Assignment 2: Exploring Tree-Based Regression Methods for 2D Linear Data

DTSC 680: Applied Machine Learning

Name: Juliana Meirelles

Directions and Overview

The main purpose of this assignment is for you to gain experience using tree-based methods to solve simple regression problems. In this assignment, you will fit a `Gradient-Boosted Regression Tree`, a `Random Forest`, and a `Decision Tree` to a noisy 2D linear data set. Since these models can be trained very quickly on the supplied data, I want you to first manually adjust hyperparameter values and observe their influence on the model's predictions. That is, you should manually sweep the hyperparameter space and try to hone in on the optimal hyperparameter values, again, *manually*. (Yep, that means guess-and-check: pick some values, train the model, observe the prediction curve, repeat.)

But wait, there's more! Merely attempting to identify the optimal hyperparameter values is not enough. Be sure to really get a visceral understanding of how altering a hyperparameter in turn alters the model predictions (i.e. the prediction curve). This is how you will build your machine learning intuition!

So, play around and build some models. When you are done playing with hyperparameter values, you should try to set these values to the optimal values manually (you're likely going to be *way* off). Then, retrain the model. Next in this assignment, we will perform several grid searches, so you'll be able to compare your "optimal" hyperparameter values with those computed from the grid search.

We will visualize model predictions for the optimal `Gradient-Boosted Regression Tree`, a `Random Forest`, and `Decision Tree` models that were determined by the grid searches. Next, you will compute the generalization error on the test set for the three models.

You will think critically about the different algorithms, as well as their prediction results, and characterize the trends you observe in the prediction results. Can you explain in your own words how the algorithms work? How do the results predicted by these models compare and contrast? Can you see any relationship between the algorithms and the model results?

Lastly, based on the arguments outlined in your critical analysis you must identify the model that best characterizes this data. In other words, which of these models is going to generalize better? Consider all factors.

Preliminaries

Let's import some common packages:

In [96]:
```
# Common imports
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
import numpy as np
import pandas as pd
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os

# Where to save the figures
PROJECT_ROOT_DIR = "."
FOLDER = "figures"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, FOLDER)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

def plot_predictions(regressor, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = regressor.predict(x1.reshape(-1, 1))
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

Import and Split Data

Complete the following:

1. Begin by importing the data from the file called `2DLinear.csv`. Name the returned DataFrame `data`.
2. Call train_test_split() with a `test_size` of 20%. Save the output into `X_train`, `X_test`, `y_train`, and `y_test`, respectively. Specify the `random_state` parameter to be `42` (do this throughout the entire note book).
3. Reshape some data, so that we don't run into trouble later.

In [177…

```
from sklearn.model_selection import train_test_split

data = pd.read_csv("2DLinear.csv")

X = data['x']
y = data['y']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

# Reshape X Data and Make NumPy Arrays
X_train = np.array(X_train).reshape(-1,1)
X_test = np.array(X_test).reshape(-1,1)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

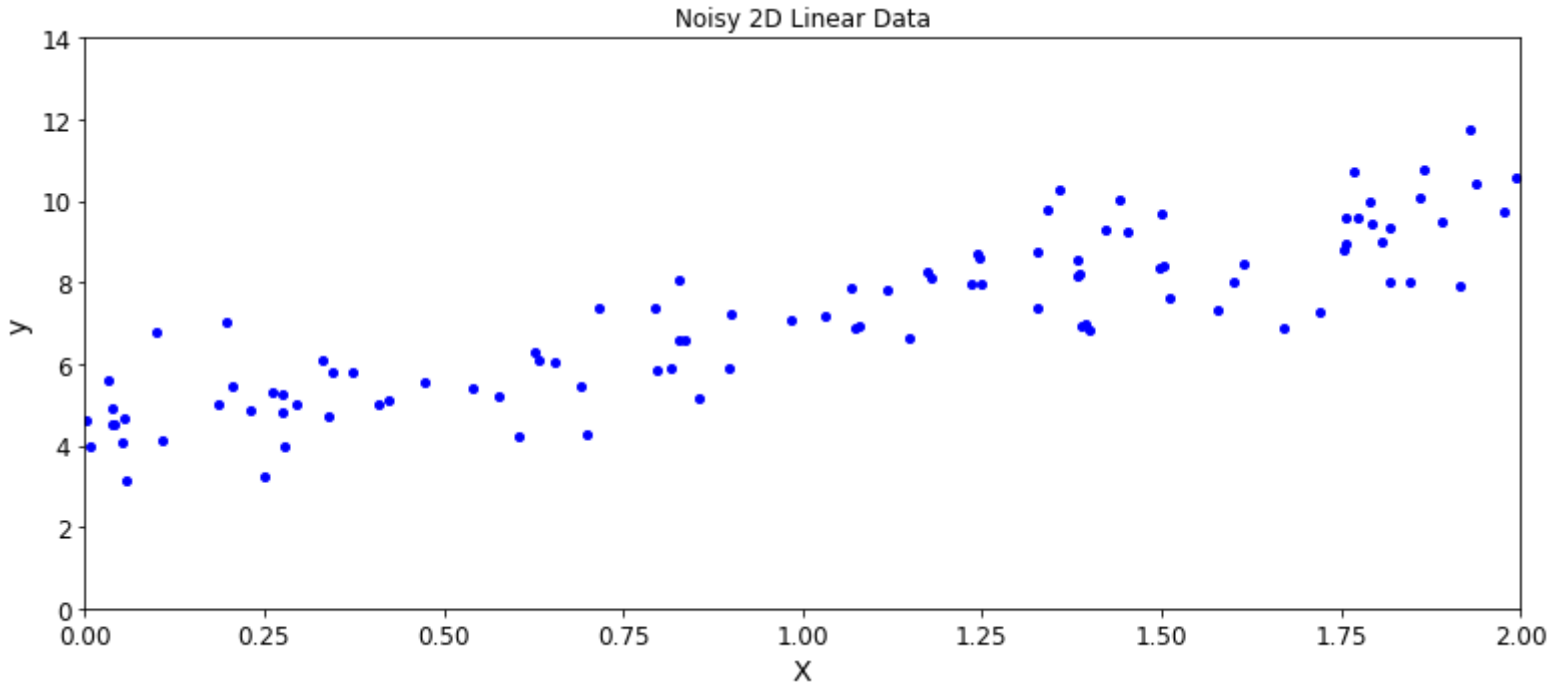
Plot Data

Simply plot your training data here in order to create the plot below.

