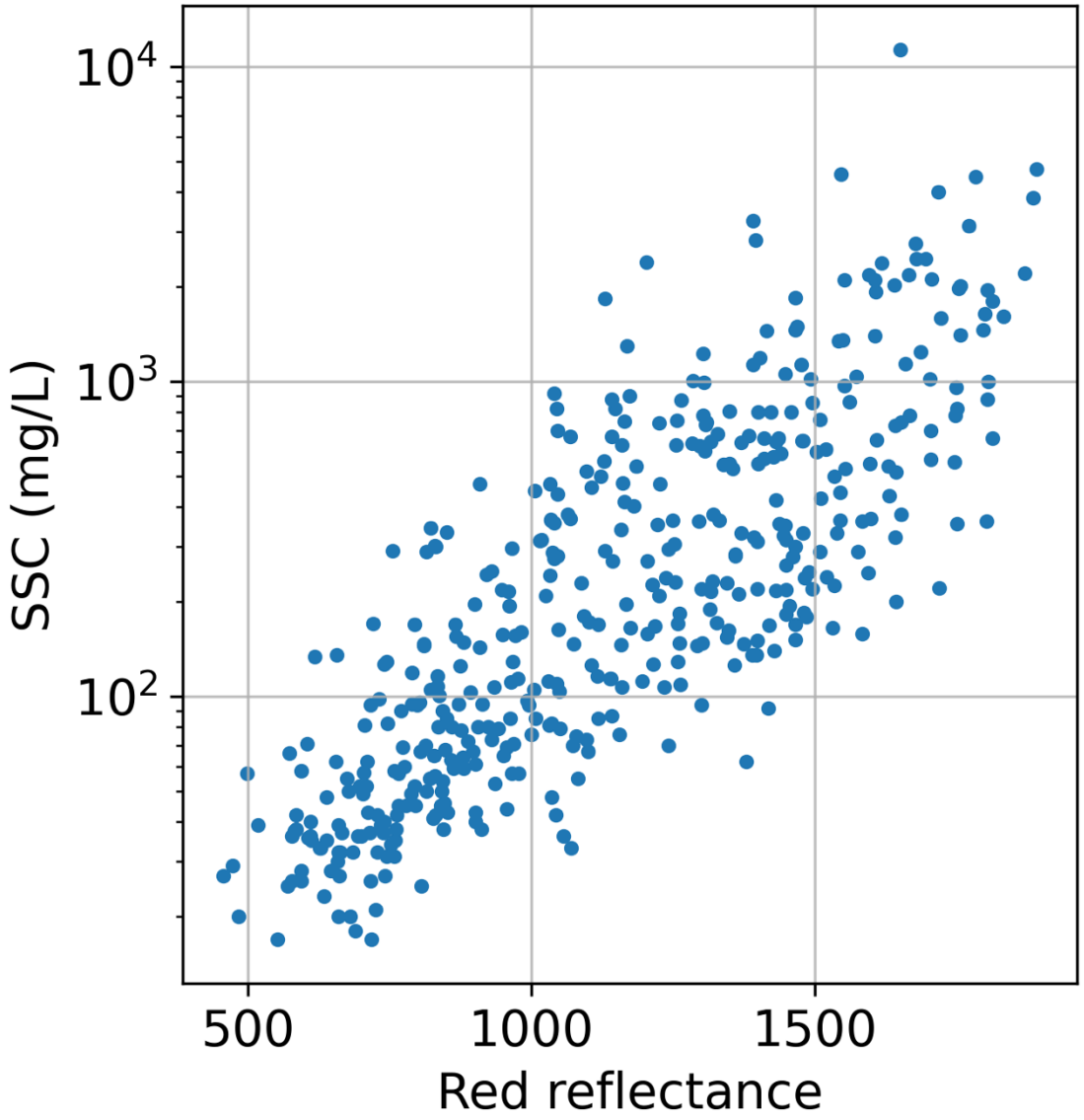
```
# identify the COMID with maximum number of remote sensing values
exp_comid = 74003095

# Extract the data for exp_comid
river_data = all_data[all_data['COMID']==exp_comid]
```

