Tyler Bryk
CPE-695 Applied Machine Learning
Homework 1

Problem 1:
Explain the following concepts:

1. Supervised Learning

In machine learning, supervised learning is a training process where data with known labels are used to map a set of inputs to a specific output. In this style of learning, a training data set will be selected where both the raw data points and the expected prediction are available. The machine learning algorithm selected for a specific problem will read in both the raw data and its labels, and then use that to fit a function to the problem. From here, the model can be validated using testing data which also contains the correct labels, then the predicted results can be compared with the labeled results, and an accuracy score can be calculated. It is also important to note that zero human supervision s required for this learning style.

2. Unsupervised Learning

When using unsupervised learning, a raw data set is fed into a model where the labels or outcomes of that data set are unknown. While unsupervised learning can be used to make predictions, it is typically used to uncover trends and patterns in big data sets. Here, the raw data points are consumed with no labels, it is then up to the model / algorithm to find patterns in those raw data points. Unlike supervised learning, human interaction is required here to interpret and make use of any patterns that are discovered by the unsupervised learning model.

3. Online Learning

Online learning (also known as incremental learning) is an approach that builds on top of supervised and unsupervised learning, where new training data can be learning from in real time. In this approach, rather than using a known training data set, the model takes in a continuous stream of input data. As each new data point enters the model, the model can be retrained to reflect the knowledge gained from that new data point. This learning style is extremely powerful because it provides real time updates to a model, so the model is always up-to-date with its predictions.

4. Batch Learning

Batch learning is very similar to online learning, with one key difference: the model cannot take in a continuous stream of data. Rather, the model takes in periodic ‘batches’ of new training data at a specified intervals. In some scenarios, it is not possible for a model to leverage online learning, so in those cases, the model can be briefly taken offline and fed the batch training data. This learning style is still powerful because it enables a model to learn as new training data becomes available, but it is not as efficient as online learning because there is latency between the availability of new data and the model learning from that new data.

5. Model-based Learning

Model-based learning, also known as reinforcement learning is an approach where actions are observed from an environment in order to stimulate learning. For example, in reinforcement learning, the model is given the option to take certain actions based on data that is inputted. Based on these actions, either a penalty or reward is given out which the model can learn from. Essentially, the optimal behavior of a system can be achieved by having the model learn from various outcomes that happen as a result from the models predictions / actions.

6. Instance-based Learning

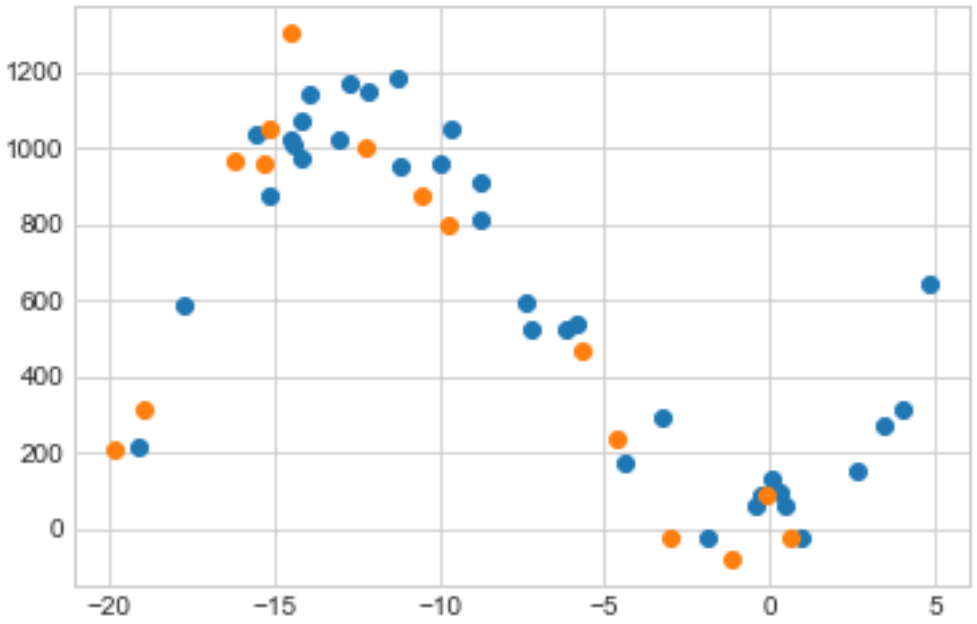
Typical machine learning algorithms often generalize or form an abstraction of different training data points that are supplied to the model. In instance-based learning, this is not the case. Here, instead of generalizing new data points as they come in, the model explicitly compares the new data instances with previous instances seen in past training data. This style of learning directly compares new data with old data, and it does not perform any operations on the data before or after making these comparisons.