In [178…

```
plt.figure(figsize=(11,5))
plt.plot(X_train, y_train, 'bo', markersize= 4)
plt.xlabel("X")
plt.ylabel("y")
plt.title("Noisy 2D Linear Data")
plt.axis([0, 2, 0, 14])
save_fig("Raw2DLinearData")
plt.show()
```

```
Saving figure Raw2DLinearData
```



Explore 2D Linear Data: GradientBoostingRegressor

Fit a `GradientBoostingRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

- `learning_rate = <value>`
- `max_depth = <value>`
- `n_estimators = <value>`
- `random_state = 42`

In [179…

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
gbrt = GradientBoostingRegressor(max_depth = 2, n_estimators = 40, learning_rate = 0.1, random_state = 42)
gbrt.fit(X_train,y_train)
```

Out[179… GradientBoostingRegressor(max_depth=2, n_estimators=40, random_state=42)

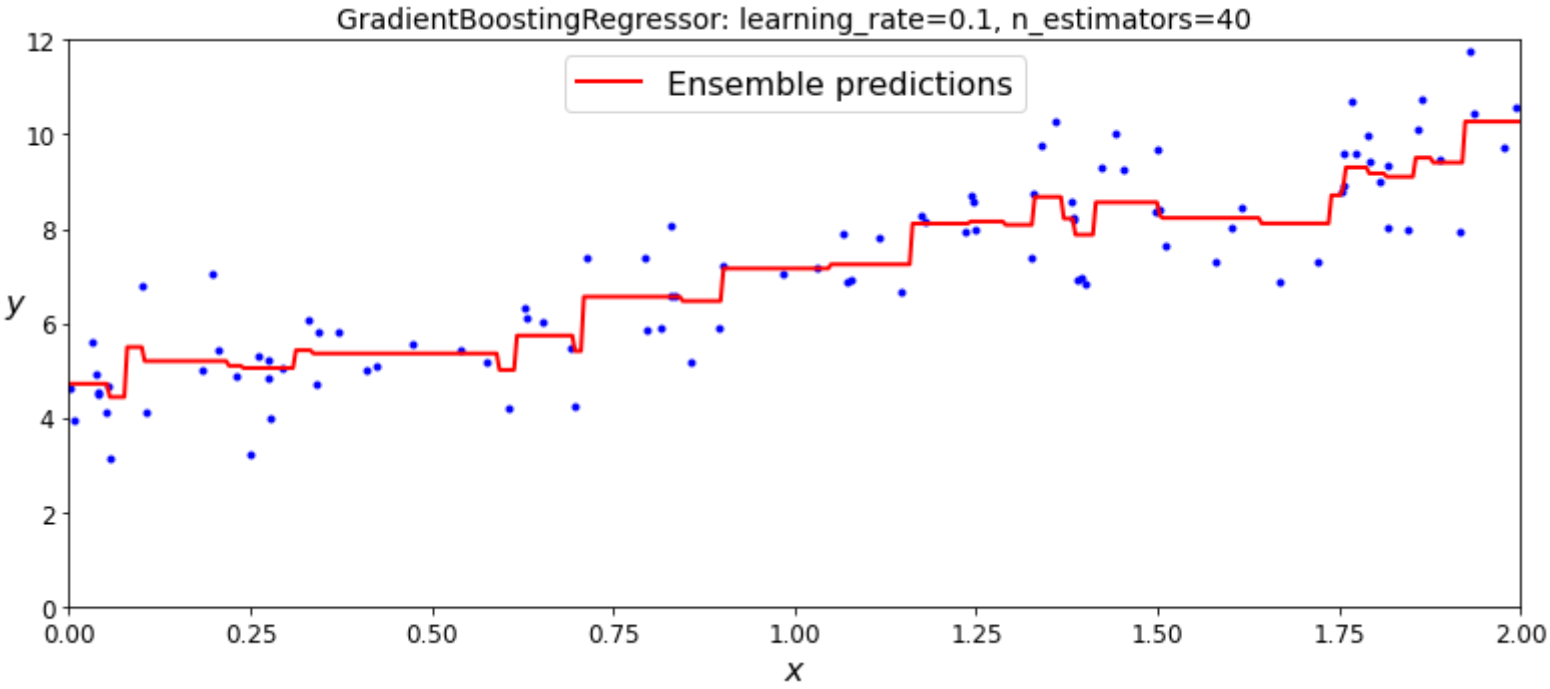
Plot Model Predictions for Training Set

In [180…
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)


plt.figure(figsize=(11,5))
plot_predictions([gbrt], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title("GradientBoostingRegressor: learning_rate={}, n_estimators={}".format(gbrt.learning_rate, gbrt.n_esti
          fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("gbrt_2DLinearReg")
plt.show()
```

```
Saving figure gbrt_2DLinearReg
```



Explore 2D Linear Data: RandomForestRegressor

Fit a `RandomForestRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

- `min_samples_split = <value>`
- `max_depth = <value>`
- `n_estimators = <value>`
- `random_state = 42`

