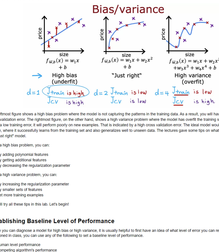
Optional Lab: Diagnosing Bias and Variance

In the previous optional lab, you saw how to evaluate a learning algorithm's performance by measuring its training and cross validation error. Given these values, you are able to quantify how well a model is doing and this helps you make a decision on which one to use for a given application. In this lab, you will build upon this process and explore some tips to improve the performance of your models. As it turns out, the training and cross validation errors can tell you what to try next to improve your models. Specifically, it will show you if you have a high bias (underfitting) or high variance (overfitting) problem. This lecture slide shown below gives an example.



The leftmost figure shows a high bias problem where the model is not capturing the patterns in the training data. As a result, you will have a high training and cross validation error. The rightmost figure, on the other hand, shows a high variance problem where the model has overfit the training set. Thus, even though it has a low training error, it will perform poorly on new examples. That is indicated by a high cross validation error. The ideal model would be the figure in the middle where it successfully learns from the training set and also generalizes well to unseen data. The lectures gave some tips on what to do next to achieve this "just right" model.

To fix a high bias problem, you can:

- try adding polynomial features
- try getting additional features
- try decreasing the regularization parameter

To fix a high variance problem, you can:

- try increasing the regularization parameter
- try smaller sets of features
- get more training examples

You will try all these tips in this lab. Let's begin!

Establishing Baseline Level of Performance

Before you can diagnose a model for high bias or high variance, it is usually helpful to first have an idea of what level of error you can reasonably get to. As mentioned in class, you can use any of the following to set a baseline level of performance:

- human level performance
- competing algorithm's performance
- guess based on experience

Real-world data can be very noisy and it's often infeasible to get to 0% error. For example, you might think that you have a high bias problem because you're getting 10% error. However, human performance on the classification task on a computer vision application. However, you saw found out that even humans can't perfect better than 10% error. If you consider this the baseline level, then you now instead have a high variance problem because you've prioritized minimizing the gap between cross validation and training error.

With this in mind, let's begin exploring the techniques to address these issues.

Imports and Lab Setup

Aside from a couple of local extensions from scikit-learn, all other functions used in this lab are found in the `utils.py` file outside this notebook. You will mostly use the same code as the last lab so you don't need to see each line here again. It mostly contains functions to split your data, as well as functions that loop over a list of parameters (e.g. degree of polynomial, regularization parameter) and plots the training and cross validation error for each one. Feel free to explore the code in the said file to see the implementation.

In [1]:
```
# for matrix math
import numpy as np

# for plotting
import matplotlib

# for the learning curves models
from sklearn.linear_model import LinearRegression, Ridge

# import all helper functions in utils.py
import utils
```

Fixing High Bias

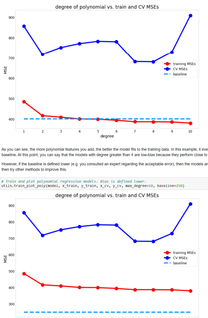
You will first look at things to try when your model is underfitting. In other words, when the training error is far worse than the baseline level of performance.

Try adding polynomial features

You've already seen this in the previous lab. Adding polynomial features can help your model learn more complex patterns in your data. Here again is an example of a plot showing how the training and cross validation errors change as you add more polynomial features. You will be using a synthetic dataset for a regression problem with one feature and one target. In addition, you will also define an arbitrary baseline performance and include it in the plot.

In [2]:
```
# split the dataset into train, cv, and test
X_train, X_cv0, X_cv, y_train, y_cv0, y_cv_test = utils.prepare_dataset('data/lab1_data1.csv')

print("The shape of the training set (input) is: {X_train.shape}")
print("The shape of the training set (target) is: {y_train.shape}")
print("The shape of the cross validation set (input) is: {X_cv.shape}")
print("The shape of the cross validation set (target) is: {y_cv.shape}")

# preview the first 5 rows
print("First 5 rows of the training inputs (1 features):\n{X_train[:5]}")

# instantiate the regression model class
model = LinearRegression()

# train and plot polynomial regression models, this is defined lower
utils.train_plot_poly(model=model, X_train=X_train, y_train=y_train, max_degree=10, baseline=200)
```

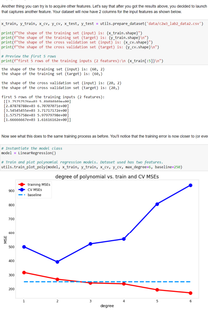
```
The shape of the training set (input) is: (30, 1)
The shape of the training set (target) is: (30,)

The shape of the cross validation set (input) is: (10, 1)
The shape of the cross validation set (target) is: (10,)

First 5 rows of the training inputs (1 feature):
[[1292.52525258]
 [1337.37373737]
 [2001.01010101]
 [1575.75757576]
 [1000.00000000]]
```



As you can see, the more polynomial features you add, the better the model fits to the training data. In this example, it even performed better than the baseline. At this point, you can say that the models with degree greater than 4 are low-bias because they perform close to or better than the baseline.

