PolynomialNoiselessDemo

July 8, 2021

Polynomial Noiseless Target

Consider the following learning scenario (urge you to tweak the parameters)

- Y = f(x) where f(x) is a polynomial of degree 50
- Suppose N=10 datapoints are given
- We have a choice to make in terms of the hypothesis set
- Assume that 2 degree hypothesis and 10 degree hypothesis

Which does better in among the two hypothesis set?

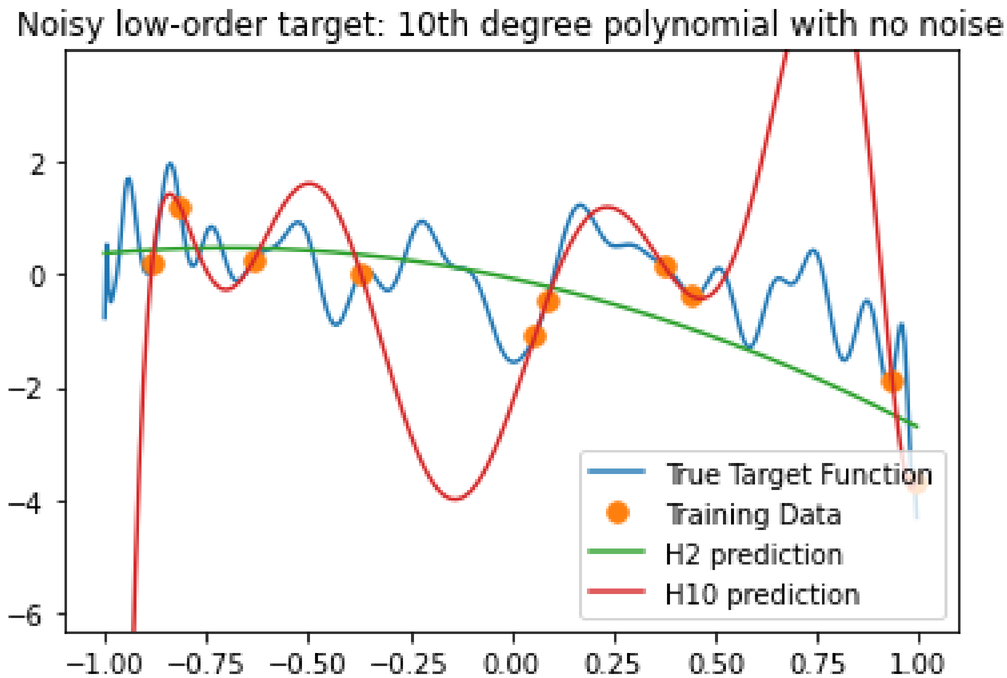
[1]:
```
import numpy as np
import scipy.special
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression


def generate_random_mixing_coeff(K):
    return( np.random.normal(0, 1, K+1))

def generate_normalizing_coeff(K):
    ret = np.zeros(K+1)
    for idx in range(K+1):
        if idx == 0:
            ret[idx] = 1
        else:
            ret[idx] = ret[idx -1] + 1/(2*idx+1)
    return np.sqrt(1/ret)

def generate_multiplying_coeff(K):
    valsToMul =  generate_normalizing_coeff(K) *generate_random_mixing_coeff(K)
    #valsToMul = normalize_legendre_coefficients(aqs)
    return valsToMul

def generate_poly(K,random_poly_coeff):
    return np.polynomial.legendre.Legendre(random_poly_coeff)

def generate_data_set(N, random_poly_coeff):
```

```python
    xs = np.random.uniform(-1,1, N)
    ys = generate_poly(K,random_poly_coeff)(xs)
    return xs, ys

def estimateEoutOnce(K,N,random_poly_coeff, degA, degB):
    polyRandomTarget = generate_poly(K,random_poly_coeff)
    xvals = np.arange(-1,1,0.005) #Legendre polynomials are defined within [-1,␣
 ↪1]
    yvals = polyRandomTarget(xvals)
    ds_x, ds_y = generate_data_set(N,random_poly_coeff)

    pA = PolynomialFeatures(degA)
    pB = PolynomialFeatures(degB)
    xA = pA.fit_transform(ds_x.reshape(-1, 1) )
    xB = pB.fit_transform(ds_x.reshape(-1, 1) )

    mA = LinearRegression()
    mA.fit(xA, ds_y)
    yvalsA = mA.predict(pA.fit_transform(xvals.reshape(-1, 1) ))

    mB = LinearRegression()
    mB.fit(xB, ds_y)
    yvalsB = mB.predict(pB.fit_transform(xvals.reshape(-1, 1) ))

    EoutA = np.mean(np.square(yvals-yvalsA))
    EoutB = np.mean(np.square(yvals-yvalsB))
    return EoutA, EoutB

def estimateEoutSeveral(K,N,random_poly_coeff, degA, degB, reps= 1000):
    EoutA = 0
    EoutB = 0
    for i in range(reps):
        currA, currB = estimateEoutOnce(K,N,random_poly_coeff, degA, degB)
        EoutA += currA
        EoutB += currB
    EoutA /= reps
    EoutB /= reps
    return EoutA, EoutB
```

Now let us see visually one run of this experiment

[2]:
```python
Qf =  K= 50
N = 10
random_poly_coeff =  generate_multiplying_coeff(K)
polyRandomTarget = generate_poly(K,random_poly_coeff)
```

```
xvals = np.arange(-1,1,0.005) #Legendre polynomials are defined within [-1, 1]
yvals = polyRandomTarget(xvals)
ds_x, ds_y = generate_data_set(N,random_poly_coeff)
tol = 1.0e-6

h2 = PolynomialFeatures(2)
h10 = PolynomialFeatures(10)
x2 = h2.fit_transform(ds_x.reshape(-1, 1) )
x10 = h10.fit_transform(ds_x.reshape(-1, 1) )

model2 = LinearRegression()
model2.fit(x2, ds_y)
yvals2 = model2.predict(h2.fit_transform(xvals.reshape(-1, 1) ))

model10 = LinearRegression()
model10.fit(x10, ds_y)
yvals10 = model10.predict(h10.fit_transform(xvals.reshape(-1, 1) ))



N = 10
plt.plot(xvals,yvals,label='True Target Function')

plt.plot(ds_x, ds_y, '.', markersize=15, label='Training Data')
plt.plot(xvals, yvals2,  label='H2 prediction')
plt.plot(xvals, yvals10,  label='H10 prediction')
plt.legend()
plt.title("Noisy low-order target: 10th degree polynomial with no noise")

plt.ylim(np.min(yvals)-2, np.max(yvals)+2)
plt.show()
```



Let us now repeat this several times

- New training dataset and model training
- Monte carlo estimation of the Eout

[3]:
```
degA = 2
degB = 10
print(estimateEoutSeveral(K,N,random_poly_coeff, degA, degB))
```

```
(1.0522464931349274, 52563160528.843796)
```