In [181…

```
from sklearn.ensemble import RandomForestRegressor
```

```
rft_reg = RandomForestRegressor(min_samples_split = 4, max_depth = 2, n_estimators = 4, random_state = 42)
rft_reg.fit(X_train,y_train)
```

Out[181…
```
RandomForestRegressor(max_depth=2, min_samples_split=4, n_estimators=4,
                      random_state=42)
```

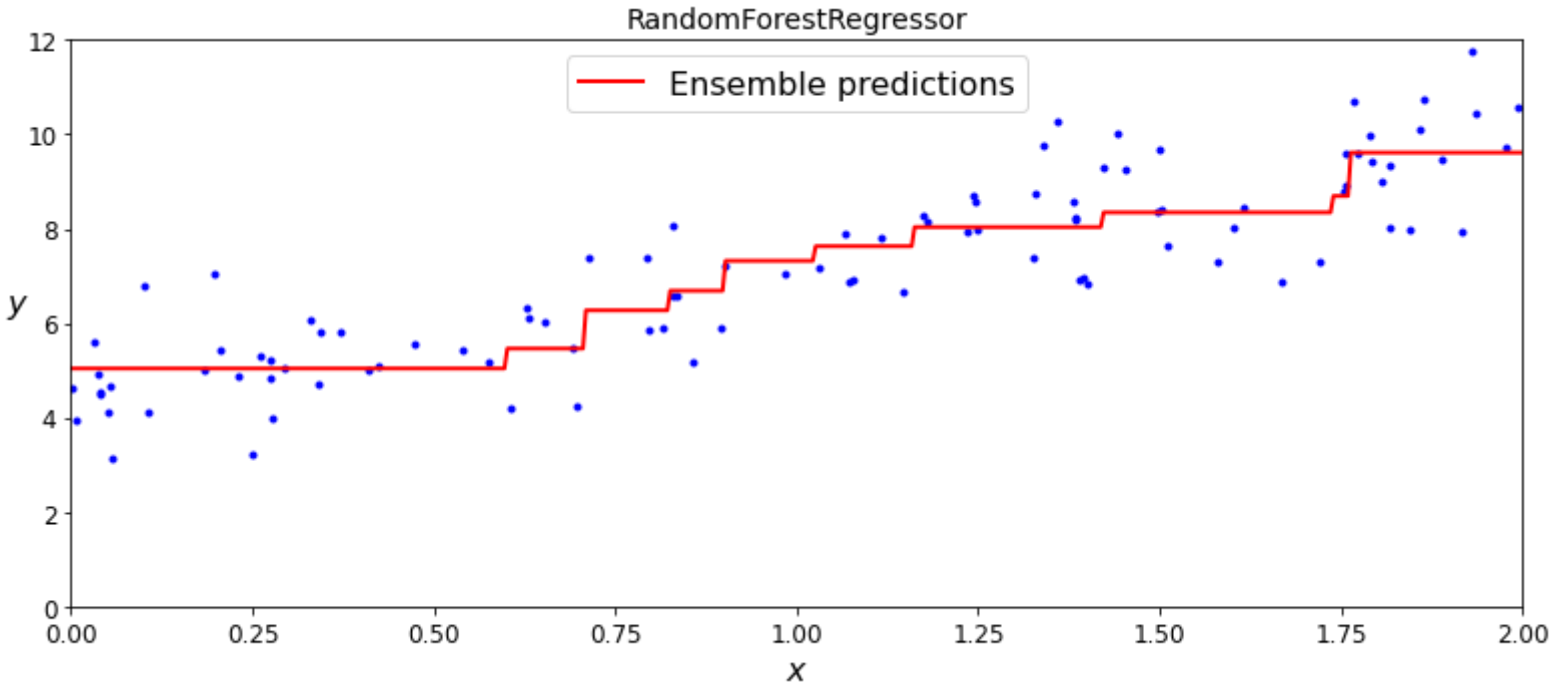
Plot Model Predictions for Training Set

In [182…
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)

plt.figure(figsize=(11,5))
plot_predictions([rft_reg], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title(("RandomForestRegressor"),fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("rf_2DLinearReg")
plt.show()
```

```
Saving figure rf_2DLinearReg
```



Explore 2D Linear Data: DecisionTreeRegressor

Fit a `DecisionTreeRegressor` model to this data. You must manually assign values to the following hyperparameters. You should "play around" by using different combinations of hyperparameter values to really get a feel for how they affect the model's predictions. When you are done playing, set these to the best values you can for submission. (It is totally fine if you don't elucidate the optimal values here; however, you will want to make sure your model is not excessively overfitting or underfitting the data. Do this by examining the prediction curve generated by your model. You will be graded, more exactly, on the values that you calculate later from performing several rounds of grid searches.)

- `splitter = <value>`
- `max_depth = <value>`
- `min_samples_split = <value>`
- `random_state = 42`

In [183…