However, if the baseline is defined lower (e.g. you consulted an expert regarding the acceptable error), then the models are still considered high bias. You can then try other methods to improve this.

In [3]:
```
# train and plot polynomial regression models, this is defined lower
utils.train_plot_poly(model=model, X_train=X_train, y_train=y_train, max_degree=10, baseline=150)
```

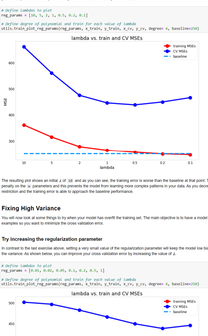


Try getting additional features

Another thing you can try is to acquire other features. Let's say that after you got the results above, you decided to launch another data collection campaign that captures another feature. Your dataset will now have 2 columns for the input features as shown below.

In [4]:
```
X_train, X_cv0, X_cv, y_train, y_cv0, y_cv_test = utils.prepare_dataset('data/lab1_data2_data.csv')

print("The shape of the training set (input) is: {X_train.shape}")
print("The shape of the training set (target) is: {y_train.shape}")
print("The shape of the cross validation set (input) is: {X_cv.shape}")
print("The shape of the cross validation set (target) is: {y_cv.shape}")

# preview the first 5 rows
print("First 5 rows of the training inputs (2 features):\n{X_train[:5]}")

# train and plot polynomial regression models, this is defined lower
utils.train_plot_poly(model=model, X_train=X_train, y_train=y_train, max_degree=10, baseline=200)
```

```
The shape of the training set (input) is: (30, 2)
The shape of the training set (target) is: (30,)

The shape of the cross validation set (input) is: (10, 2)
The shape of the cross validation set (target) is: (10,)

First 5 rows of the training inputs (2 features):
[[1292.52525258  1.67676767]
 [1337.37373737  1.73737374]
 [2001.01010101  2.60101010]
 [1575.75757576  2.04545455]
 [1000.00000000  1.41666667]]
```



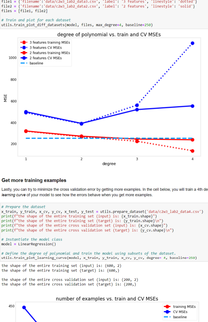
Now see what this does to the same training process as before. You'll notice that the training error is now closer to (or even better than) the baseline.

In [5]:
```
# instantiate the model class
model = LinearRegression()

# train and plot polynomial regression models, dataset used has two features
utils.train_plot_poly(model=model, X_train=X_train, y_train=y_train, max_degree=10, baseline=150)
```

Try decreasing the regularization parameter

At this point, you might want to introduce regularization to avoid overfitting. One thing to watch out for is you might make your models underfit if you set the regularization parameter too high. The cell below trains a 4th degree polynomial model using the `Ridge` class which allows you to set a regularization parameter (i.e. lambda or α). You will try several values and compare the results.

In [6]:
```
# define lambda to plot
reg_params = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1]

# define degree of polynomial and train for each value of lambda
utils.train_plot_reg_params(reg_params=reg_params, X_train=X_train, y_train=y_train, degree=4, baseline=150)
```



The resulting plot shows an initial $J_{cv} \gg J_{train}$ and as you can see, the training error is worse than the baseline at that point. This implies that it is placing a huge penalty on the w_j parameters and this prevents the model from learning more complex patterns in your data. As you decrease λ, the model lowers this restriction and the training error is able to approach the baseline performance.

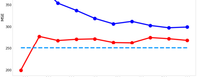
Fixing High Variance

You will now look at some things to try when your model has overfit the training set. The main objective is to have a model that generalizes well to new examples so you want to minimize the cross validation error.

Try increasing the regularization parameter

In contrast to the last exercise above, setting a very small value of the regularization parameter will keep the model low bias but might not do much to improve the variance. As shown below, you can improve your cross validation error by increasing the value of λ.

