Lec13ex2-v2

November 22, 2022

1 Exercise 2

Create a large regression dataset and demonstrate that by taking the mean of the models obtained with large p and relatively small training sets, we are able to break down the variance and thus significantly improve performance. Why can't this methodology actually be used? How to reduce the variance of the models even with small training sets?

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from numpy.random import default_rng

np.random.seed(42)
```

For the first part of this exercise, we will generate a regression dataset with 100000 points using trigonometry functions. We will store `x_range` and `true_f` as the value of this trigonometry function.

We will then split this dataset into 5000 seperate samples and each sample has 20 data points.

[2]:
```
def f(x):
    return np.sin(2*x) + np.cos(x)
```

[3]:
```
x_range = np.arange(-2,7.00001,0.00001)
true_f = f(x_range)
```

[4]:
```
no_samples = 100000
```

[5]:
```
x = np.random.uniform(low = -2, high = 7, size = no_samples)
y = f(x) + np.random.normal(0, 3, no_samples)
```

[6]:
```
rng = default_rng()
samples = np.split(rng.choice(len(x), size=len(x), replace=False), 5000)
```

[7]:
```
print("There are", len(samples), "samples and each samples has",␣
 ↪len(samples[0]), "points.")
```

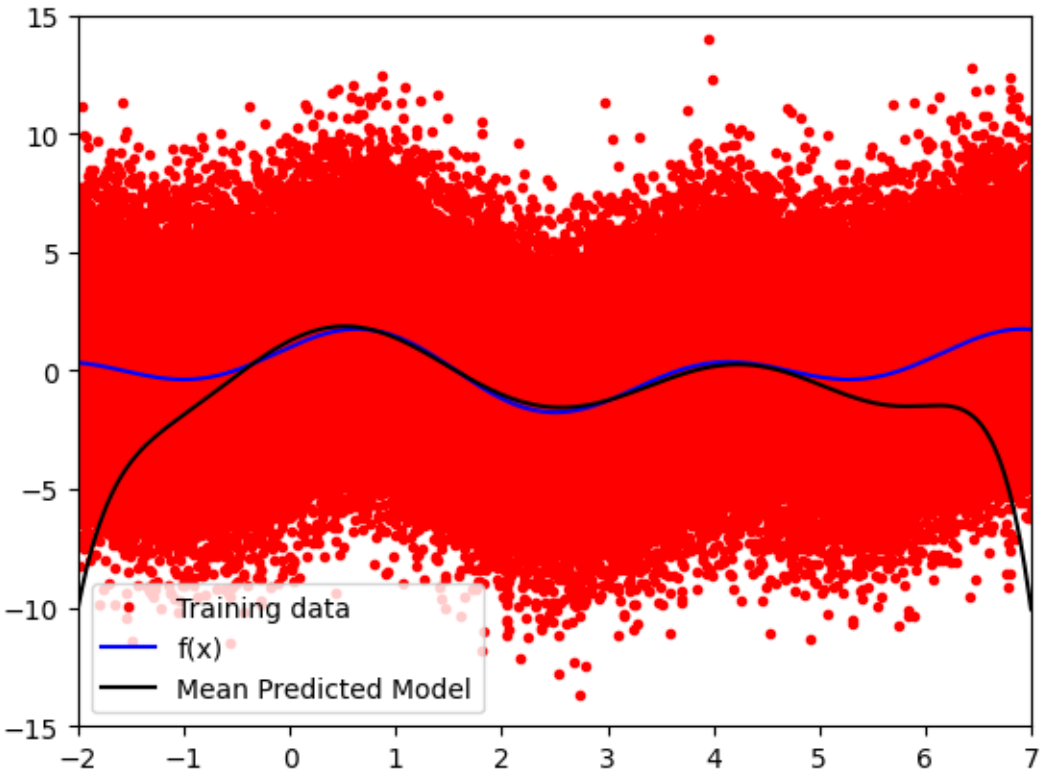
```
There are 5000 samples and each samples has 20 points.
```

For each sample, we will fit a Polynomial Regression of degree `p` (specified below as 8). We will use this predicted model to predict the y values for each point in `x_range`. Finally, we will take the mean of all these predictions and the graph of ths "average" prediction is shown below.

[8]:
```
p = 8
```

[9]:
```
y_predicted = np.zeros((len(samples), len(x_range)))
for i in range(len(samples)):
    poly = PolynomialFeatures(degree = p, include_bias=False)
    poly_features = poly.fit_transform(x[samples][i].reshape(-1, 1))
    poly_reg_model = LinearRegression()
    poly_reg_model.fit(poly_features, y[samples][i])
    poly_features_predicted = poly.fit_transform(x_range.reshape(-1, 1))
    y_predicted[i] = poly_reg_model.predict(poly_features_predicted)
mean_y_predicted = np.mean(y_predicted, axis = 0)
```

[10]:
```
plt.plot(x, y, 'r.', label = 'Training data')
plt.plot(x_range,true_f,'b-', label = 'f(x)')
plt.plot(x_range, mean_y_predicted,'k-', label = 'Mean Predicted Model')
plt.legend()
plt.xlim(-2,7)
plt.ylim(-15,15)
plt.show()
```



It seems that this average model fits our data really well apart from the 2 tails. This can be a sign that this method of model train is poor at extrapolation.

The variance of a prediction model is given by $Var[\hat{f}(\mathbf{x})] = \mathbb{E}[\hat{f}(\mathbf{x}) - \mathbb{E}[\hat{f}(\mathbf{x})]]^2$. Here, we have fitted 5000 models using 20 points each. We can calculate the variance for these individual models as well as the variance for the mean prediction model.

[11]:
```
sample_var = np.zeros(len(y_predicted))
```

[12]:
```
# This kills kernal
#sample_var = np.var(y_predicted, axis=1)