```
from sklearn.tree import DecisionTreeRegressor
```

```
dtree_reg = DecisionTreeRegressor(splitter = "best", max_depth = 2, min_samples_split = 4, random_state = 42)
dtree_reg.fit(X_train,y_train)
```

Out[183… DecisionTreeRegressor(max_depth=2, min_samples_split=4, random_state=42)

Plot Model Predictions for Training Set

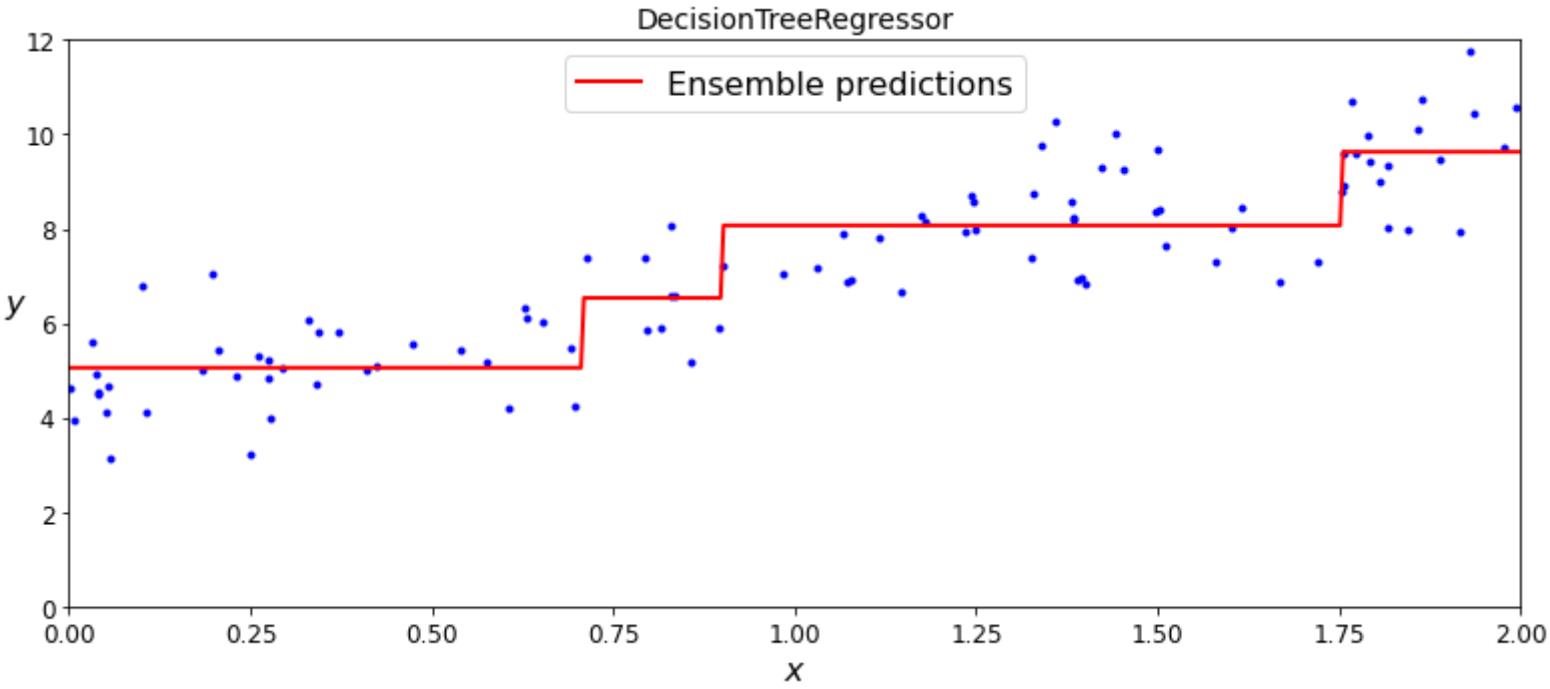
In [184…

```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)

plt.figure(figsize=(11,5))
plot_predictions([dtree_reg], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title(("DecisionTreeRegressor"), fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("dt_2DLinearReg")
plt.show()
```

```
Saving figure dt_2DLinearReg
```



Perform Grid Searches

You will perform a series of grid searches, which will yield the optimal hyperparamter values for each of the three model types. You can compare the values computed by the grid search with the values you manually found earlier. How do these compare?

You must perform a course-grained grid search, with a very broad range of values first. Then, you perform a second grid search using a tighter range of values centered on those identified in the first grid search. You may have to use another round of grid searching too (it took me at least three rounds of grid searches per model to ascertain the optimal hyperparameter values below).

Note the following:

1. Be sure to clearly report the optimal hyperparameters in the designated location after you calculate them!
2. You must use `random_state=42` everywhere that it is needed in this notebook.
3. You must use grid search to compute the following hyperparameters. Use the following hyperparameter values as the bounds of the ranges to be probed in your initial (course-grained) stint of grid searching. Note that these are NOT upper bounds for the

further refined grid searches. If your initial search returns the upper bound of your values, you can (and should) search further above and below that value. You should examine these hyperparameters and their associated ranges, learn from this example, and use these values as a basis for your own future work when performing grid searches. This is how you will build your machine learning intuition. Keep in mind, these values also depend somewhat on this specific dataset! (For example, it is important to notice that I decided to probe up to 1,000 for n_estimators. 10 would have been far too small. 1,000,000 would have been far too large.)

GradientBoostingRegressor:

- `max_depth = up to 32`
- `n_estimators = up to 1000`
- `learning_rate = from .01 to 1`

RandomForestRegressor:

- `max_depth = up to 32`
- `n_estimators = up to 1000`
- `min_samples_split = up to 20`

DecisionTreeRegressor:

- `splitter = ["best", "random"]`
- `max_depth = up to 32`
- `min_samples_split = up to 20`

1. `learning rate` should be rounded to two decimals.
2. The number of cross-folds. Specify `cv=3`

Perform Individual Model Grid Searches

In this section you will perform a series of grid searches to compute the optimal hyperparameter values for each of the three model types.

In [163…
```
from sklearn.model_selection import GridSearchCV
```

In [187…
```
# -----
# Coarse-Grained GradientBoostingRegressor GridSearch
# -----
```

```
param_grid = {'learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0],
              'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'n_estimators': [100,200,300,400,500,600,700,800,900,1000]
             }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 550 candidates, totalling 1650 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 100}
[Parallel(n_jobs=1)]: Done 1650 out of 1650 | elapsed:  4.8min finished
```

In [188…

