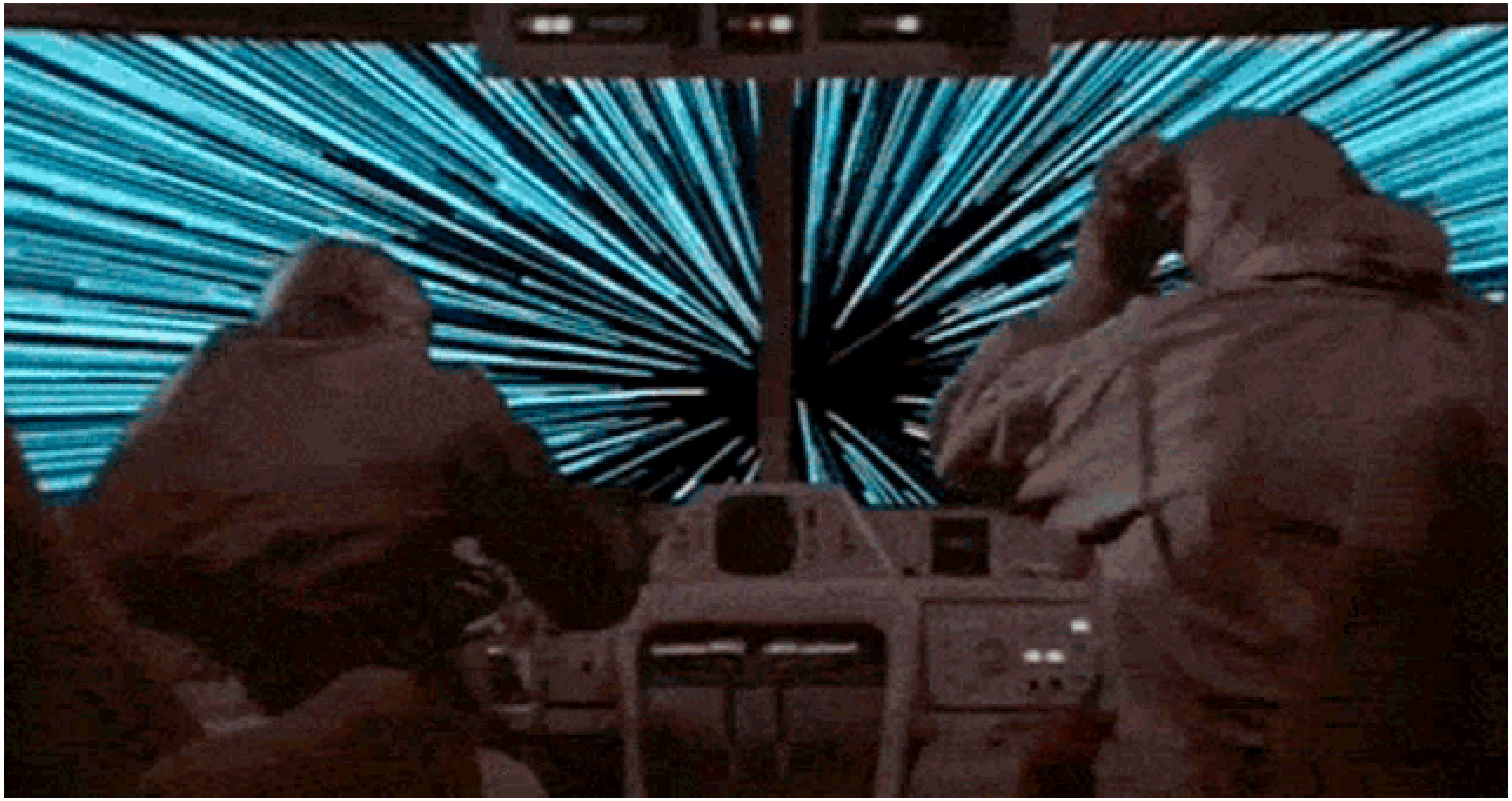
Boosted Trees and `xgboost`



Ensemble Flavors - Extended

Single Model Paradigm

Standard Model:

$$\hat{f}(x) = f^*(x) \approx f(x)$$

All training data is used to generate our single best estimate of the true functional form, $f(x)$.