Tyler Bryk

September 25, 2020

[1]:
```
# Import Libraries
import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
plt.style.use('seaborn-whitegrid')
```

[8]:
```
# Define Simulated Data from Problem
noise_scale = 100
number_of_samples = 50
x = 25 * (np.random.rand(number_of_samples, 1) - 0.8)
y = 5 * x + 20 * x**2 + 1 * x**3 + noise_scale * np.random.
 ↪randn(number_of_samples, 1)

# Split Training Data (70%) and Test Data (30%)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30,␣
 ↪random_state=10413641)

# Plot Training Data (Blue) and Test Data (Orange)
plt.scatter(x_train, y_train)
plt.scatter(x_test, y_test)
plt.show()
```



[10]:
```
# Squeeze 'x' and 'y' into 1D arrays
x_train = np.squeeze(x_train)
x_test = np.squeeze(x_test)
y_train = np.squeeze(y_train)
y_test = np.squeeze(y_test)
```

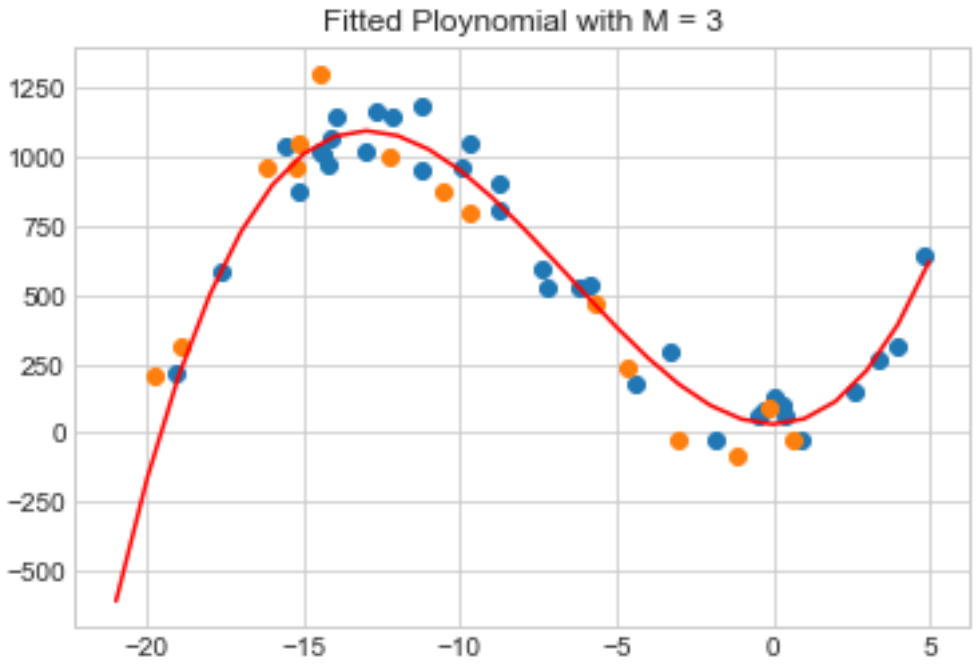
Part 1

Please plot the noisy data and the polynomial you found (in the same figure). You can use any value of m selected from 2, 3, 4, 5, 6.

For this problem, since the noisy data follows a third degree polynomial, we will select m=3 for the polyfit() function. The original noisy data can be seen below (Blue) overlayed by the fitted polynomial (Red). It can be observed that for m $>$ 3 the same polynomial shape will be found, since the original data is only third order.

[11]:
```
# Fit Training Data using polyfit() Function
poly = np.poly1d(np.polyfit(x_train, y_train, 3))
xfit = []
yfit = []
for i in range(math.floor(min(x_train))-1, math.ceil(max(x_train))+1):
  xfit.append(i)
  yfit.append(poly(i))

plt.title('Fitted Ploynomial with M = 3')
plt.plot(xfit, yfit, 'r')
plt.scatter(x_train, y_train)
```

```
plt.scatter(x_test, y_test)
plt.show()
```



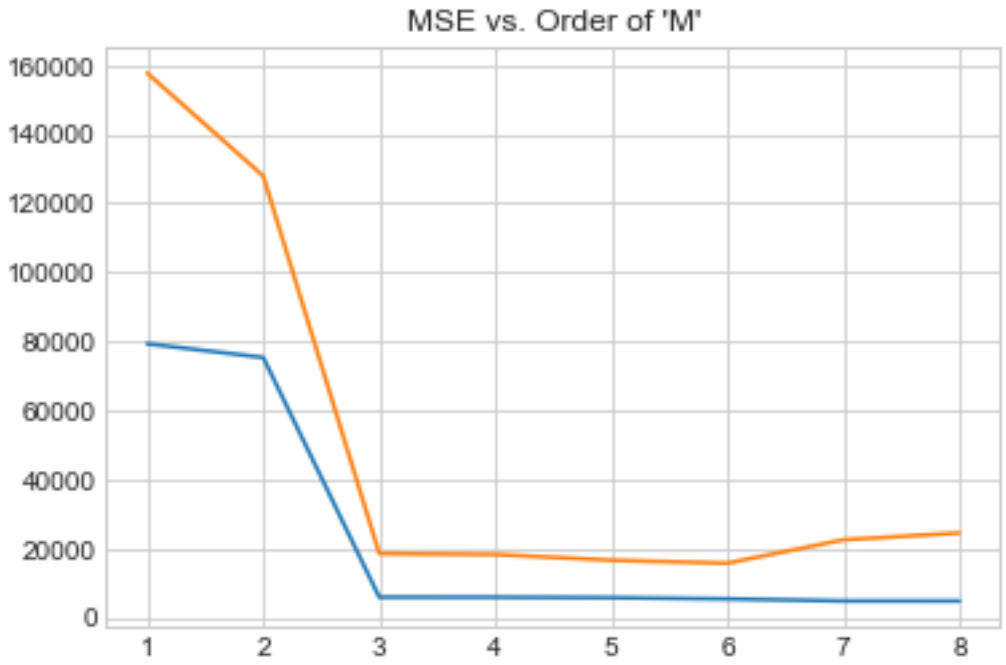
Part 2

Plot MSE versus order m, for m = 1, 2, 3, 4, 5, 6, 7, 8 respectively. Identify the best choice of m.

Based on the results shown below, the mean-squared error decreases as the order of 'm' increases. For our specific plot, an order of m=3 is sufficient to achieve a minimal error, and prevent overfitting. This result would also be expected since the original data follows a third order polynomial. As shown in the graph, the testing data (Orange) is slightly higher than the training data (Blue) which also makes sense because we would expect the model to perform slightly worse on unseen testing data.

[12]:
```
# Plot MSE vs. Order 'M'
mseTrain = []
mseTest  = []
for d in range(8):
  poly = np.poly1d(np.polyfit(x_train, y_train, (d+1)))
  yfitTrain = []
  yfitTest = []
  for i in range(len(x_train)):
    yfitTrain.append(poly(x_train[i]))
  for i in range(len(x_test)):
    yfitTest.append(poly(x_test[i]))
  mseTrain.append(np.square(y_train - yfitTrain).mean())
```

```
  mseTest.append(np.square(y_test - yfitTest).mean())

plt.plot([1,2,3,4,5,6,7,8], mseTrain)
plt.plot([1,2,3,4,5,6,7,8], mseTest)
plt.title("MSE vs. Order of 'M'")
plt.show()
```



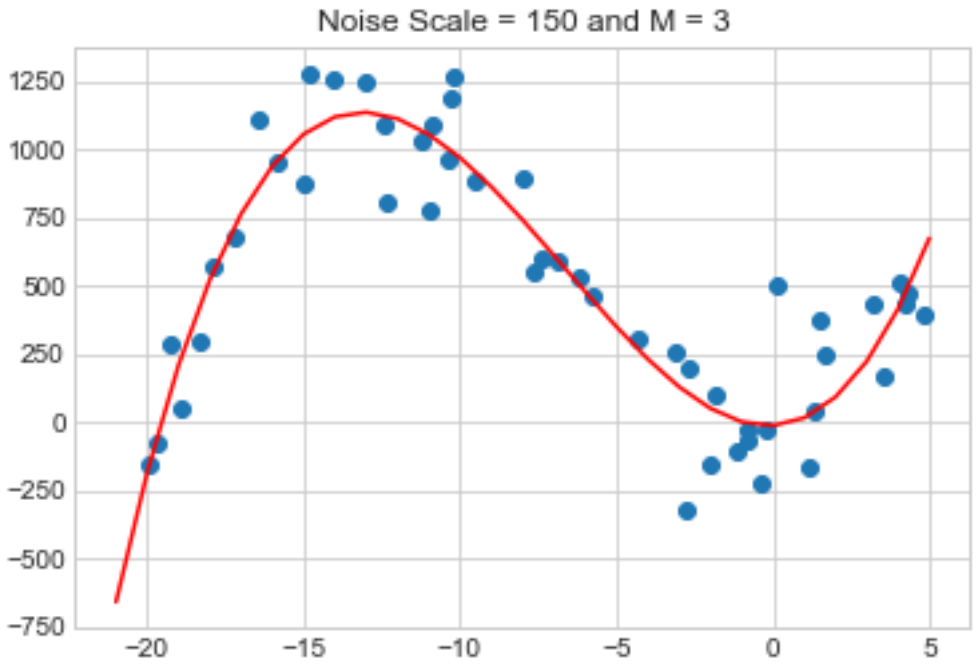
Part 3

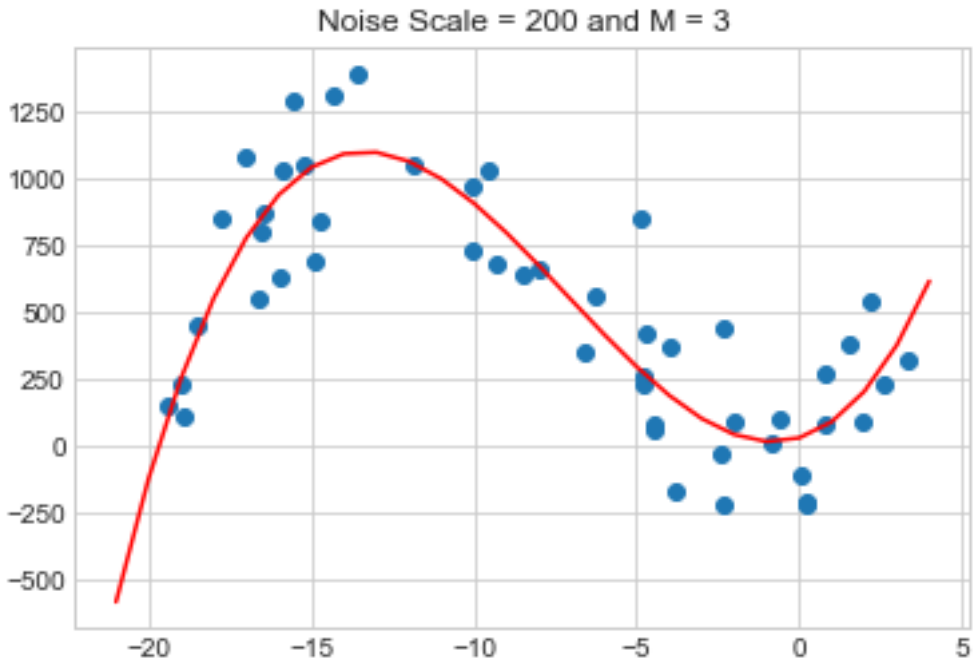
Change variable noise_scale to 150, 200, 400, 600, 1000 respectively, re-run the algorithm and plot the polynomials with the m found in 2). Discuss the impact of noise scale to the accuracy of the returned parameters. [You need to plot a figure like in 1) for each choice of noise_scale.]

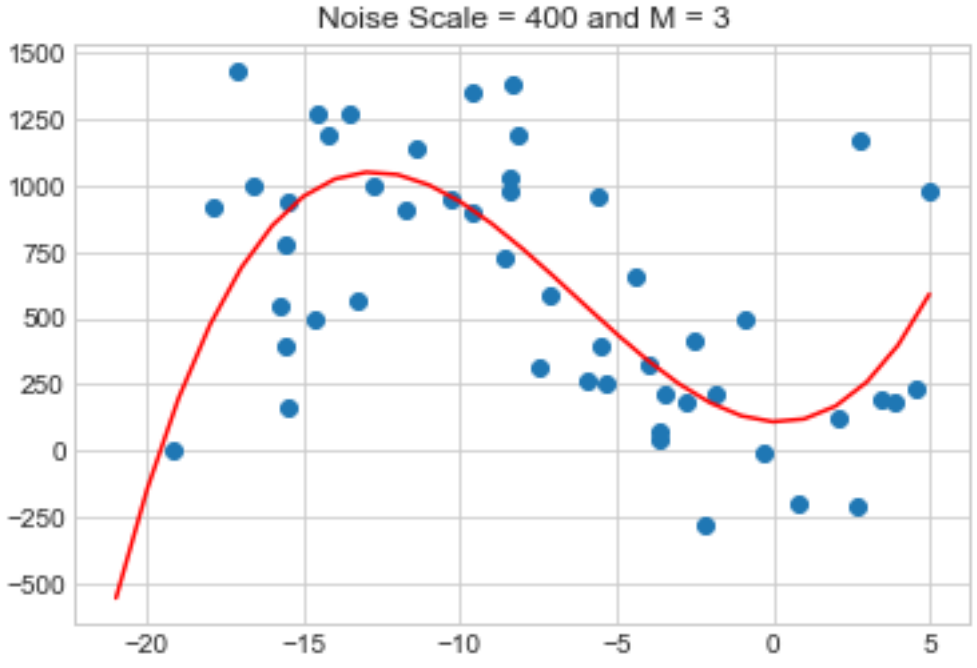
In this problem, the noise scale of the original ploynomial was quadratically sampled from 150 to 1000. The original noisy data can be seen below (Blue) overlayed by the fitted polynomial (Red). As the noise scale increased, the polynomial data became more spread out on the graph, making it harder for the polyfit() function to accurately fit a curve. In the first graph (noise: 150), the data points are very tight, and the fitted resultant polynomial has sharp modes that most accurately reflect the data. As the noise scale increases, the fitted ploynomial begins to smoothen out, and the modes become less apparent. In the last graph (noise: 1000), the data points are so spread out that the resultant curve is practically a straight horizontal line. **Ultimately, as the noise scale increases, the accuracy of the fitted curve decreases.**

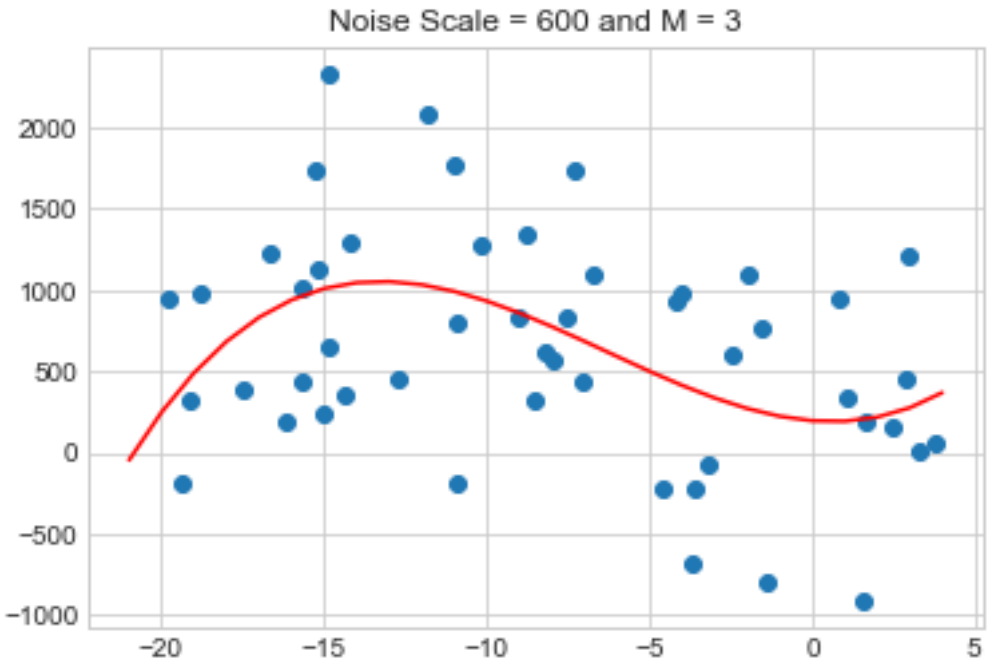
[13]:
```
noise = [150, 200, 400, 600, 1000]

for n in noise:
```

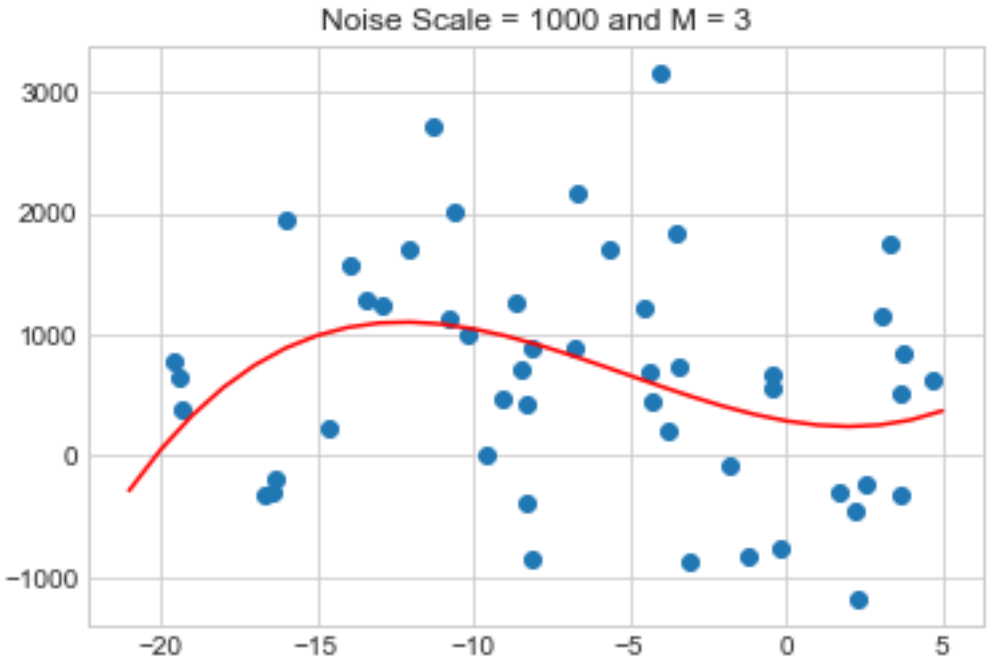
```
x = 25 * (np.random.rand(number_of_samples, 1) - 0.8)
y = 5 * x + 20 * x**2 + 1 * x**3 + n * np.random.randn(number_of_samples, 1)
x = np.squeeze(x)
y = np.squeeze(y)
poly = np.poly1d(np.polyfit(x, y, 3))
xfit = []
yfit = []
for i in range(math.floor(min(x))-1, math.ceil(max(x))+1):
  xfit.append(i)
  yfit.append(poly(i))

plt.title(('Noise Scale = ' + str(n) + ' and M = 3'))
plt.plot(xfit, yfit, 'r')
plt.scatter(x, y)
plt.show()
```




Noise Scale = 200 and M = 3
1250
1000
750
500
250
0
−250
−500
−20
−15
−10
−5
0
5


Noise Scale = 400 and M = 3
1500
1250
1000
750
500
250
0
−250
−500
−20
−15
−10
−5
0
5





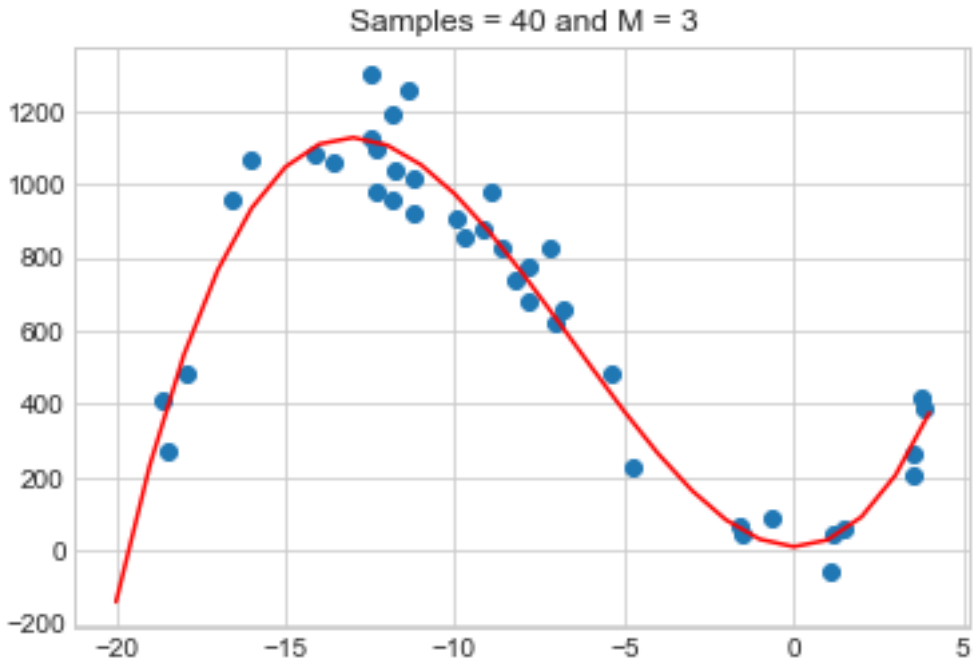
Part 4

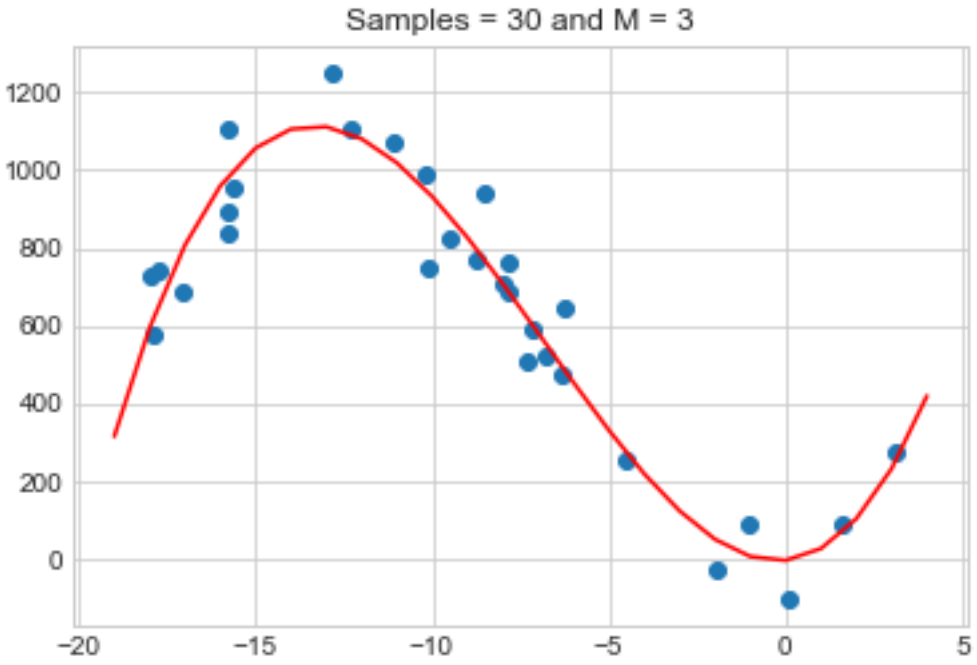
Change variable number_of_samples to 40, 30, 20, 10 respectively, re-ran the algorithm and plot the polynomials with the m found in 2). Discuss the impact of the number of samples to the

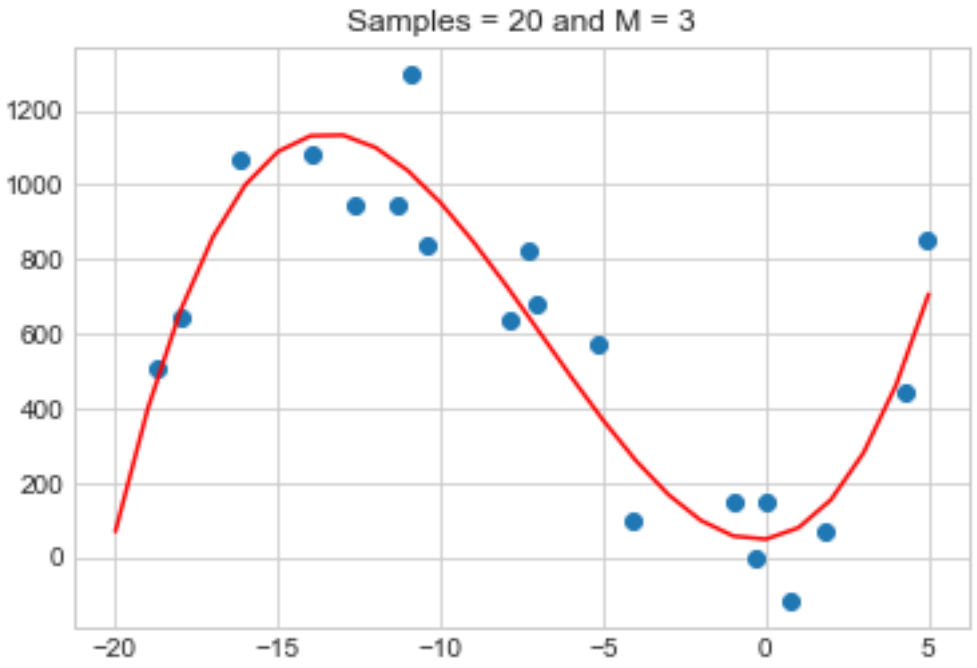
accuracy of the returned parameters. [You need to plot a figure like in 1) for each choice of number_of_samples.]

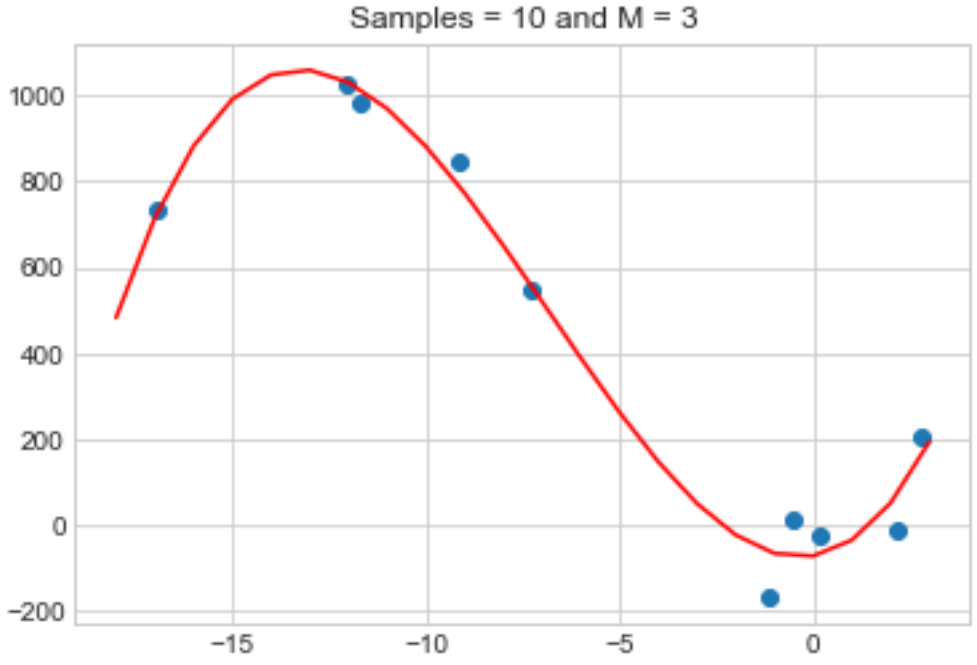
In this problem, the number of samples was linearly scaled from 40 to 10. The original noisy data can be seen below (Blue) overlayed by the fitted polynomial (Red). As the number of samples decreased, the polyfit() function had an easier time minimizing the distance of the fitted curve to each of the sample points. In the first graph (sample: 40), the data points are very spread out, and the fitted resultant polynomial left large gaps between some of the data points. In the last graph (sample: 10), the fitted curve passes through almost every data point. **Ultimately, as the sample size decreases, the accuracy of the fitted curve increases.**

[14]:
```
samples = [40, 30, 20, 10]

for n in samples:
  x = 25 * (np.random.rand(n, 1) - 0.8)
  y = 5 * x + 20 * x**2 + 1 * x**3 + 100 * np.random.randn(n, 1)
  x = np.squeeze(x)
  y = np.squeeze(y)
  poly = np.poly1d(np.polyfit(x, y, 3))
  xfit = []
  yfit = []
  for i in range(math.floor(min(x))-1, math.ceil(max(x))+1):
    xfit.append(i)
    yfit.append(poly(i))

  plt.title(('Samples = ' + str(n) + ' and M = 3'))
  plt.plot(xfit, yfit, 'r')
  plt.scatter(x, y)
  plt.show()
```


Samples = 40 and M = 3
1200
1000
800
600
400
200
0
−200
−20
−15
−10
−5
0
5


Samples = 30 and M = 3
1200
1000
800
600
400
200
0
−20
−15
−10
−5
0
5


Samples = 20 and M = 3
1200
1000
800
600
400
200
0
−20
−15
−10
−5
0
5


Samples = 10 and M = 3
1000
800
600
400
200
0
−200
−15
−10
−5
0