```
# -----
# Refined GradientBoostingRegressor GridSearch
# -----

param_grid = {'learning_rate': [0.1, 0.2, 0.3],
              'max_depth': [1, 2, 3],
              'n_estimators': [50, 100, 150, 200]
             }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 200}
[Parallel(n_jobs=1)]: Done 108 out of 108 | elapsed:    2.6s finished
```

In [189…

```
# -----
# Refined GradientBoostingRegressor GridSearch (round 2)
# -----

param_grid = {'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5],
              'max_depth': [1, 2, 3],
```

```
              'n_estimators': [100, 200, 300, 400]
             }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 60 candidates, totalling 180 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 400}
[Parallel(n_jobs=1)]: Done 180 out of 180 | elapsed:    7.7s finished
```

In [190…

```
# -----
# Final GradientBoostingRegressor GridSearch
# -----

param_grid = {'learning_rate': [.26, 0.28, 0.3, 0.32, 0.34, 0.36],
              'max_depth': [1],
              'n_estimators': [300, 350, 400, 450, 500]
             }

grid_search_cv = GridSearchCV(GradientBoostingRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'learning_rate': 0.3, 'max_depth': 1, 'n_estimators': 400}
[Parallel(n_jobs=1)]: Done  90 out of  90 | elapsed:    6.1s finished
```

On this dataset, the optimal model parameters for the `GradientBoostingRegressor` class are:

- `learning_rate = 0.3`
- `max_depth = 1`
- `n_estimators = 400`

In [196…

```
# -----
# Coarse-Grained RandomForestRegressor GridSearch
```

```
# -----

param_grid = {'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'n_estimators': [100,200,300,400,500,600,700,800,900,1000],
              'min_samples_split': [2, 4, 8, 10, 12, 14, 18, 20]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 880 candidates, totalling 2640 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 2640 out of 2640 | elapsed: 32.6min finished
The best parameters are:  {'max_depth': 8, 'min_samples_split': 8, 'n_estimators': 900}
```

In [197…

```
# -----
# Refined RandomForestRegressor GridSearch
# -----

param_grid = {'max_depth': [4, 5, 6, 7, 8, 9, 10, 11, 12],
              'n_estimators': [850, 860, 870, 880, 890, 900, 910, 920, 930, 940, 950],
              'min_samples_split': [6, 7, 8, 9, 10]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 495 candidates, totalling 1485 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1485 out of 1485 | elapsed: 41.0min finished
The best parameters are:  {'max_depth': 5, 'min_samples_split': 8, 'n_estimators': 900}
```

In [198…

```
# -----
# Final RandomForestRegressor GridSearch
# -----

param_grid = {'max_depth': [4, 5, 6],
              'n_estimators': [896, 897, 898, 899, 900, 901, 902, 903, 904],
```

```
              'min_samples_split': [7, 8, 9]
             }

grid_search_cv = GridSearchCV(RandomForestRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print("The best parameters are: ", grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 243 out of 243 | elapsed:  5.7min finished
The best parameters are:  {'max_depth': 5, 'min_samples_split': 8, 'n_estimators': 901}
```

On this dataset, the optimal model parameters for the `RandomForestRegressor` class are:

- `max_depth = 5`
- `n_estimators = 8`
- `min_samples_split = 901`

In [121…

```
# -----
# Coarse-Grained DecisionTreeRegressor GridSearch
# -----

param_grid = {'splitter': ['best', 'random'],
              'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 20, 25, 32],
              'min_samples_split': [2, 4, 8, 10, 12, 14, 18, 20]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 176 candidates, totalling 528 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
The best parameters are:  {'max_depth': 6, 'min_samples_split': 4, 'splitter': 'random'}
[Parallel(n_jobs=1)]: Done 528 out of 528 | elapsed:    0.6s finished
```

In [200…

```
# -----
# Refined DecisionTreeRegressor GridSearch
# -----
```

```
param_grid = {'splitter': ['random'],
              'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],
              'min_samples_split': [2, 3, 4, 5, 6]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 45 candidates, totalling 135 fits
The best parameters are:  {'max_depth': 6, 'min_samples_split': 4, 'splitter': 'random'}
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 135 out of 135 | elapsed:    0.2s finished
```

In [201…

```
# -----
# Final DecisionTreeRegressor GridSearch
# -----

param_grid = {'splitter': ['random'],
              'max_depth': [5, 6, 7],
              'min_samples_split': [3, 4, 5]
              }

grid_search_cv = GridSearchCV(DecisionTreeRegressor(random_state = 42),
                              param_grid, verbose = 1, cv = 3)

grid_search_cv.fit(X_train, y_train)

print('The best parameters are: ', grid_search_cv.best_params_)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
The best parameters are:  {'max_depth': 6, 'min_samples_split': 4, 'splitter': 'random'}
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  27 out of  27 | elapsed:    0.1s finished
```

On this dataset, the optimal model parameters for the `DecisionTreeRegressor` class are:

- `splitter = 'random'`
- `max_depth = 6`
- `min_samples_split = 4`

Visualize Optimal Model Predictions

In the previous section you performed a series of grid searches designed to identify the optimal hyperparameter values for all three models. Now, use the `best_params_` attribute of the grid search objects from above to create the three optimal models below. For each model, visualize the models predictions on the training set - this is what we mean by the "prediction curve" of the model.

Create Optimal GradientBoostingRegressor Model

In [191…
```
optimal_gbrt = GradientBoostingRegressor(learning_rate = 0.3, max_depth = 1, n_estimators = 400, random_state =