Bagging (Refresh)

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b^*(x)$$

In bagging, each estimate utilizes a bootstrap (random) sample of the training data

The bagged estimate is then based on the weighted average of all of the models

Boosting

If we boost an algorithm using M stages, then we need to define $f_m(x)$ at each stage

$$\hat{f}_0(x) = 0$$

At each subsequent stage, we solve for

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) - f^*_m(x)$$

So that each stage adds more information to our model.

Q: Why do we subtract??

Boosting vs Bagging

Bagging:

- An averaged model utilizing bootstrapped samples of the complete dataset

Boosting:

- An "additive" model, where the predictions are incrementally improved

Boosting vs Bagging

Bagging:

- Much easier to implement
- Less Overfitting

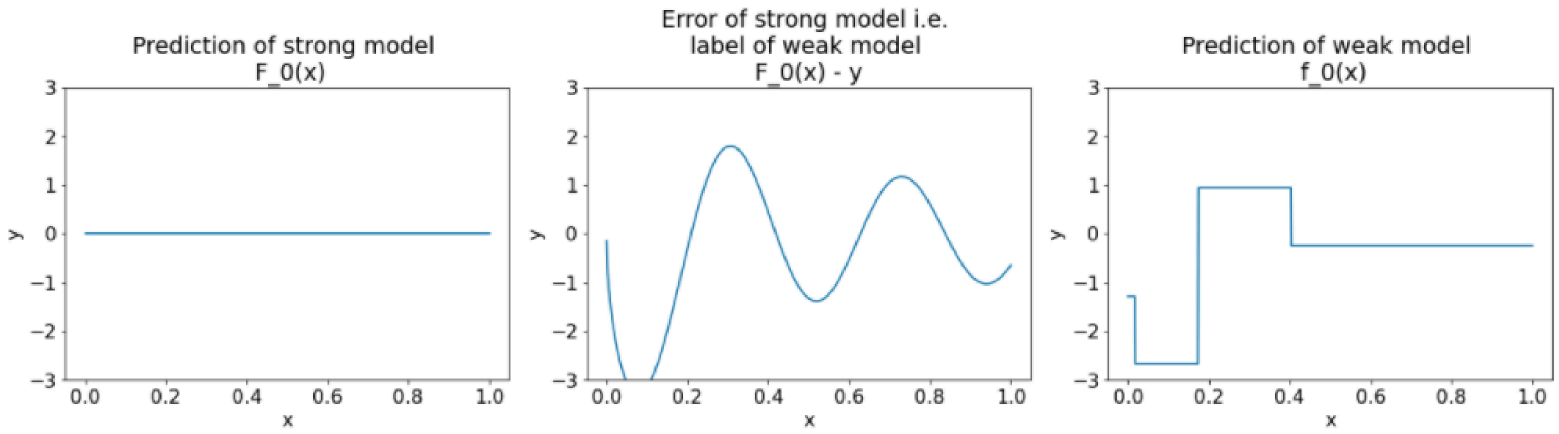
Boosting:

- Better Performance (generally)
- More vulnerable to overfitting

A visual example

Start with

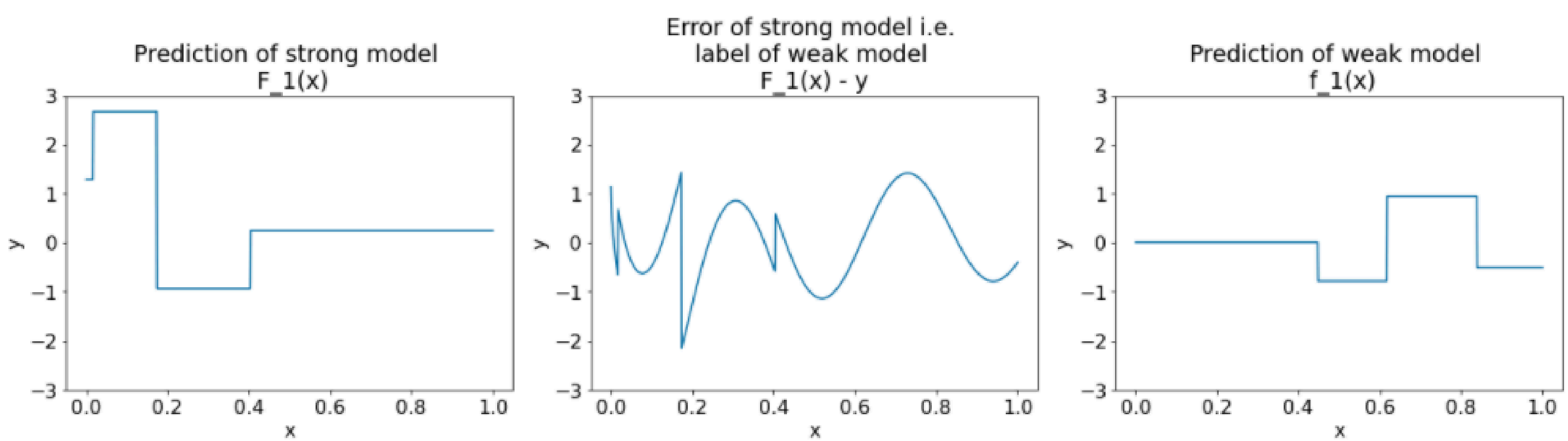
- A baseline model
- Error
- Tree fitted to the error of the baseline



A visual example

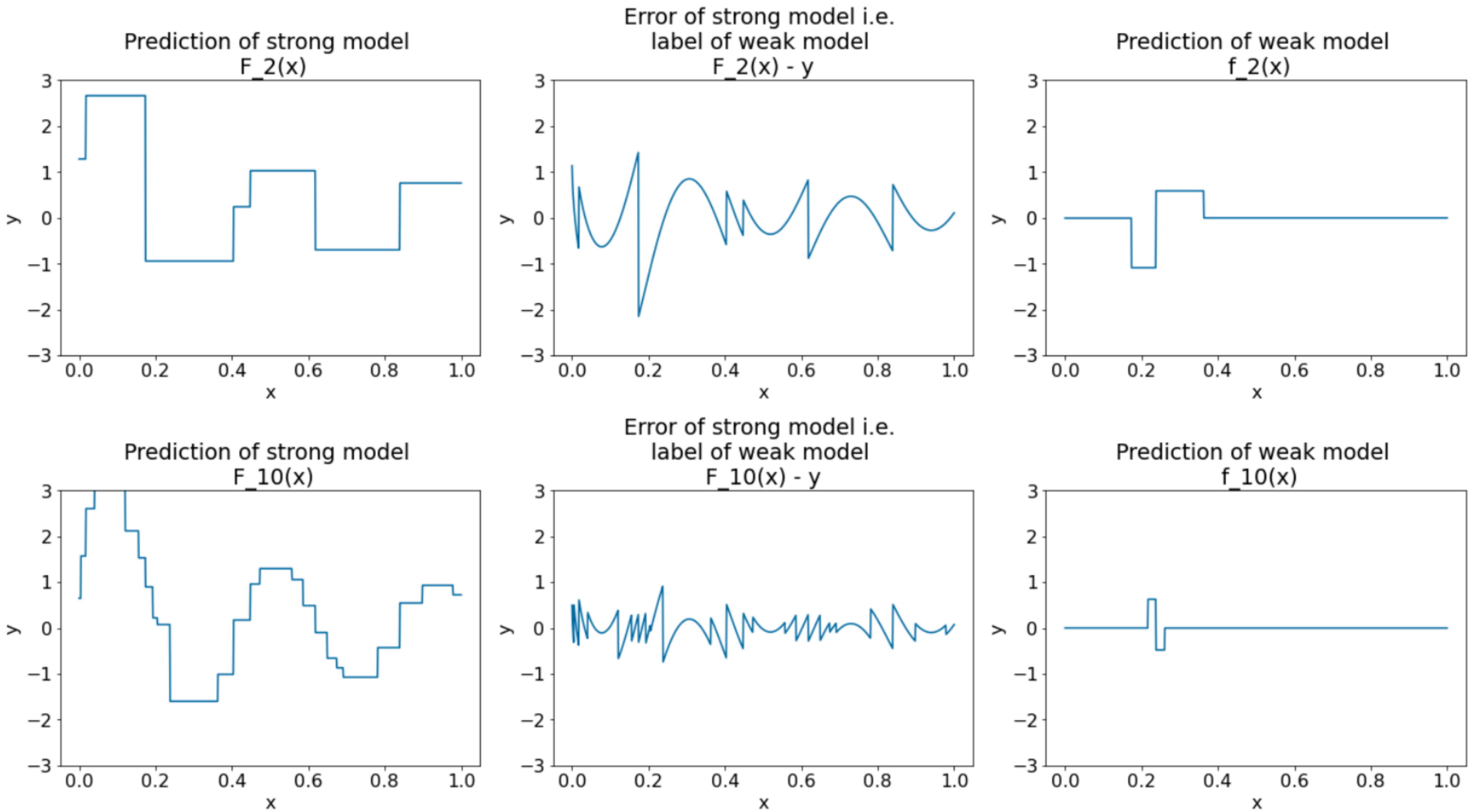
Our new model is the old baseline - the new tree, then we repeat:

- Baseline
- Error
- Tree predicting error



A visual example

We keep doing it! Over time, our model converges toward the patterns observed in the data:



Coding `xgboost` models

Why `xgboost`? It enables parallel computation of the model and distributed training!

This makes it a useful production model. 😃

Coding `xgboost` models

First things first, let's import all of our other libraries and read in our MNIST data:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = pd.read_csv("https://github.com/dustywhite7/
Econ8310/raw/master/DataSets/mnistTrain.csv")
```

Coding `xgboost` models

Create those train and test splits!

```
# Upper case before split, lower case after
Y = data['Label']
# make sure you drop a column with the axis=1 argument
X = data.drop('Label', axis=1)

x, xt, y, yt = train_test_split(X, Y, test_size=0.1,
     random_state=42)
```

Coding `xgboost` models

Implement the model:

```
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators=50, max_depth=3,
 learning_rate=0.5, objective='multi:softmax')
```

Coding `xgboost` models

Fit the model and test its performance:

```
xgb.fit(x, y)

pred = xgb.predict(xt)

print(accuracy_score(yt, pred)*100)
```

```
93.4
```

Nice! Low effort, high accuracy!

A Note on Cross-Validation

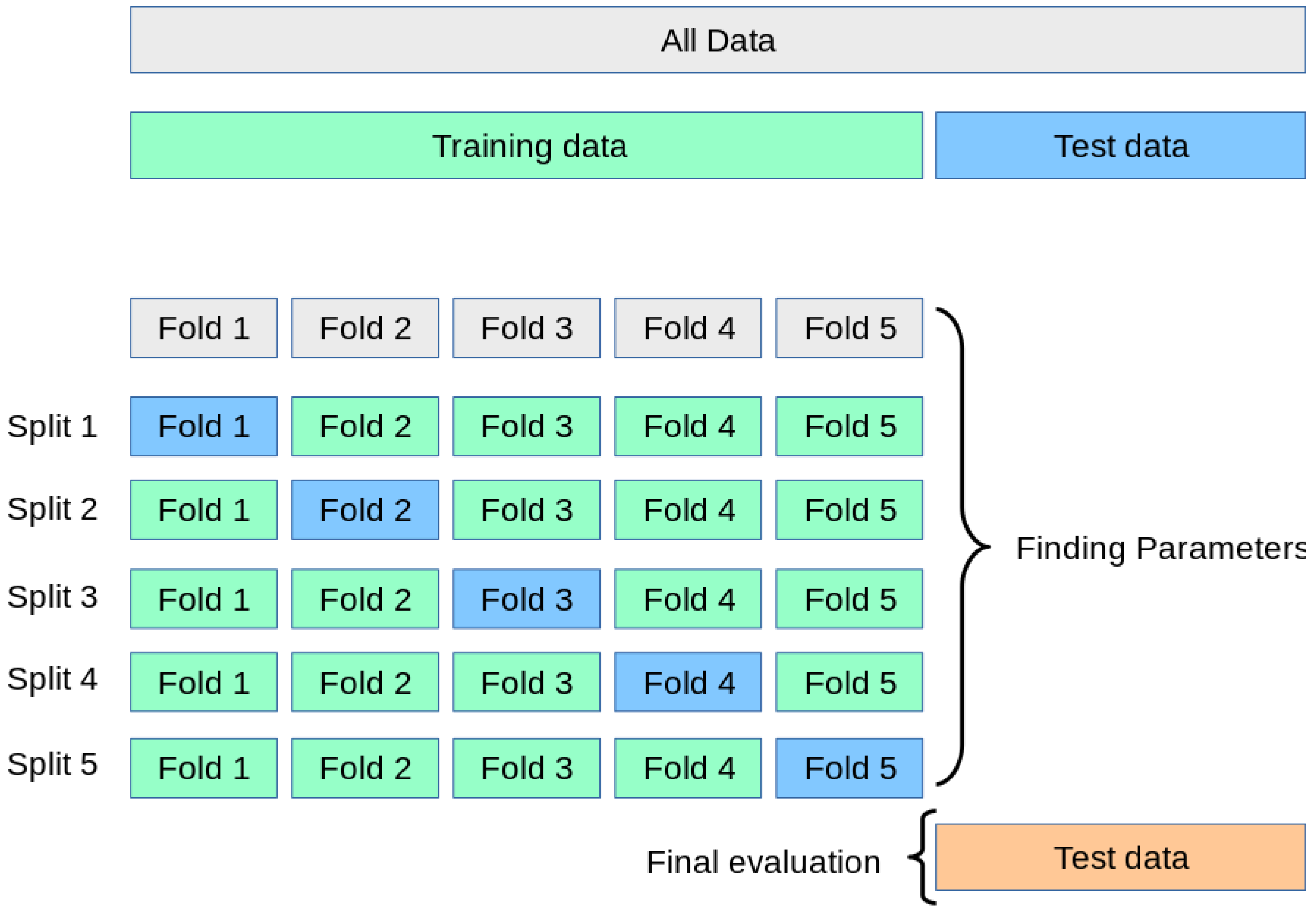
Cross-Validation

In order to maximize the information that we have about our model's performance, we will often test the model on many draws of our existing data.

This is called **cross-validation**. We resample our training and testing data k times, and compare the performance of the models.

If the models perform more or less equally well, then we can treat our model as well-specified. If not, then we know performance was sample-dependent.

Cross-Validation Diagram



credit to `sklearn` for the image

Cross-Validation Code

```
from sklearn.model_selection import KFold

# If we have imported data and created x, y already:
kf = KFold(n_splits=10) # 10 "Folds"

models = [] # We will store our models here

for train, test in kf.split(x): # Iterate over folds
  model = model.fit(x[train], y[train]) # Fit model
  accuracy = accuracy_score(y[test],    # Store accuracy
    model.predict(x[test]))
  print("Accuracy: ", accuracy_score(y[test],
    model.predict(x[test])))            # Print results
  models.append([model, accuracy])      # Store it all

print("Mean Model Accuracy: ",          # Print aggregate
  np.mean([model[1] for model in models]))
```

One More Note

After we complete our cross-validation, we do not use the cross-validation models. When we are satisfied with the results of the cross-validation, we

- Recombine all training data
- Train the model on all training data
- Test the model on withheld testing data (should not have been used at all in cross-validation)
- If results are still positive, implement model!

Lab Time!