for i in range(len(y_predicted)):
    sample_var[i] = np.var(y_predicted[i])
```

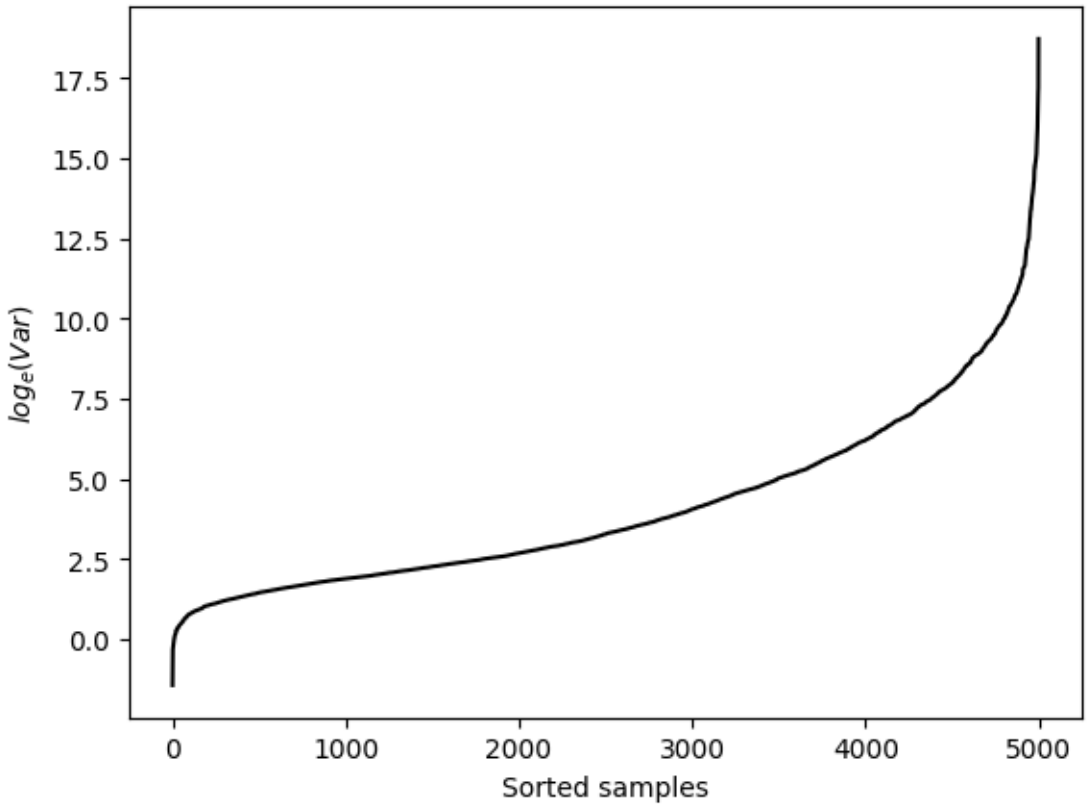
[13]:
```
print("The highest variance we have across the 5000 models is",␣
 ↪round(max(sample_var),3), "and the 20 highest recorded variances are:")
for i in range(20):
    print(round(np.sort(sample_var)[-1-i],3))
```

```
The highest variance we have across the 5000 models is 135680733.798 and the 20
highest recorded variances are:
135680733.798
```

```
29406811.672
25672496.214
17980986.929
15497167.323
12460432.627
8446076.538
7327519.114
6469281.261
5636439.503
5061441.134
4848035.958
3754847.504
3524585.807
3281418.682
3236851.806
3025971.814
2993098.807
2986816.304
2750429.355
```

Graphing a log scale scatter plot of these 5000 variances, sorted by variance. We can see that a majority of models have high variance.

[14]:
```
plt.plot(np.log(np.sort(sample_var)),'k-')
plt.xlabel('Sorted samples')
plt.ylabel('$log_{e}(Var)$')
plt.show()
```



[15]:
```
print("The variance of the mean prediction model is", round(np.
↪var(mean_y_predicted),3))
```

```
The variance of the mean prediction model is 4.441
```

The mean prediction model seems to break down the variance. Combined with the accuracy of this model (relative to the 5000 individual models), this model significantly improve performance and accuracy.

However, this methodology might not be the best as we have observed above. It is very poor at extrapolation.

[16]:
```
print("Looking at the Mean Square Error of our mean prediction model, we have a␣
↪very poor result of", round(np.mean((mean_y_predicted-true_f)**2),3))
```

```
Looking at the Mean Square Error of our mean prediction model, we have a very
poor result of 6.136
```

If we narrow down the range of x, the MSE improves dramatically, which is another sign of extrapolation!

[17]:
```
np.mean((mean_y_predicted[(x_range > -1) & (x_range <= 6)]-true_f[(x_range >␣
↪-1) & (x_range <= 6)])**2)
```

[17]:
```
0.2676955949732778
```

[18]:
```
np.mean((mean_y_predicted[(x_range > 0) & (x_range <= 5)]-true_f[(x_range > 0)␣
↪& (x_range <= 5)])**2)
```

[18]:
```
0.023455658539645607
```

Finally, to reduce the variance of the models even with small training sets. One can consider using Hold-out or Cross-Validation procedures when training and validating the models.