optimal_gbrt.fit(X_train, y_train)
```

Out[191…
```
GradientBoostingRegressor(learning_rate=0.3, max_depth=1, n_estimators=400,
                          random_state=42)
```

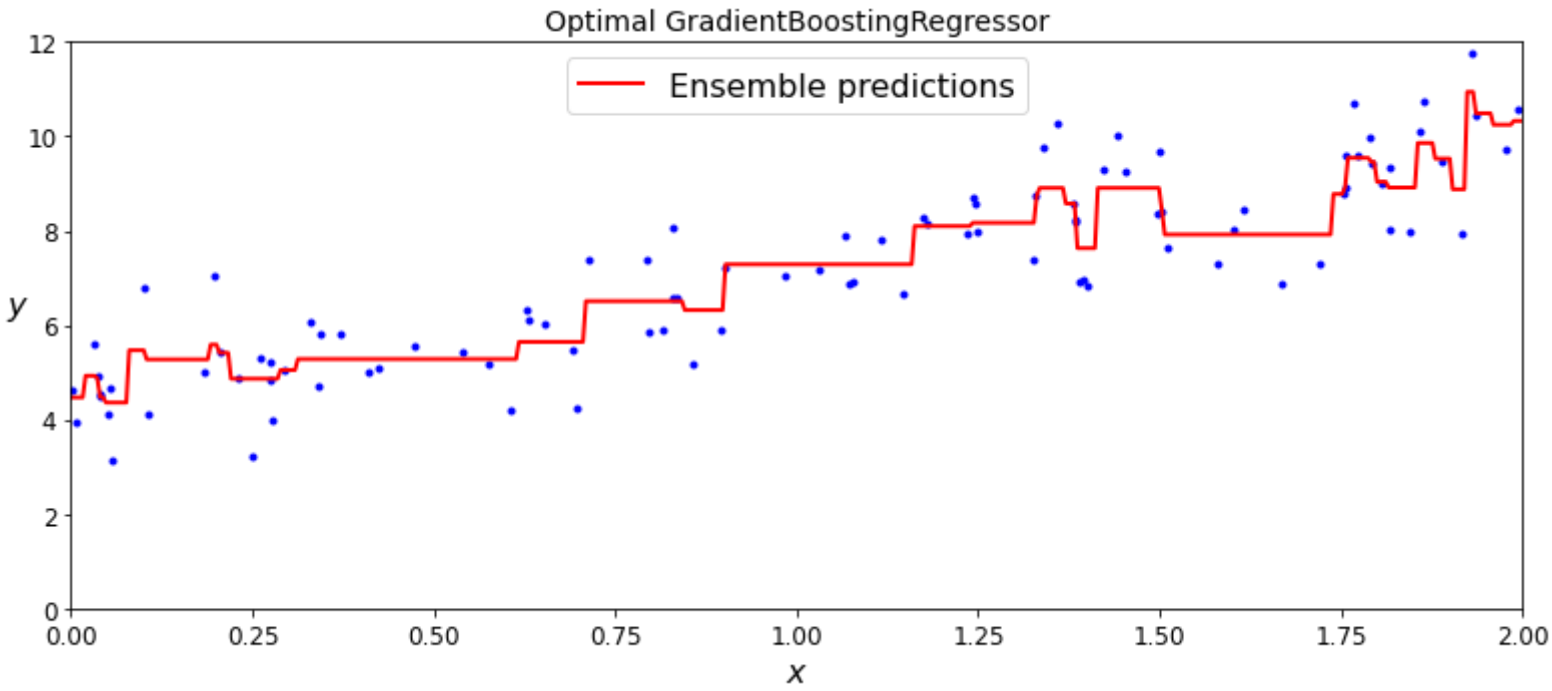
Plot Model Predictions for Training Set

In [192…
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)

plt.figure(figsize=(11,5))
plot_predictions([optimal_gbrt], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title(("Optimal GradientBoostingRegressor"), fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("gb_2DLinearReg")
plt.show()
```

```
Saving figure gb_2DLinearReg
```



Create Optimal RandomForestRegressor Model

In [202…
```
optimal_rft_reg = RandomForestRegressor(max_depth = 5, min_samples_split = 8, n_estimators = 901, random_state

optimal_rft_reg.fit(X_train, y_train)
```

Out[202…
```
RandomForestRegressor(max_depth=5, min_samples_split=8, n_estimators=901,
                      random_state=42)
```

