[CSE302] Introduction to Machine Learning

< Assignment 1 >

(Deadline: 2022-04-08)

202123008 Jinmin Kim (김진민)

Phone: 010-6266-6099

Mail: rlawlsals@dgist.ac.kr

1. Plot 10 samples, spaced uniformly in range [0, 1], with the function sin(2πx) with a Gaussian noise like below.

** 주요코드

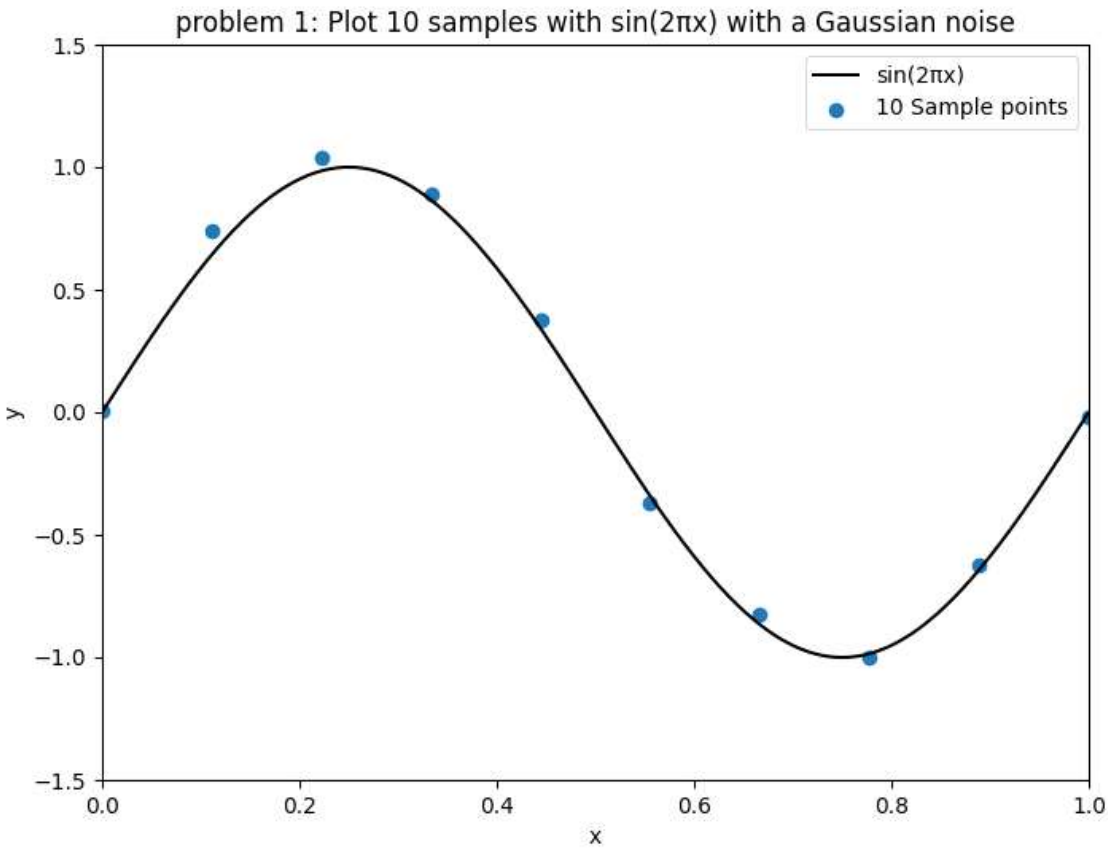
```python
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

- Problem 1 ~ Problem 5 에서 사용할 라이브러리 임포트

```python
# Define 10 random sample data
datalen = 10
Sample_x = np.linspace(0, 1, datalen)
Sample_y = np.array([np.sin(2*np.pi*Sample_x[k]) + random.gauss(0.0, 0.05) for k in range(datalen)])
Sample_x = np.array([Sample_x]).T
Sample_y = np.array([Sample_y]).T

# Define sine graph
x = np.arange(0, 1, 0.001)
y = np.sin(2*np.pi*x)

# Plot
plt.plot(x, y, label='sin(2πx)', color='k')
plt.scatter(Sample_x, Sample_y, label='10 Sample points')
```

- 0~1사이를 균일한 간격으로 나누어 10개의 Sample_x를 생성함.
- random.gauss(0.0, 0.05) 함수를 통해 평균이 0이고 표준편차가 0.05인 가우시안 노이즈를 랜덤하게 추가하였음.

** 결과



** 분석

- Sine function(검정선)과 10개의 샘플 포인트(파란점)가 생성되었음.
- 샘플 포인트의 노이즈는 코드를 실행할 때마다 랜덤하게 설정됨.

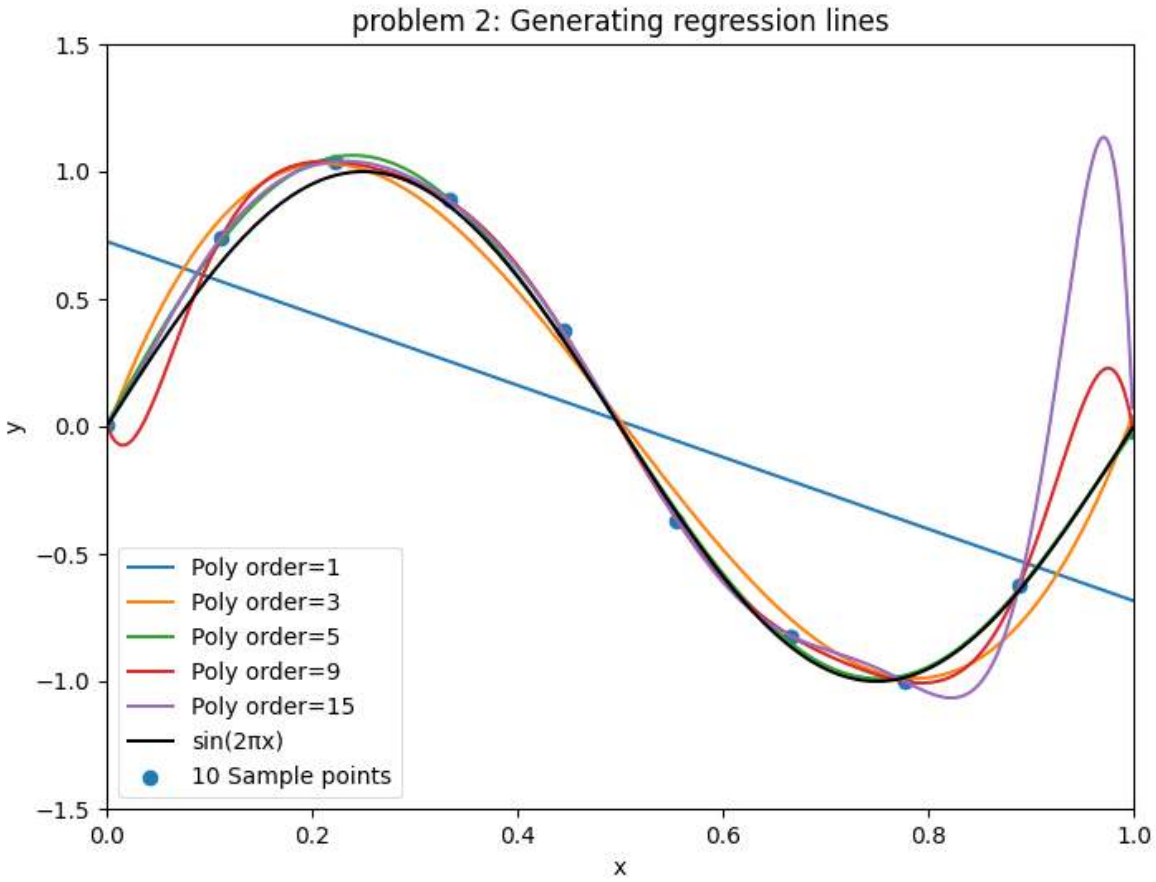
2. Generate regression lines with polynomial basis function with order 1, 3, 5, 9, and 15.

** 주요코드

```
def poly_generation(order, sam_x, sam_y):
    model = LinearRegression()

    poly = PolynomialFeatures(degree = order)
    x_poly = poly.fit_transform(sam_x)
    model.fit(x_poly, sam_y)
    y_p = model.predict(x_poly)

    print(order, '차항 변환 데이터 : ', x_poly.shape)

    return y_p


def poly_drawing(points_x, points_y, order):
    fit = np.polyfit(points_x, points_y, order)
    fi = np.poly1d(fit)

    return fi
```

- 원하는 차수에 대한 polynomial function을 정의함.
- poly_generation 함수를 사용하면 input 데이터에 대해서 원하는 차수까지의 feature를 만들고, 그 feature에 대한 label의 prediction 값을 출력함.
- poly_drawing 함수를 사용하면 좌표에 대한 fitting curve를 디스플레이 할 수 있음.

** 결과



** 분석

- 위 결과 그래프와 같이, sine 함수에 대해서 1차 함수는 직선으로 그려졌고, 3차 함수부터는 사인 함수와 유사한 그래프가 그려짐.
- 9차 함수와 15차 함수의 경우 overfitting이 발생하는 것을 알 수 있음.

- 아래와 같이 n차항 변환 데이터를 print해서 확인해본 결과, 각각 bias 가 추가된 polynomial feature를 얻을 수 있었음.

```
1 차항 변환 데이터 :  (10, 2)
3 차항 변환 데이터 :  (10, 4)
5 차항 변환 데이터 :  (10, 6)
9 차항 변환 데이터 :  (10, 10)
15 차항 변환 데이터 :  (10, 16)
```

- 예를 들어 3차항 변환 데이터의 경우 $1, x, x^2, x^3$ 로, bias가 포함된 4개의 레이블이 생성됨.

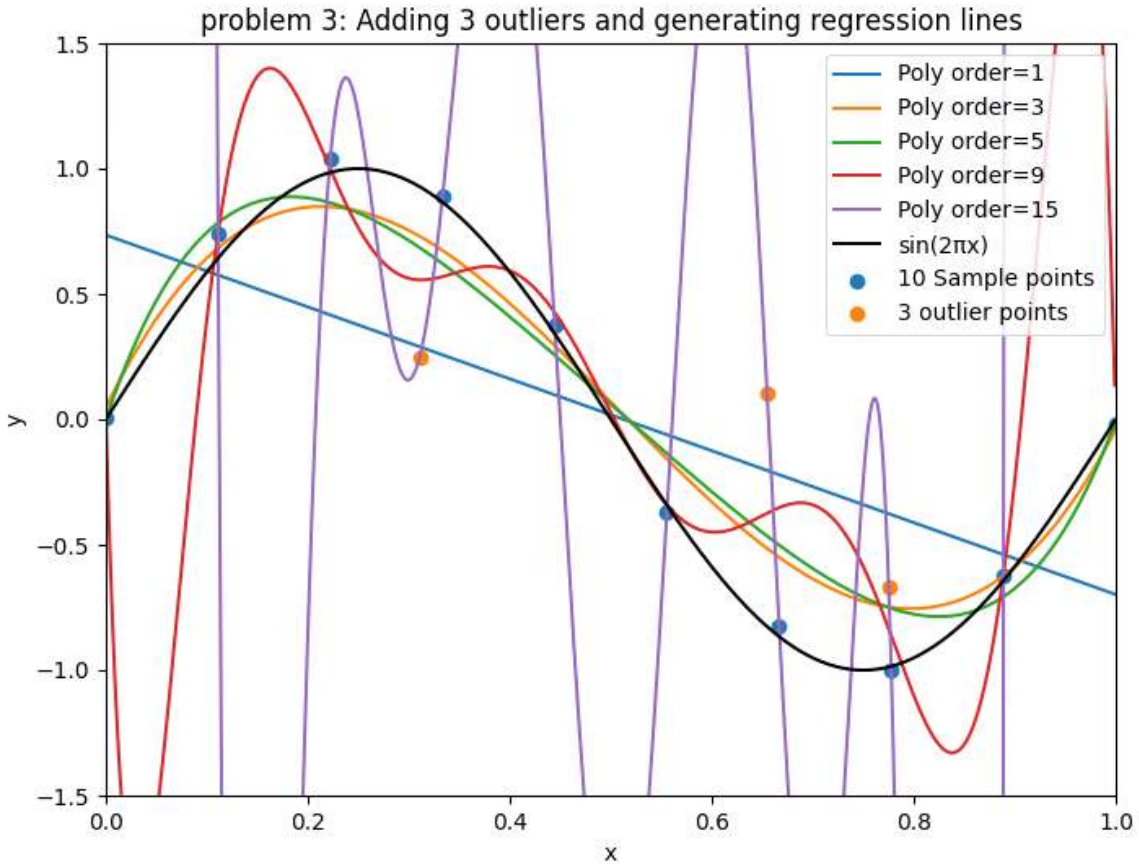
3. Add 2 or 3 points of exceptional outliers that do not follow sin(2πx) and then generate regression lines with polynomial basis function with order 1, 3, 5, 9, and 15.

** 주요코드

```
# Define 3 outlier points
outlier_x = [random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1)]
outlier_y = [random.gauss(0, 0.5), random.gauss(0, 0.5), random.gauss(0, 0.5)]
```

- x축으로 0부터 1까지의 범위에서, 평균이 0이고 표준편차가 0.5인 가우시안 노이즈를 y축에 생성하여 3개의 outlier points를 생성함.
- 해당 outlier points를 기존 데이터에 추가하여, 총 13개의 샘플 데이터를 이용함.

** 결과



** 분석

- 아래와 같이 n차항 변환 데이터를 print해서 확인해본 결과, 문제 2와 마찬가지로 13개의 데이터에 대해서 각각 bias가 추가된 polynomial feature를 얻을 수 있었음.

```
1 차항 변환 데이터 :  (13, 2)
3 차항 변환 데이터 :  (13, 4)
5 차항 변환 데이터 :  (13, 6)
9 차항 변환 데이터 :  (13, 10)
15 차항 변환 데이터 :  (13, 16)
```

- 결과 그래프를 보면 문제 2에서는 9차 함수와 15차 함수에서 약간의 overfitting이 발생했었지만, 문제 3에서는 outlier로 인해 overfitting의 정도가 훨씬 심한 모습을 볼 수 있음.

4. For the case including the outliers, generate the regression lines with the L2 regularization term with order 9
and 15. Show how the lines are changed with respect to λ .
Generate the regression lines with the L1
regularization term and compare the lines with L2 regularization.

** 주요코드

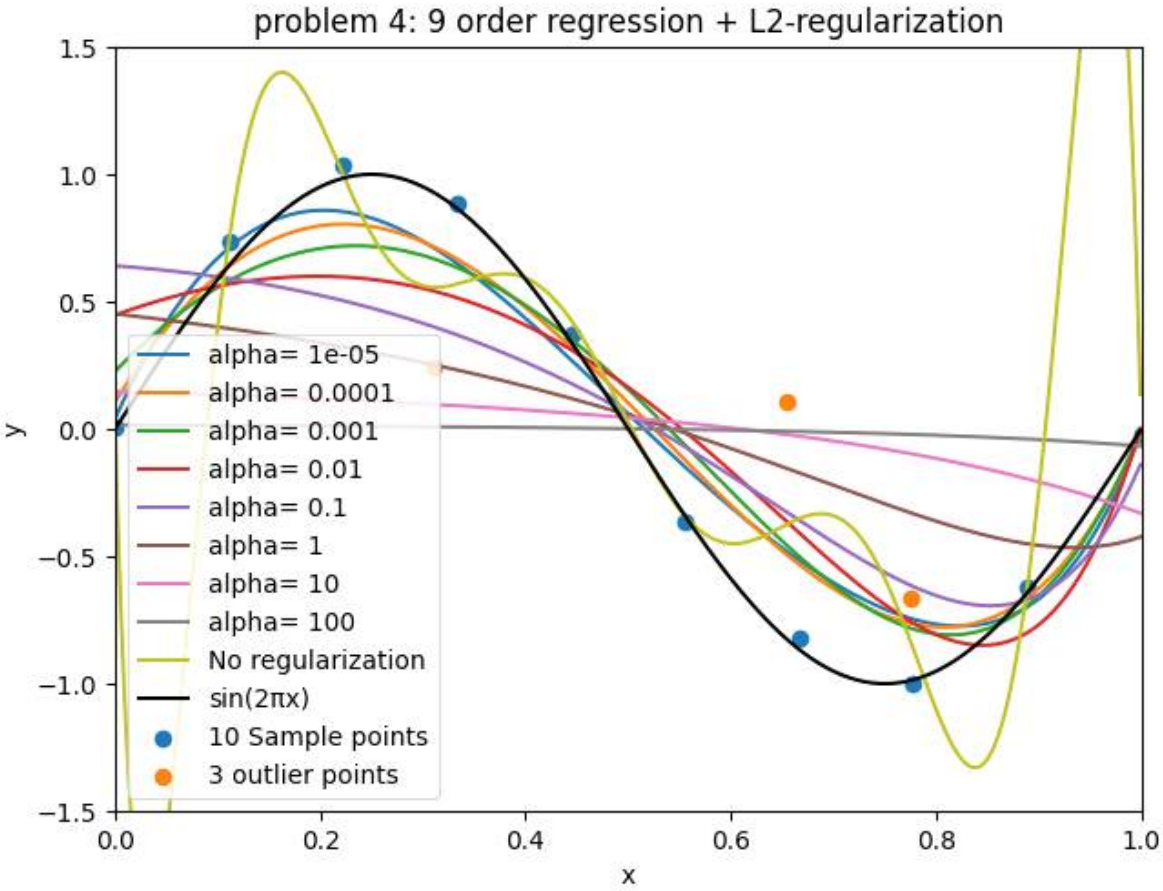
```
def poly_generation_with_l2_regularization(order, sam_x, sam_y, alpha_val):

    poly = PolynomialFeatures(degree=order)
    x_poly = poly.fit_transform(sam_x)

    model = Ridge(alpha=alpha_val)
    model.fit(x_poly, sam_y)
    y_p = model.predict(x_poly)

    print('Alpha value', alpha_val)

    return y_p


def poly_generation_with_l1_regularization(order, sam_x, sam_y, alpha_val):
    poly = PolynomialFeatures(degree=order)
    x_poly = poly.fit_transform(sam_x)

    model = Lasso(alpha=alpha_val)
    model.fit(x_poly, sam_y)
    y_p = model.predict(x_poly)

    print('Alpha value', alpha_val)

    return y_p
```

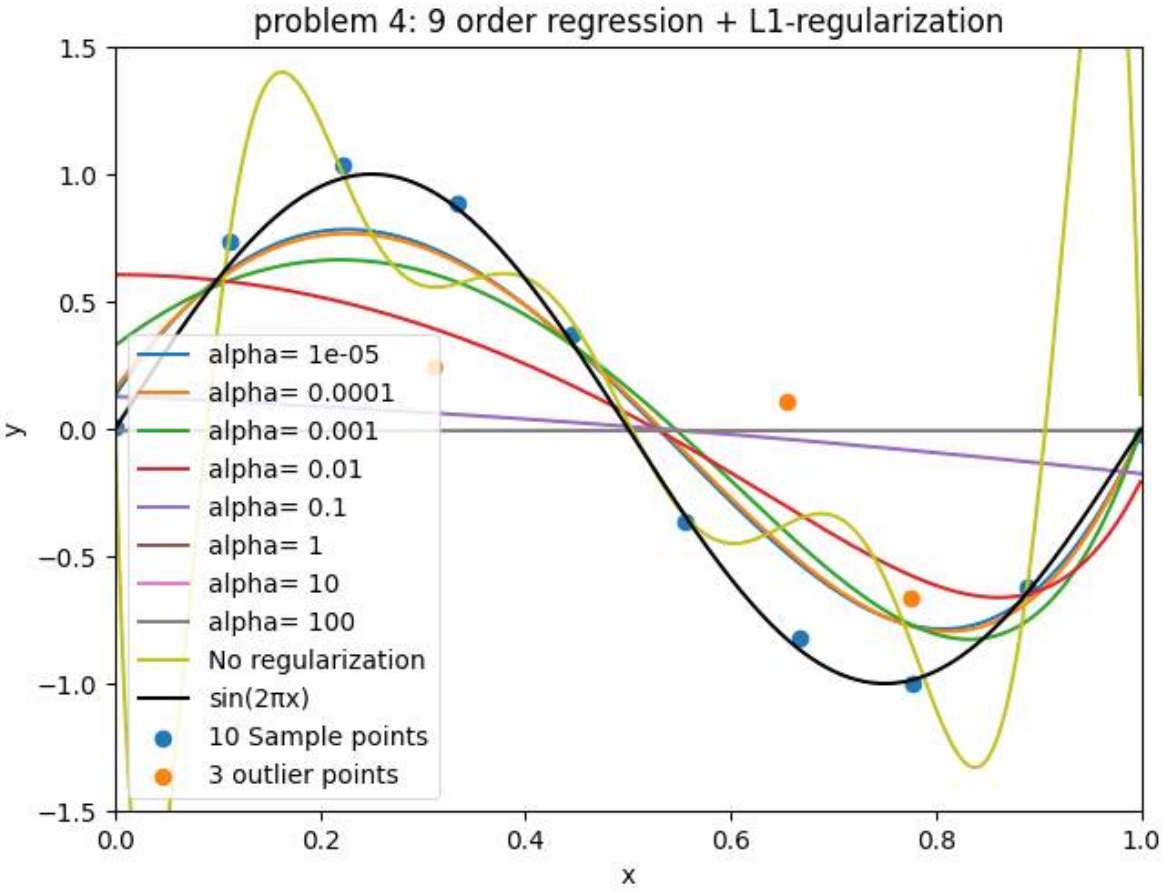
- L2 norm에 대해서는 sklearn.linear_model의 Ridge 라이브러리를 사용했고,
  L1 norm에 대해서는 sklearn.linear_model의 Lasso 라이브러리를 사용함.
- 해당 라이브러리들에서는 수업시간에 배운 λ값이 α값으로 정의되어 있음. 따라서 추후 코딩된 alpha values은 λ값을 뜻함.

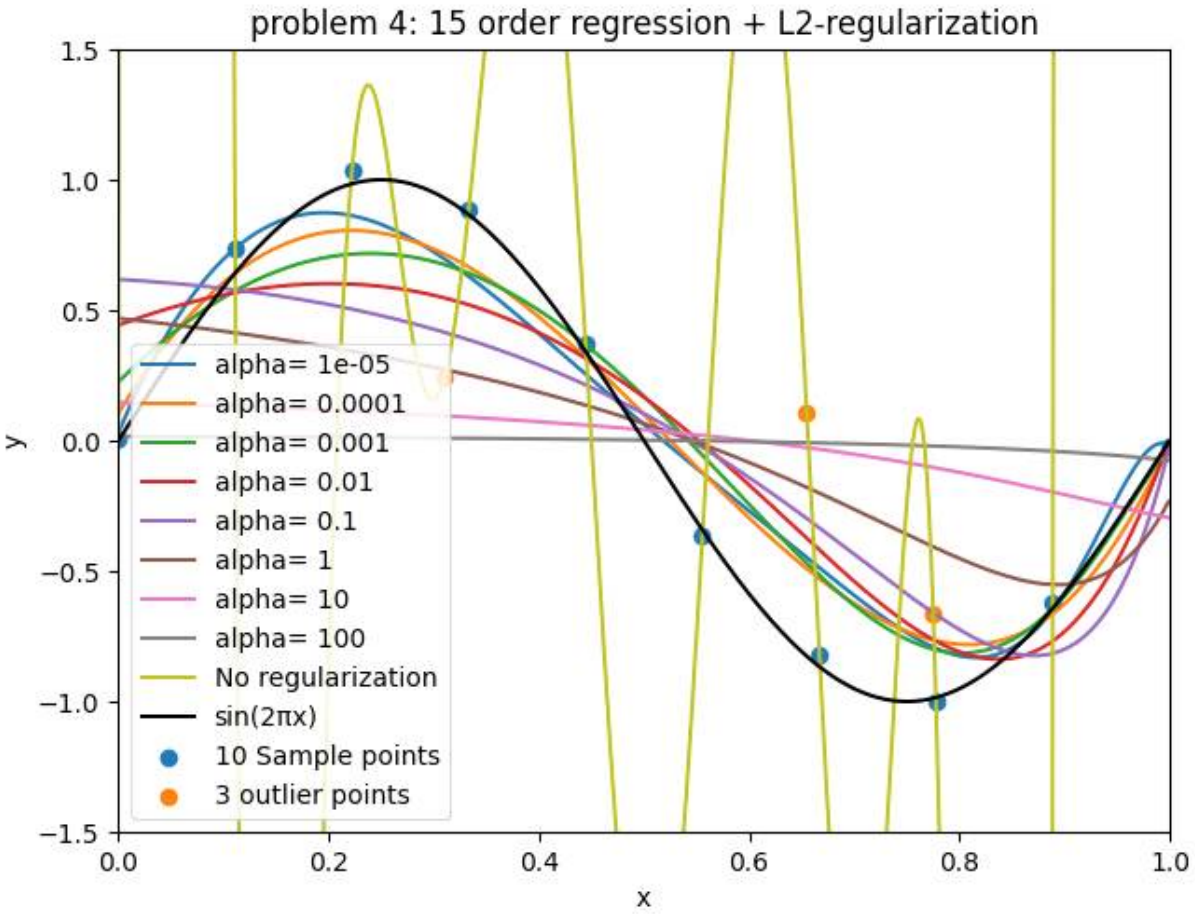
```
alpha_values = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
```

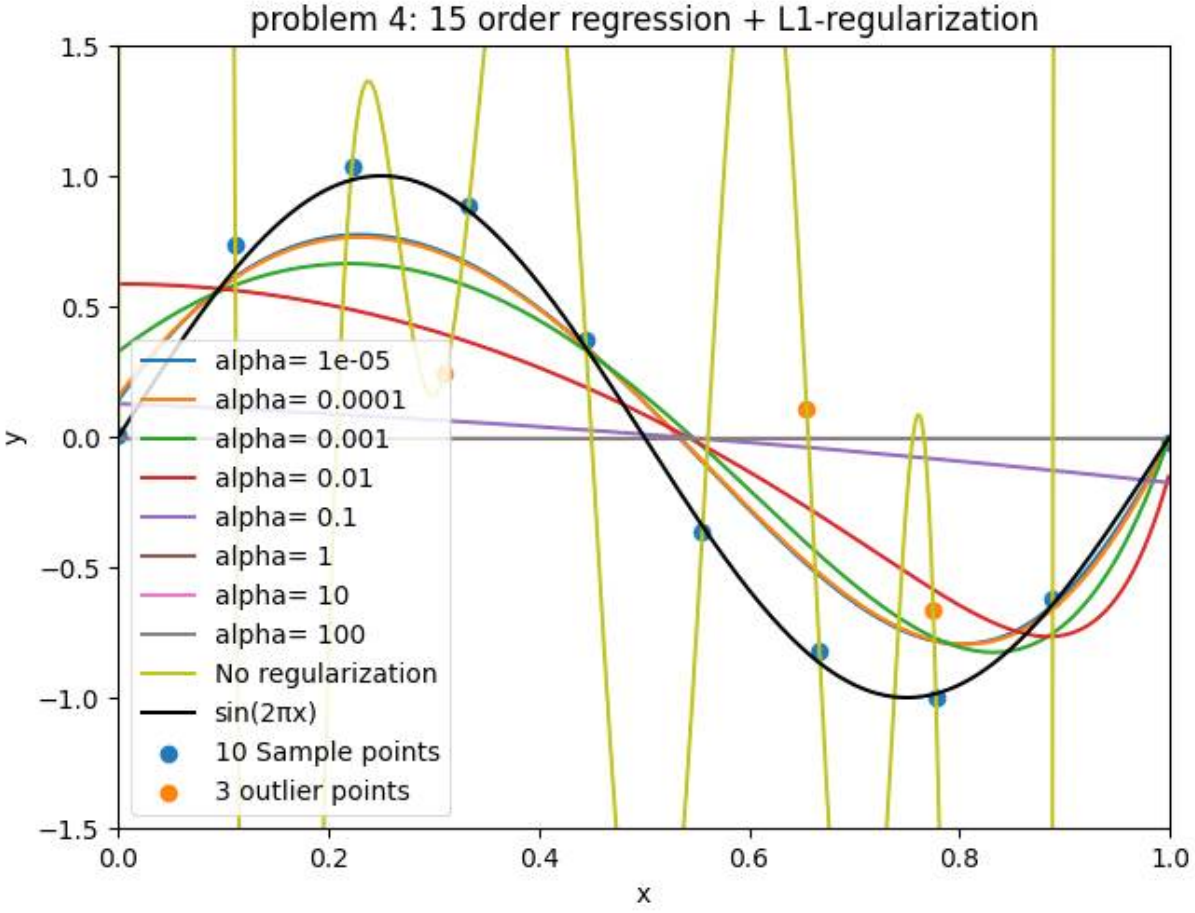
- α값을 위와 같이 설정하여 α값에 따라 그래프가 어떻게 학습되는지 보고자 함.

** 결과


problem 4: 9 order regression + L2-regularization
alpha= 1e-05
alpha= 0.0001
alpha= 0.001
alpha= 0.01
alpha= 0.1
alpha= 1
alpha= 10
alpha= 100
No regularization
sin(2πx)
10 Sample points
3 outlier points
x
y


problem 4: 9 order regression + L1-regularization
alpha= 1e-05
alpha= 0.0001
alpha= 0.001
alpha= 0.01
alpha= 0.1
alpha= 1
alpha= 10
alpha= 100
No regularization
sin(2πx)
10 Sample points
3 outlier points
x
y


problem 4: 15 order regression + L2-regularization
alpha= 1e-05
alpha= 0.0001
alpha= 0.001
alpha= 0.01
alpha= 0.1
alpha= 1
alpha= 10
alpha= 100
No regularization
sin(2πx)
10 Sample points
3 outlier points
x
y


problem 4: 15 order regression + L1-regularization
alpha= 1e-05
alpha= 0.0001
alpha= 0.001
alpha= 0.01
alpha= 0.1
alpha= 1
alpha= 10
alpha= 100
No regularization
sin(2πx)
10 Sample points
3 outlier points
x
y

** 분석

- 9차 함수의 L2 regularization을 보면, α값이 100부터 0.00001까지 작아질 때 일정한 속도로 그래프가 sine함수와 유사해지는 것을 볼 수 있음.
- 하지만 9차 함수의 L1 regularization을 보면 α값이 작아짐에 따라서 낮은 속도로 sine함수와 유사해지다가, α값이 0.01부터는 빠른 속도로 sine함수와 유사해짐을 알 수 있음.
- 마찬가지로 15차 함수의 L2 regularization과 L1 regularization을 보면 9차 함수와 같은 경향을 볼 수 있음.

- 공통적으로 regularization의 α값이 0.0001 이하로 작아졌을 때, 고차 함수에 대해서 overfitting이 해소되는 경향을 볼 수 있었음.

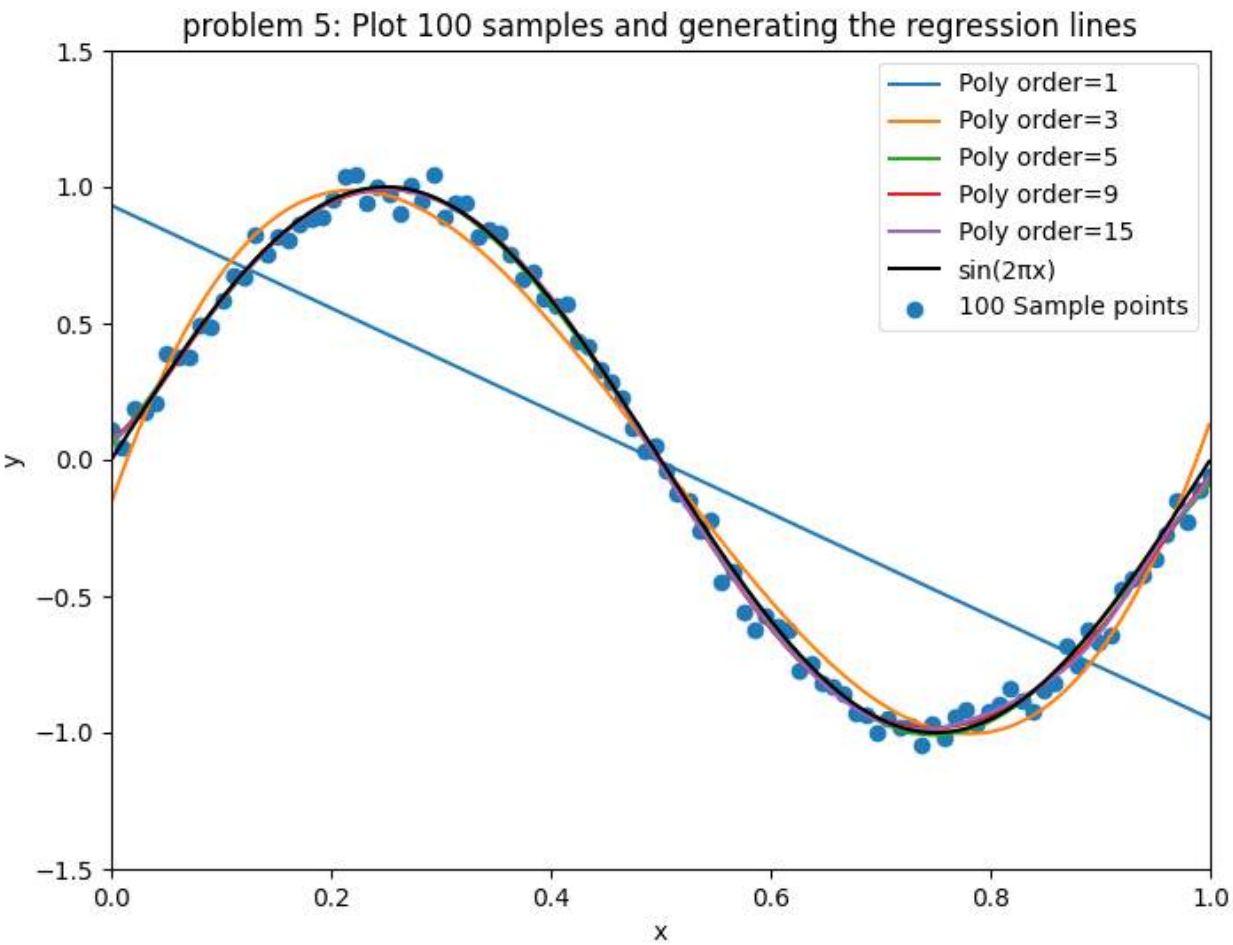
5. Plot 100 samples with the function sin(2πx) instead of 10 samples, and then generate the regression lines with order 1, 3, 5, 9, and 15.

** 주요코드

```
datalen = 100
Sample_x3 = np.linspace(0, 1, datalen)
Sample_y3 = np.array([np.sin(2*np.pi*Sample_x3[k]) + random.gauss(0.0, 0.05) for k in range(datalen)])
Sample_x3 = np.array([Sample_x3]).T
Sample_y3 = np.array([Sample_y3]).T
```

- 데이터 크기를 100개의 늘려서, 문제 2와 동일한 코드로 polynomial feature를 생성함.

** 결과



** 분석

- 데이터 개수가 많아짐에 따라, 수업시간에 배운 것처럼 고차 함수에 대해서 overfitting이 해소되었음을 알 수 있음.