In [7]:
```
# define lambda to plot
reg_params = [0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1]

# define degree of polynomial and train for each value of lambda
utils.train_plot_reg_params(reg_params=reg_params, X_train=X_train, y_train=y_train, degree=4, baseline=150)
```



Try smaller sets of features

You've already seen in the last lab that having too many polynomial terms can result in overfitting. You can reduce the number of such terms and see where you get the best balance of training and cross validation performance. In practice, you can try adding the number of features would be helpful is when you have irrelevant features in your data. For example, patient IDs that hospitals give will not help in diagnosing a tumor so you should make sure to remove it from your training data.

To illustrate how removing features can improve performance, you will do polynomial regression for 2 datasets: the same data you used above (2 features) and another with a random 20 columns (3 features). You can preview these using the cell below. Notice that 2 columns are identical and a 3rd one is added to include random numbers.

In [8]:
```
# prepare dataset with random feature
X_train, X_cv0, X_cv, y_train, y_cv0, y_cv_test = utils.prepare_dataset('data/lab1_data2_data.csv')

# preview the first 5 rows
print("First 5 rows of the training set with 2 features:\n{X_train[:5]}")

# prepare dataset with random feature
X_train, X_cv0, X_cv, y_train, y_cv0, y_cv_test = utils.prepare_dataset('data/lab1_data2_data.csv')

# preview the first 5 rows
print("First 5 rows of the training set with 3 features (1st column is a random 20):\n{X_train[:5]}")
```

```
First 5 rows of the training set with 2 features:
[[1292.52525258  1.67676767]
 [1337.37373737  1.73737374]
 [2001.01010101  2.60101010]
 [1575.75757576  2.04545455]
 [1000.00000000  1.41666667]]

First 5 rows of the training set with 3 features (1st column is a random 20):
[[1.01010101e+02 1.57575758e+03 1.48400000e+00]
 [5.05050505e+01 1.47878788e+03 1.59797101e+00]
 [1.50505051e+02 1.92424242e+03 2.34666667e+00]
 [1.25252525e+02 1.57575758e+03 1.93767956e+00]
 [1.70707071e+02 1.00000000e+03 1.63333333e+00]]
```

Now you will train the models and plot the results. The solid lines in the plot show the errors for the data with 2 features while the dotted lines show the errors for the dataset with 3 features. As you can see, the one with 3 features has higher cross validation error especially as you introduce more polynomial terms. This is because the model is also trying to learn from the random IDs even though it has nothing to do with the target.

Another way to look at it is to observe the points at degree=4. You'll notice that even though the training error is lower with 3 features, the gap between the training error and cross validation error is a lot wider than when you only used 2 features. This should also warn you that the model is overfitting.

In [9]:
```
# define the model
model = LinearRegression()

# define properties of the 4 datasets
file0 = f"{filename}'data/lab1_data2_data.csv', 'label':'2 features', 'linestyle':'dotted'}
file1 = f"{filename}'data/lab1_data2_data.csv', 'label':'2 features', 'linestyle':'solid'}
files = [file0, file1]

# train and plot for each dataset
utils.train_plot_diff_dataset(model=model, files, max_degree=4, baseline=150)
```


Get more training examples

Lastly, you can try to minimize the cross validation error by getting more examples. In the cell below, you will train a 4th degree polynomial model then plot the learning curve of your model to see how the errors behave when you get more examples.

In [10]:
```
# prepare the dataset
X_train, X_cv0, X_cv, y_train, y_cv0, y_cv_test = utils.prepare_dataset('data/lab1_data2_data.csv')

print("The shape of the entire training set (input) is: {X_train.shape}")
print("The shape of the entire training set (target) is: {y_train.shape}")
print("The shape of the entire cross validation set (input): {X_cv.shape}")
print("The shape of the entire cross validation set (target): {y_cv.shape}")

# instantiate the model class
model = LinearRegression()

# define the degree of polynomial and train the model using subsets of the dataset.
utils.train_plot_learning_curve(model=model, X_train=X_train, y_train=y_train, degree=4, baseline=200)
```

```
The shape of the entire training set (input) is: (300, 2)
The shape of the entire training set (target) is: (300,)

The shape of the entire cross validation set (input): (300, 2)
The shape of the entire cross validation set (target): (300,)
```


From the results, it shows that the cross validation error starts to approach the training error as you increase the dataset size. Another insight you can get from this is that adding more examples will not likely solve a high bias problem. That's because the training error remains relatively flat even as the dataset increases.

Wrap Up

In this lab, you were able to practice how to address high bias and high variance in your learning algorithm. By learning how to spot these issues, you have honed your intuition on what to try next when developing your machine learning models. In the next lectures, you will look deeper into the machine learning development process and explore more aspects that you need to take into account when working on your projects. See you there!