Plot Model Predictions for Training Set

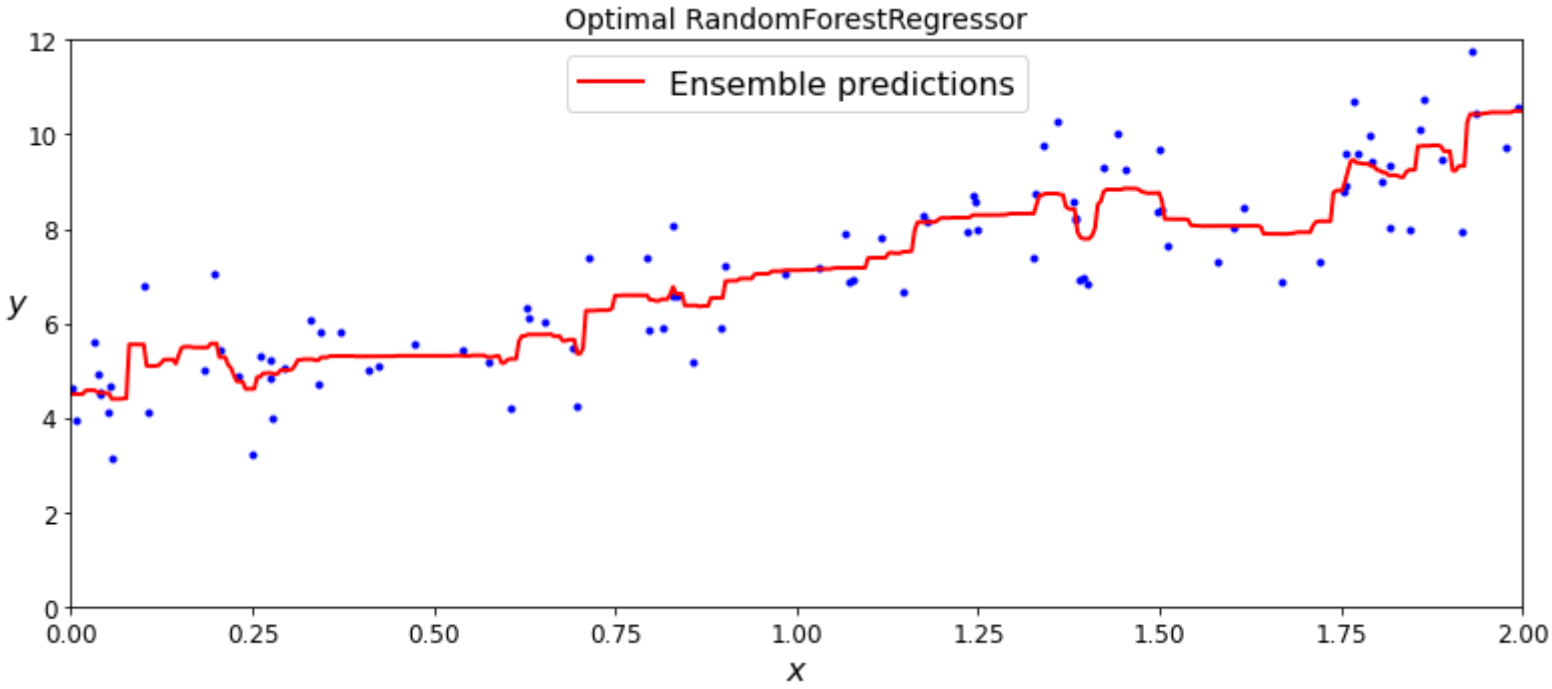
In [203…
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)

plt.figure(figsize=(11,5))
```

```
plot_predictions([optimal_rft_reg], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title(("Optimal RandomForestRegressor"), fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("rf_2DLinearReg")
plt.show()
```

```
Saving figure rf_2DLinearReg
```



Create Optimal DecisionTreeRegressor Model

In [204…
```
optimal_dtree_reg = DecisionTreeRegressor(max_depth = 6, min_samples_split = 4, splitter = 'random', random_sta

optimal_dtree_reg.fit(X_train, y_train)
```

Out[204…
```
DecisionTreeRegressor(max_depth=6, min_samples_split=4, random_state=42,
                      splitter='random')
```

Plot Model Predictions for Training Set

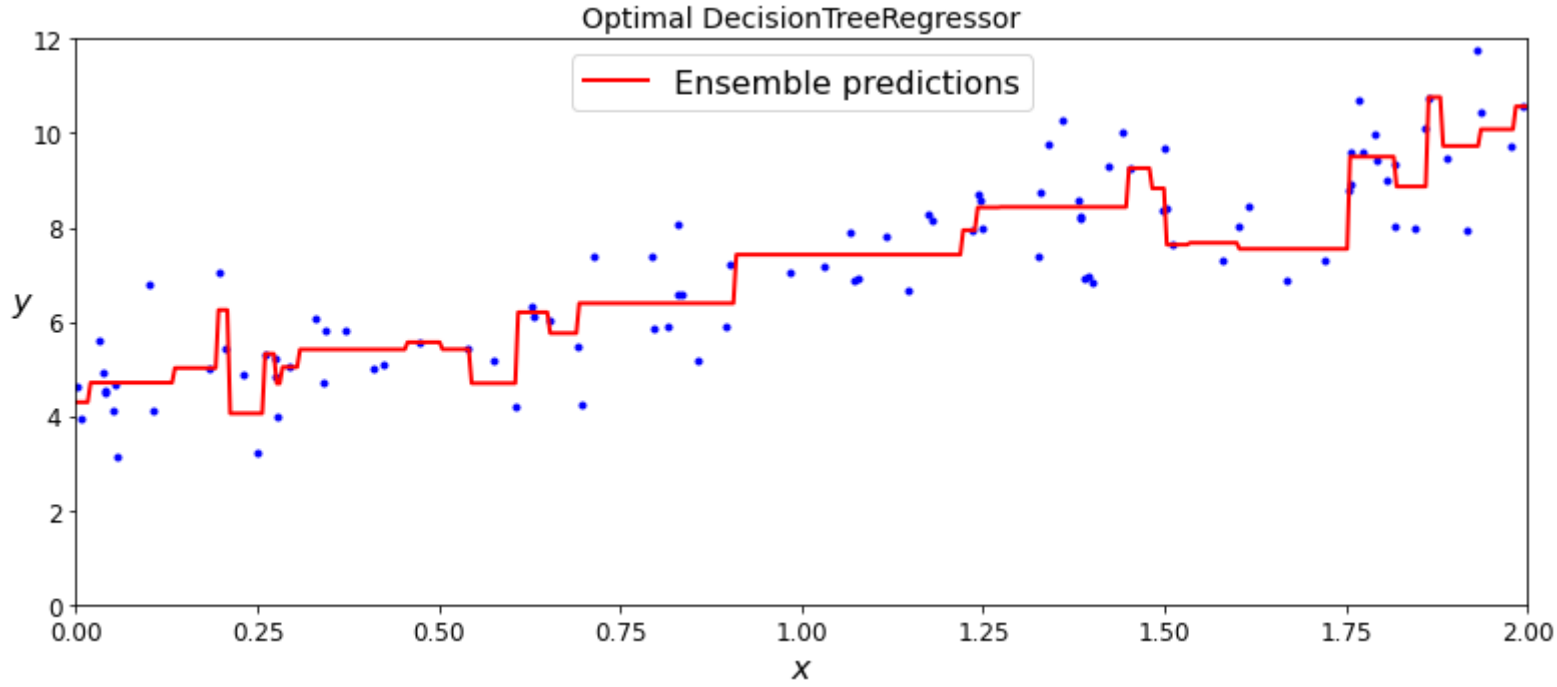
In [205…
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
```

```
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)

plt.figure(figsize=(11,5))
plot_predictions([optimal_dtree_reg], X_train, y_train, axes=[0, 2, 0, 12], label="Ensemble predictions")
plt.title(("Optimal DecisionTreeRegressor"), fontsize=14)
plt.xlabel("$x$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)


save_fig("rf_2DLinearReg")
plt.show()
```

```
Saving figure rf_2DLinearReg
```