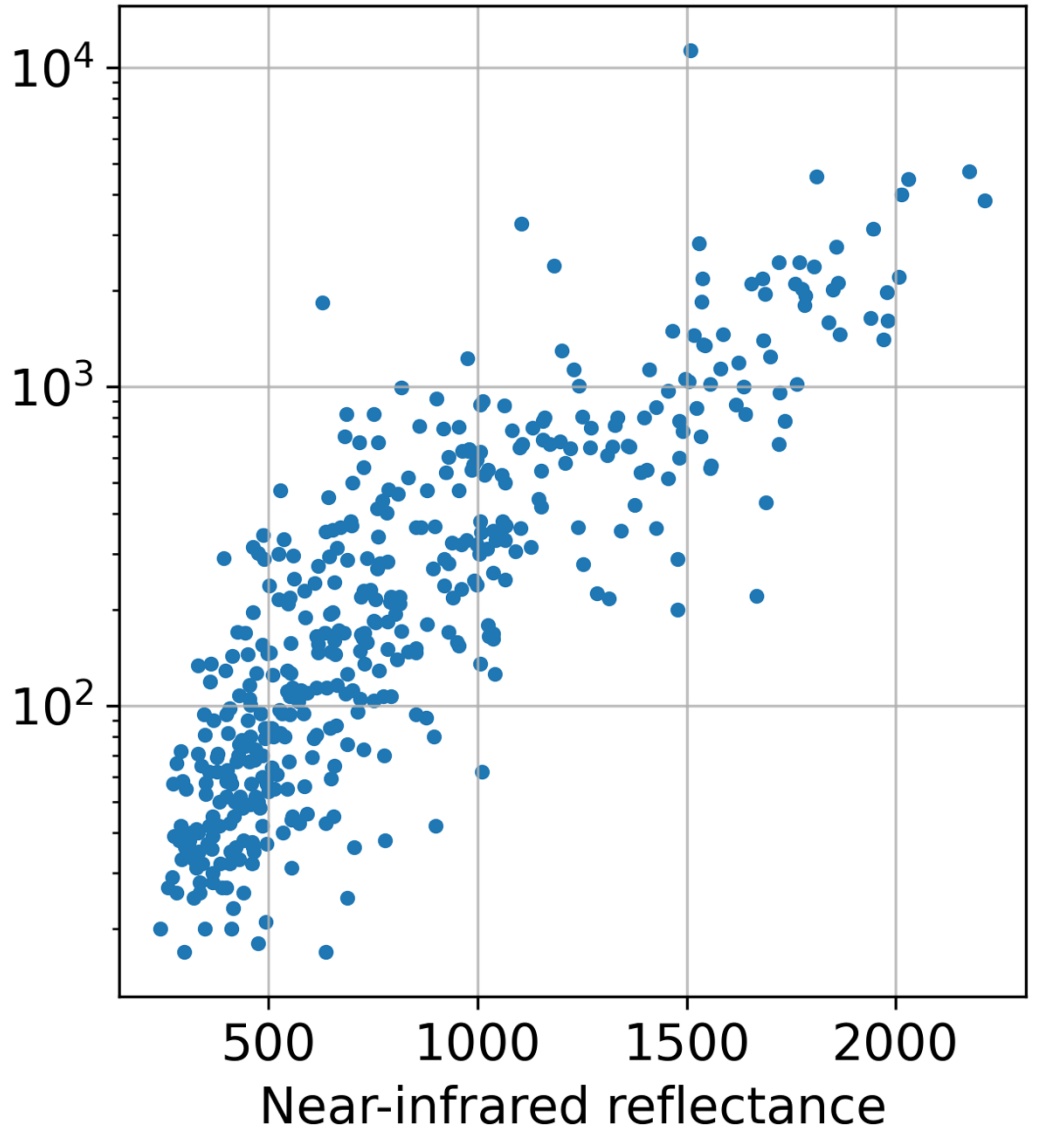
Select a COMID’ for example ‘74003095’ to develop the model

# Plot data

```
# Plot data
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 14
plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
plt.scatter(river_data['redR'], river_data['Obs_SSC'], s=10)
plt.yscale('log')
plt.xlabel('Red reflectance')
plt.ylabel('SSC (mg/L)')
plt.grid(alpha=0.8)

plt.subplot(1,2,2)
plt.scatter(river_data['nirR'], river_data['Obs_SSC'], s=10)
plt.yscale('log')
plt.xlabel('Near-infrared reflectance')
plt.grid(alpha=0.8)

plt.savefig(os.path.join(main_dir, 'reflectance_vs_SSC.tiff'), dpi=300)
```

# Results of plotting



# Remove the NaNs

```
# Indentify the rows containing RS data
indices = river_data['redR'].notna().to_numpy().nonzero()[0]

# Extract relevant data
final_data = river_data.iloc[indices]

# Now remove the rows containing NaN in place of obsSSC data
indices = final_data['Obs_SSC'].notna().to_numpy().nonzero()[0]
final_data = final_data.iloc[indices]
```

The SSC data are also missing for many rivers. This code block removes the days with missing SSC observations.

A total of 437 days on which both observed SSC and RS data are available

**final_data** is the variable containing the data that will be used for modeling

# Select training data

```
# Define parameters
trn_frac = 0.70
```

```
# Get training and test data
n = final_data.shape[0]
trn_last = int(trn_frac*n)
train_inds = range(0, trn_last)
test_inds = range(trn_last,n)

train_data, test_data = final_data.iloc[train_inds], final_data.iloc[test_inds]
ytrain, xtrain  = train_data['Obs_SSC'].values, train_data[['redR', 'greenR', 'blueR',
'nirR', 'swir1R', 'swir2R']].values
ytest, xtest  = test_data['Obs_SSC'].values, test_data[['redR', 'greenR', 'blueR',
'nirR', 'swir1R', 'swir2R']].values
```

Use some data to train the model and rest of it to independently test the model

**trn_frac** is the is the fraction of data that will be used for training

# Select training data

```
# Define parameters
trn_frac = 0.70
```

```
# Get training and test data
n = final_data.shape[0]
trn_last = int(trn_frac*n)
train_inds = range(0, trn_last)
test_inds = range(trn_last,n)

train_data, test_data = final_data.iloc[train_inds], final_data.iloc[test_inds]
ytrain, xtrain  = train_data['Obs_SSC'].values, train_data[['redR', 'greenR', 'blueR',
'nirR', 'swir1R', 'swir2R']].values
ytest, xtest  = test_data['Obs_SSC'].values, test_data[['redR', 'greenR', 'blueR',
'nirR', 'swir1R', 'swir2R']].values
```

**xtrain =** Training set predictor variables (remote sensing data)
**ytrain =** Training set target variables (SSC data)
**xtest =** Testing set predictor variables (remote sensing data)
**ytest =** Testing set target variables (SSC data)

# Function to compute NSE and implement cross-validation

```
# function to compute NSE
def computeNSE(obs, pred):
    sse = np.sum((obs - pred)**2)
    sst = np.sum((obs - np.mean(obs))**2)
    nse = 1 - sse/sst
    return nse

def XgboostParamTuning(X,y):
    if __name__ == "__main__":
        print("Parallel Parameter optimization")
        xgb_model = xgb.XGBRegressor(n_jobs=10, tree_method="hist",
                                     objective = 'reg:quantileerror', quantile_alpha=0.5, verbosity=0)
        clf = GridSearchCV(
            xgb_model,
            {"max_depth": [2, 4, 6, 8, 10, 15, 20], "n_estimators": [50, 100, 200], 'booster': ['gbtree', 'gblinear'],
             'eta': [0.10, 0.30, 0.50, 0.80, 1.00], 'subsample': [0.5, 0.75, 1.00]},
            verbose = 1,
            n_jobs = 2,
            cv = 5
        )
        clf.fit(X, y)

    return clf
```

5-fold cross validation to determine the optimal hyperparameters

# Implement cross-validation

```
# Cross-validation to determine the optimal hyperparameters
clf = XgboostParamTuning(xtrain, ytrain)
param = clf.best_params_
```

This code block uses the function defined in previous slide to determine the optimal hyperparameters

It should 2-3 minutes for this cell to run

**param =** contains the optimal hyperparameters

# Train the model with optimal hyperparameter and predict

```
# Train the model
xgb_model = xgb.XGBRegressor(n_jobs=multiprocessing.cpu_count(), tree_method="hist",
                             objective = 'reg:quantileerror', quantile_alpha=0.5,
                            booster = param['booster’], max_depth=param['max_depth’],
                            n_estimators=param['n_estimators'], eta=param['eta'], subsample=param['subsample'])
xgb_model.fit(xtrain, ytrain)
```

```
# predict for the test samples
ytest_est = xgb_model.predict(xtest)
```

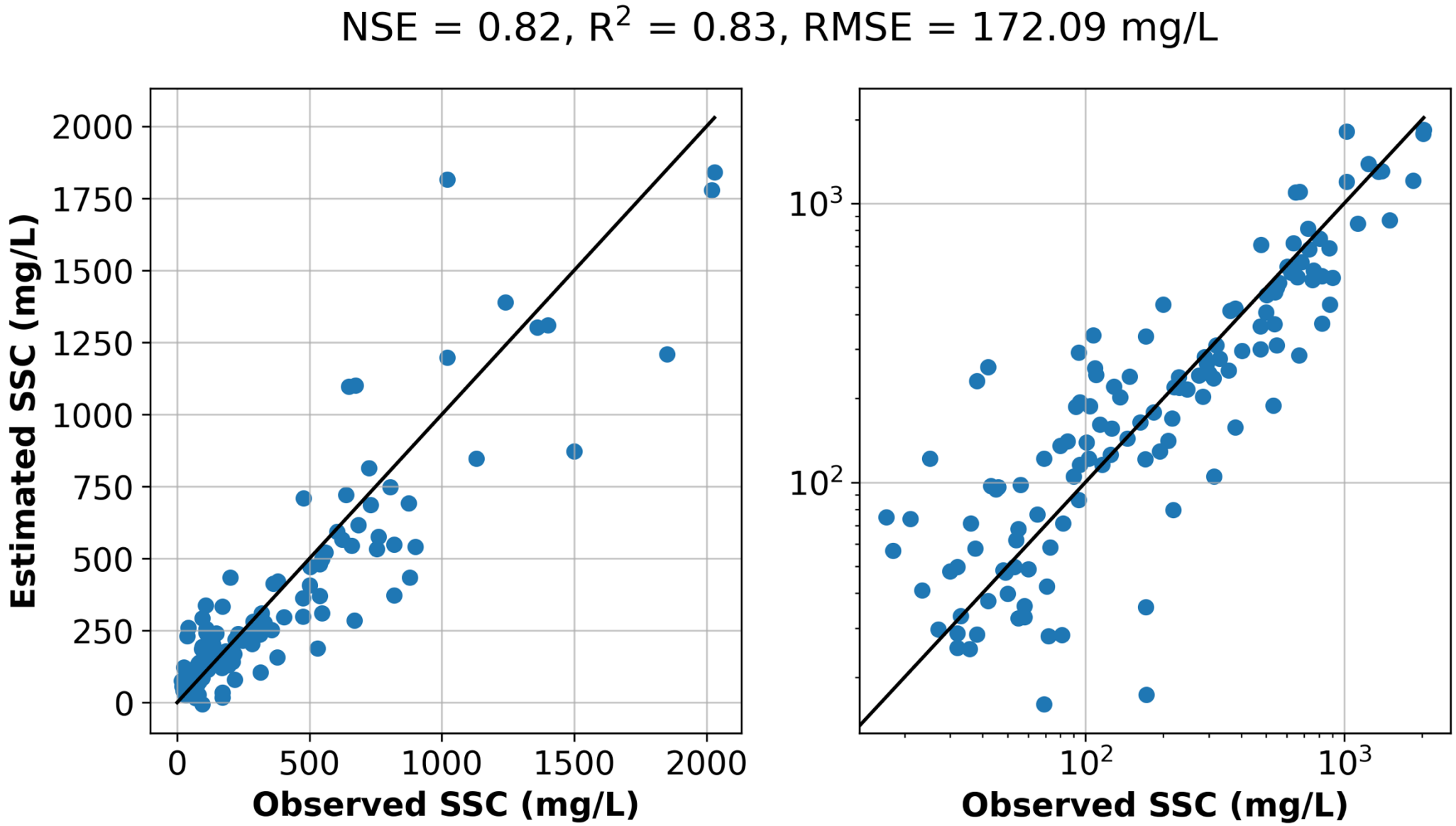
Develop the final model using the optimal hyperparameters

Note that **mean absolute error** is used as the objective function

# Compute the model performance and plot the results

```
# Assess the model performance
nse = computeNSE(ytest, ytest_est)
r2 = np.corrcoef(ytest, ytest_est)[0,1]**2
rmse = np.mean((ytest - ytest_est)**2)**0.5

# Plot
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.size'] = 14
plt.figure(figsize=(10, 5))

plt.subplot(1,2,1)
mx = np.max((ytest, ytest_est))
plt.scatter(ytest, ytest_est)
plt.plot([0, mx], [0, mx], color='black')
plt.grid(alpha=0.7)
plt.xlabel('Observed SSC (mg/L)', fontweight = 'bold')
plt.ylabel('Estimated SSC (mg/L)', fontweight = 'bold')

plt.subplot(1,2,2)
mx = np.max((ytest, ytest_est))
plt.scatter(ytest, ytest_est)
plt.plot([0, mx], [0, mx], color='black')
plt.grid(alpha=0.7)
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Observed SSC (mg/L)', fontweight = 'bold')

plt.suptitle('NSE = {}, R$^2$ = {}, RMSE = {} mg/L'.format(round(nse, 2), round(r2, 2), round(rmse, 2)))
```

# Summary plots



# Summary

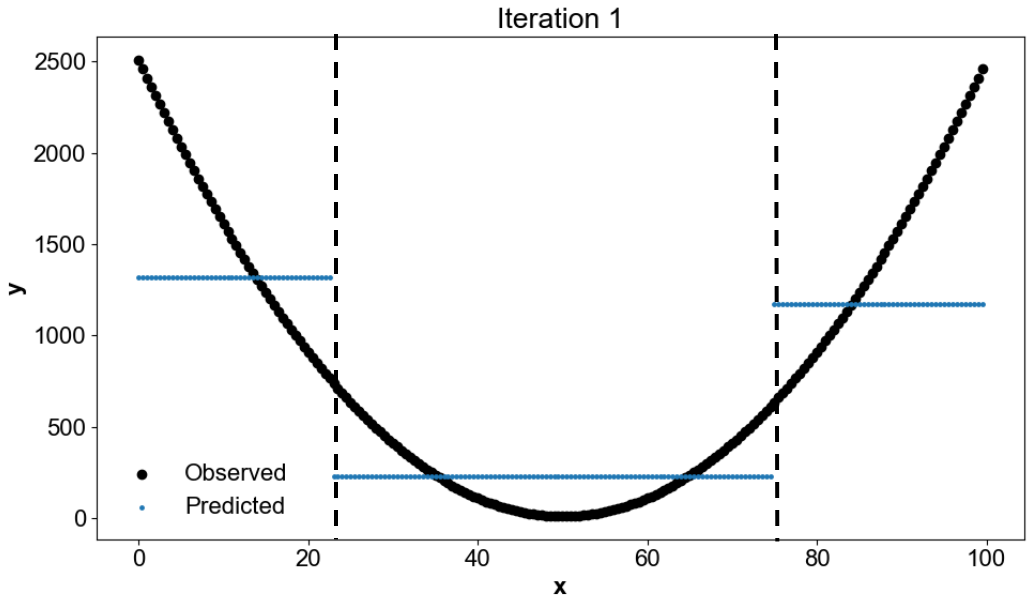
- This model was developed for a particular river (COMID)
- Different rivers have different model performance\
- Try a different COMID

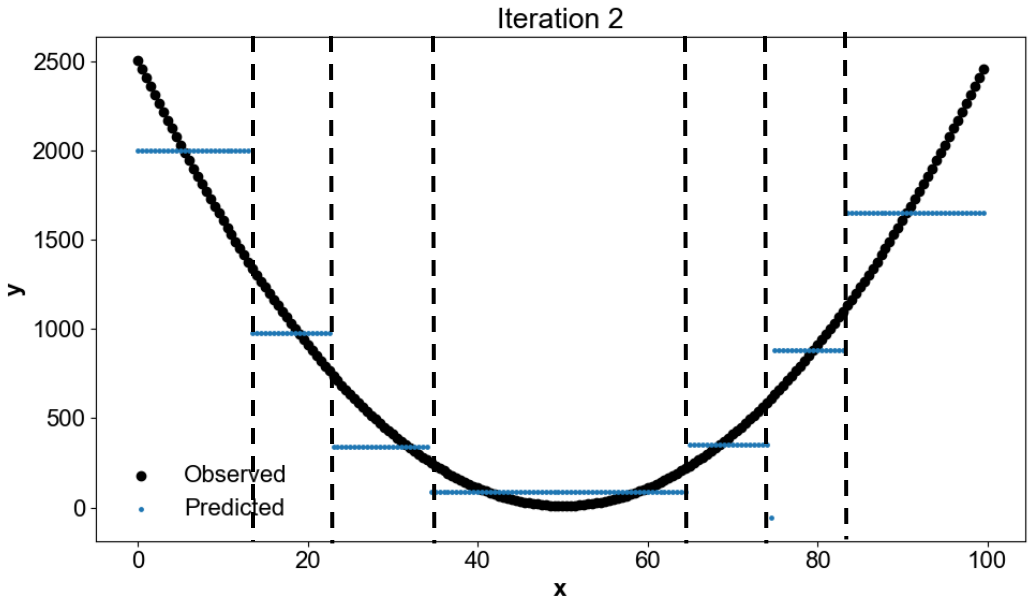
**Thank you.**

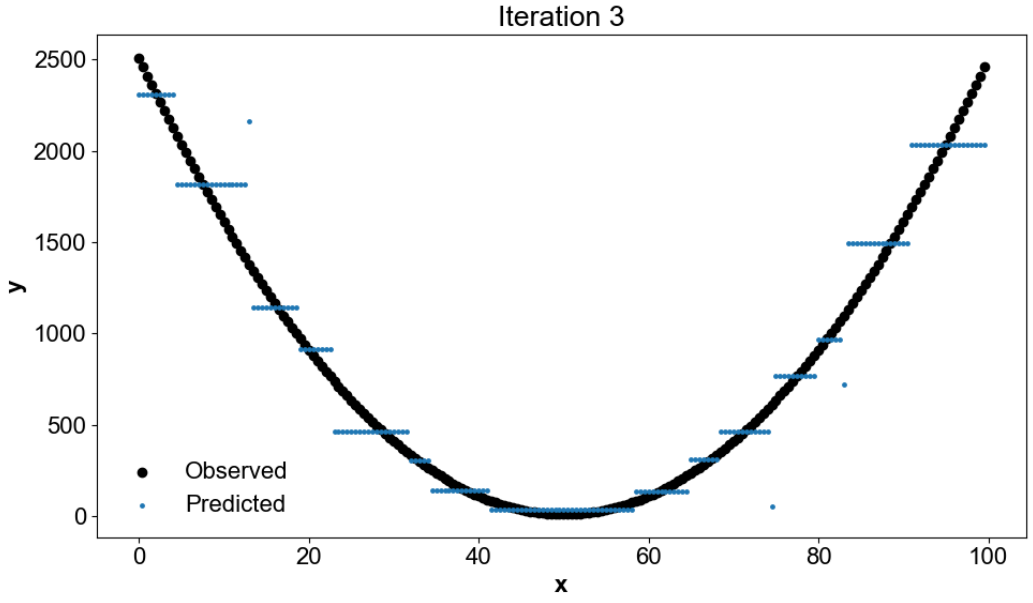
**gupta4ab@ucmail.uc.edu**

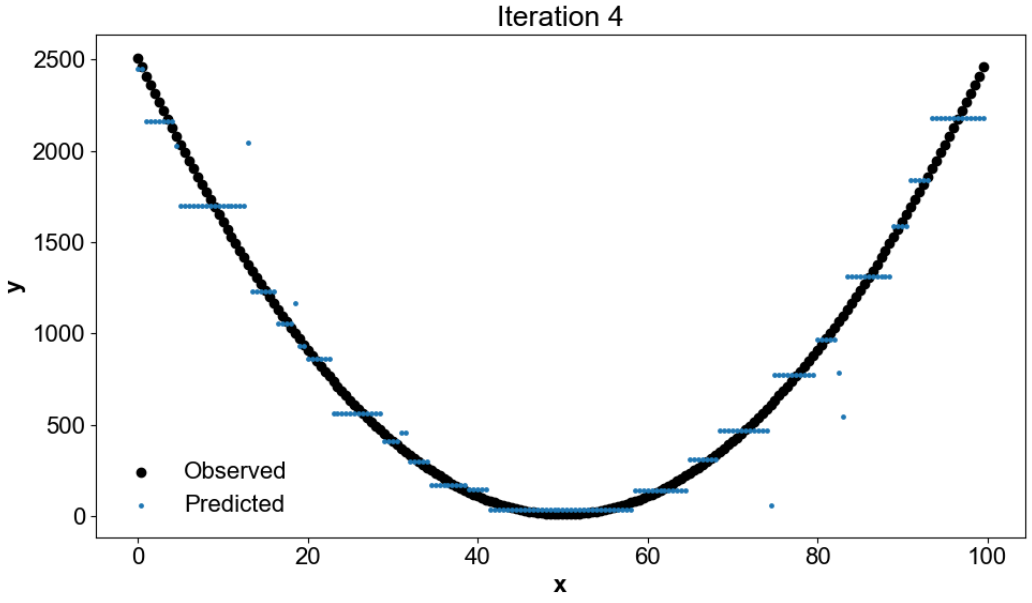
https://github.com/AbhinavGupta1611/

# Xgboost: An illustration



# Hyperparameters

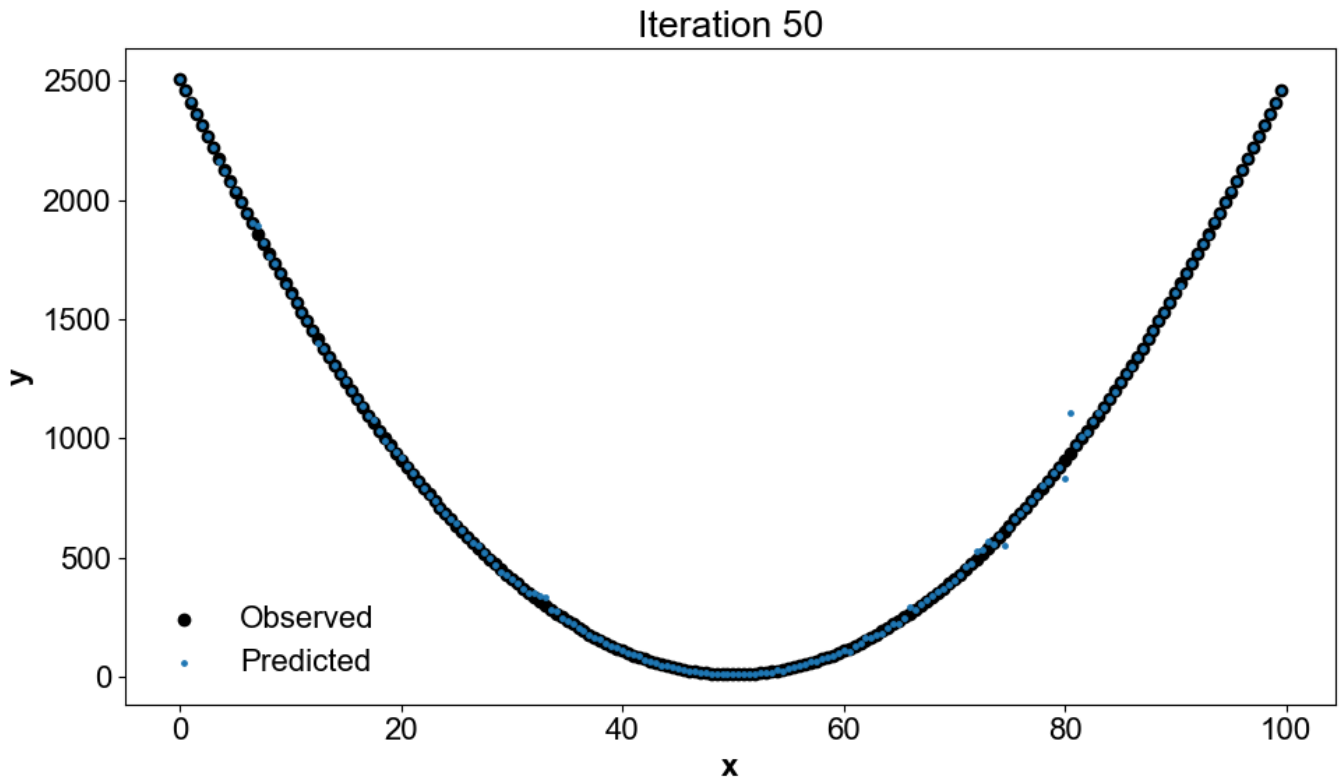
**Boosting method** (gbtree)

**Maximum tree depth**

**Number of estimators**

**Shrinkage rate**

**Training subsample**




$$F(x) = f_1(x) + f_2(x) + \cdots + f_N(x)$$