Compute Generalization Error

Compute the generalization error for each of the optimal models computed above. Use MSE as the generalization error metric. Round your answers to four significant digits. Print the generalization error for all three models.

In [206…

```
from sklearn.metrics import mean_squared_error

y_pred1 = optimal_gbrt.predict(X_test)

y_pred2 = optimal_rft_reg.predict(X_test)

y_pred3 = optimal_dtree_reg.predict(X_test)

gbrt_mse = mean_squared_error(y_test, y_pred1)

rf_mse = mean_squared_error(y_test, y_pred2)

dtree_mse = mean_squared_error(y_test, y_pred3)

print('GradientBoostingRegressor Model MSE', round(gbrt_mse, 4))
print('RandomForestRegressor Model MSE', round(rf_mse, 4))
print('DecisionTreeRegressor Model MSE', round(dtree_mse, 4))
```

```
GradientBoostingRegressor Model MSE 1.3261
RandomForestRegressor Model MSE 1.3099
DecisionTreeRegressor Model MSE 1.3172
```

Critical Analysis

Think critically about the different algorithms, as well as their prediction results, and characterize the trends you observe in the prediction results. Can you explain in your own words how the algorithms work? How do the results predicted by these models compare and contrast? Can you see any relationship between the algorithms and the model results? Can you see any relationship between the prediction curves and the generalization error?

I am looking for **meaningful content** here. Do not copy-and-paste model definitions off of the Internet. You should think and write critically.

Answer:

One trend that I observed in the prediction results is that the Random Forest Regressor showed significantly better results than the Decision Tree Regressor, which is interesting because the Random Forests are composed of Decision Trees. The Random Forest Regressor was not underfitting or overfitting the data, but the Decision Tree Regressor was underfitting the data, which can be seen

in the MSE. The prediction curve shows this more through comparing the three prediction curves side by side where you can see that the Random Forest model seems like it will generalize better. I noticed that the Random Forest Regressor had the longest run time when conducting a grid search, most likely due to the randomness it introduces. The Decision Tree Regressor had the shortest run time when conducting a grid search. I also noticed that the Random Forest Regressor's optimal hyperparameter for number of estimators was very high compared to that of the Gradient Boosting Regressor (900 vs 400). I suspect that the Random Forest Regressor takes a high number of estimators because adding estimators to the forest still avoids overfitting because there is more randomness due to the way Random Forest's select features. Additionally, a trend I noticed is that when I increased the learning rate, the model was slower, but better at learning from the data and in the end I did not use a high learning rate anywhere in my model.

Something that was very helpful for improving my prediction curves and MSE scores was fine-tuning my grid searches for the final search. I made my final search scope to be numbers very close to each other to really hone in on what would be the absolute best value for my hyperparameter. For example, running numbers close to each other such 890, 891, 892 vs 800, 850, 900.

Gradient Boosting Regressors work by adding predictors after fitting the next predictor to the residual errors made by the predictor that came before it. The residual error is the difference between a value and its mean. Therefore, this type of algorithm continues correcting itself by using the residual error to see where the model is guessing incorrectly, and each predictor added helps to correct the algorithm. Decision Tree Regressors work by dividing the data into parts, which we call nodes, and these nodes split the data according to their impurity (specifically, minimizing their impurity). It's called a tree because it has all these different nodes branching out and each of these helps guide the algorithm in making decisions and eventually predicting values. Lastly, Random Forests are many Decision Trees, but with more random characteristics as they search random features rather than targeted features.

The results predicted by these models differ in their MSE scores. The first model has a MSE of 1.3261 and the second model has the lowest MSE score of 1.3099. The last model has a MSE of 1.3172. The GradientBoostingRegressor and the DecisionTreeRegressor models had the highest MSEs meaning they were not performing as well as the RandomForestRegressor. These models will then also struggle to generalize data well.

Final Model Selection and Justification

Based on the arguments outlined in your critical analysis, make one final model recommendation. Which model best characterizes this data? In other words, which of these models is going to generalize better? Consider all factors. What is the type of the optimal model (i.e. `GradientBoostingRegressor`, `RandomForestRegressor`, or `DecisionTreeRegressor`)? What are the optimal model hyperparameters that should be used for training on this data set? Succinctly summarize the justification for your choice based on your arguments made above in the critical analysis section.

Answer:

The model that best characterizes this data is the RandomForestRegressor using the hyperparameters max_depth = 5, min_samples_split = 8, n_estimators = 901. I have made this choice based on the fact that the Optimal RandomForestRegressor had the lowest MSE score. The RandomForestRegressor also was not overfitting the data as much as the GradientBoostingRegressor in some sections of the prediction curve. Therefore, I believe it will generalize better.

In [ ]: