TU Berlin
Machine Learning 1

05 Homework

Model Selection

Jiayun, Fabi, Vincent, Lukas, Paul
December 5, 2020

Exercise 1: Bias and Variance of Mean Estimators

Bias:

$$Bias\left(\hat{\mu}\right) = E\left[\hat{\mu} - \mu\right] = E\left[\alpha \cdot \tfrac{1}{N}\sum_{i=1}^{N} x_i - \mu\right] = \alpha E\left[\tfrac{1}{N}\sum_{u=1}^{N} x_i\right] - E\left[\mu\right] = \tfrac{\alpha}{N}\sum_{i=1}^{N}\underbrace{E\left[x_i\right]}_{=\mu} - \mu =$$

$$\tfrac{\alpha}{N} \cdot N \cdot \mu - \mu = \alpha \cdot \mu - \mu = \mu\left(\alpha - 1\right)$$

Variance:

$$Var\left(\hat{\mu}\right) = Var\left(\tfrac{\alpha}{N}\sum_{i=1}^{N} x_i\right) = \tfrac{\alpha^2}{N^2} Var\left(\sum_{i=1}^{N} x_i\right) = \tfrac{\alpha^2}{N^2}\sum_{i=1}^{N}\underbrace{Var\left(x_i\right)}_{=\sigma^2} = \tfrac{\alpha^2}{N^2} N\sigma^2 = \tfrac{\alpha^2}{N}\sigma^2$$

Mean Squared Error:

$$Error\left(\hat{\mu}\right) = Bias\left(\hat{\mu}\right)^2 + Var\left(\hat{\mu}\right) = \mu^2\left(\alpha - 1\right)^2 + \tfrac{1}{N}\alpha^2\sigma^2$$

Exercise 2

a) First we reparameterize $R_i = \exp(Z_i)$ to ensure that$R$ is positive. To find the distribution that minimizes its expected KL divergence from the the class distribution estimator we have to set the derivative to 0. Observe that the derivative of Z_j is

$$\frac{d}{dZ_j}\mathrm{E}[D_{KL}(R||\hat{P})] = \mathrm{E}[\frac{d}{dZ_j}D_{KL}(R||\hat{P})] = \mathrm{E}[\frac{d}{dZ_j}\sum_{i=1}^{C}\exp(Z_i)\log(\exp(Z_i)/\hat{P}_i)]$$

$$= \mathrm{E}[\frac{d}{dZ_j}\sum_{i=1}^{C}\exp(Z_i)(Z_i - \log(\hat{P}_i))] = \mathrm{E}[\exp(Z_j)(Z_j - \log(\hat{P}_j)) + \exp(Z_j)]$$

$$= \exp(Z_j)(Z_j - \mathrm{E}[\log(\hat{P}_j)] + 1)$$

for all $1 \leq j \leq C$. Now setting this equal to zero we get

$$Z_j = \mathrm{E}[\log(\hat{P}_j)] + 1 \implies R_j = \exp(\mathrm{E}[\log(\hat{P}_j)] + 1)$$

To get the actual probability distribution we have to normalize our solution. Thus

$$R_i = \frac{\exp(\mathrm{E}[\log(\hat{P}_i)]) \cdot e}{\sum_{j=1}^{C}\exp(\mathrm{E}[\log(\hat{P}_j)]) \cdot e} = \frac{\exp(\mathrm{E}[\log(\hat{P}_i)])}{\sum_{j=1}^{C}\exp(\mathrm{E}[\log(\hat{P}_j)])}$$

for $1 \leq i \leq C$.

b) First we consider the term that is given in the hint, which will be useful in our calculations

$$\mathrm{E}[\log(R_i) - \log(\hat{P}_i)] = \mathrm{E}[\mathrm{E}[log(\hat{P}_i)] - \log(\sum_{j=1}^{C}\exp(\mathrm{E}[\log(\hat{P}_j)])) - \log(\hat{P}_i)]$$

$$= -\log(\sum_{j=1}^{C}\exp(\mathrm{E}[\log(\hat{P}_j)])) =: \mathrm{E}[\log(R) - \log(\hat{P})]$$

So this does indeed not depend on i. One can show with the same calculations that this also works for

$$\mathrm{E}[\log(P_i) - \log(R_i)] =: \mathrm{E}[\log(P) - \log(R)]$$

We can write by definition of the expectation with respect to the probability distribution defined over the simplex $[1, ..., C]$ as

$$\mathrm{E}[\log(P) - \log(R)] = \sum_{i=1}^{C} P_i(\log(P_i) - \log(R_i))$$

and

$$\mathrm{E}[\log(R) - \log(\hat{P})] = \sum_{i=1}^{C} R_i(\log(R_i) - \log(\hat{P}_i))$$

Now we can show the bias-variance decomposition

$$\begin{aligned}
&\mathrm{Bias}(\hat{P}) + \mathrm{Var}(\hat{P}) = D_{KL}(P||R) + \mathrm{E}[D_{KL}(R||\hat{P})] \\
&\quad= \sum_{i=1}^{C} P_i \log(P_i/R_i) + \mathrm{E}[\sum_{i=1}^{C} R_i \log(R_i/\hat{P}_i)] \\
&\quad= \sum_{i=1}^{C} P_i(\log(P_i) - \log(R_i)) + \mathrm{E}[\sum_{i=1}^{C} R_i(\log(R_i) - \log(\hat{P}_i))] \\
&\quad= \mathrm{E}[\log(P) - \log(R)] + \mathrm{E}[\mathrm{E}[\log(R) - \log(\hat{P})]] \\
&\quad= \mathrm{E}[\mathrm{E}[\log(P) - \log(R)] + \mathrm{E}[\log(R) - \log(\hat{P})]] \\
&\quad= \mathrm{E}[\mathrm{E}[\log(P) - \log(\hat{P})]] \\
&\quad= \mathrm{E}[\sum_{i=1}^{C} P_i(\log(P_i) - \log(\hat{P}_i))] \\
&\quad= \mathrm{E}[D_{KL}(P||\hat{P})] = \mathrm{Error}(\hat{P})
\end{aligned}$$

sheet05

December 6, 2020

0.1 Part 1: The James-Stein Estimator (20 P)

Let $x_1, \ldots, x_N \in \mathbb{R}^d$ be independent draws from a multivariate Gaussian distribution with mean vector μ and covariance matrix $\Sigma = \sigma^2 I$. It can be shown that the maximum-likelihood estimator of the mean parameter μ is the empirical mean given by:

$$\hat{\mu}_{\mathrm{ML}} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Maximum-likelihood appears to be a strong estimator. However, it was demonstrated that the following estimator

$$\hat{\mu}_{JS} = \Big(1 - \frac{(d-2) \cdot \frac{\sigma^2}{N}}{\|\hat{\mu}_{\mathrm{ML}}\|^2}\Big) \hat{\mu}_{\mathrm{ML}}$$

(a shrinked version of the maximum-likelihood estimator towards the origin) has actually a smaller distance from the true mean when $d \geq 3$. This however assumes knowledge of the variance of the distribution for which the mean is estimated. This estimator is called the James-Stein estimator. While the proof is a bit involved, this fact can be easily demonstrated empirically through simulation. This is the object of this exercise.

The code below draws ten 50-dimensional points from a normal distribution with mean vector $\mu = (1, \ldots, 1)$ and covariance $\Sigma = I$.

[123]:
```python
import numpy

def getdata(seed):

    n = 10                # data points
    d = 50                # dimensionality of data
    m = numpy.ones([d])   # true mean
    s = 1.0               # true standard deviation

    rstate = numpy.random.mtrand.RandomState(seed)
    X = rstate.normal(0,1,[n,d])*s+m

    return X,m,s
```

The following function computes the maximum likelihood estimator from a sample of the data assumed to be generated by a Gaussian distribution:

[124]:
```
def ML(X):
    return X.mean(axis=0)
```

0.1.1 Implementing the James-Stein Estimator (10 P)

- **Based on the ML estimator function, write a function that receives as input the data $(X_i)_{i=1}^n$ and the (known) variance σ^2 of the generating distribution, and computes the James-Stein estimator**

[125]:
```
def JS(X,s):
    # REPLACE BY YOUR CODE
    Uml = ML(X)
    (N,d) = X.shape
    m_JS = (1 - (d-2)*s**2/(numpy.linalg.norm(Uml,2)**2*N))*Uml
    ###
    return m_JS
```

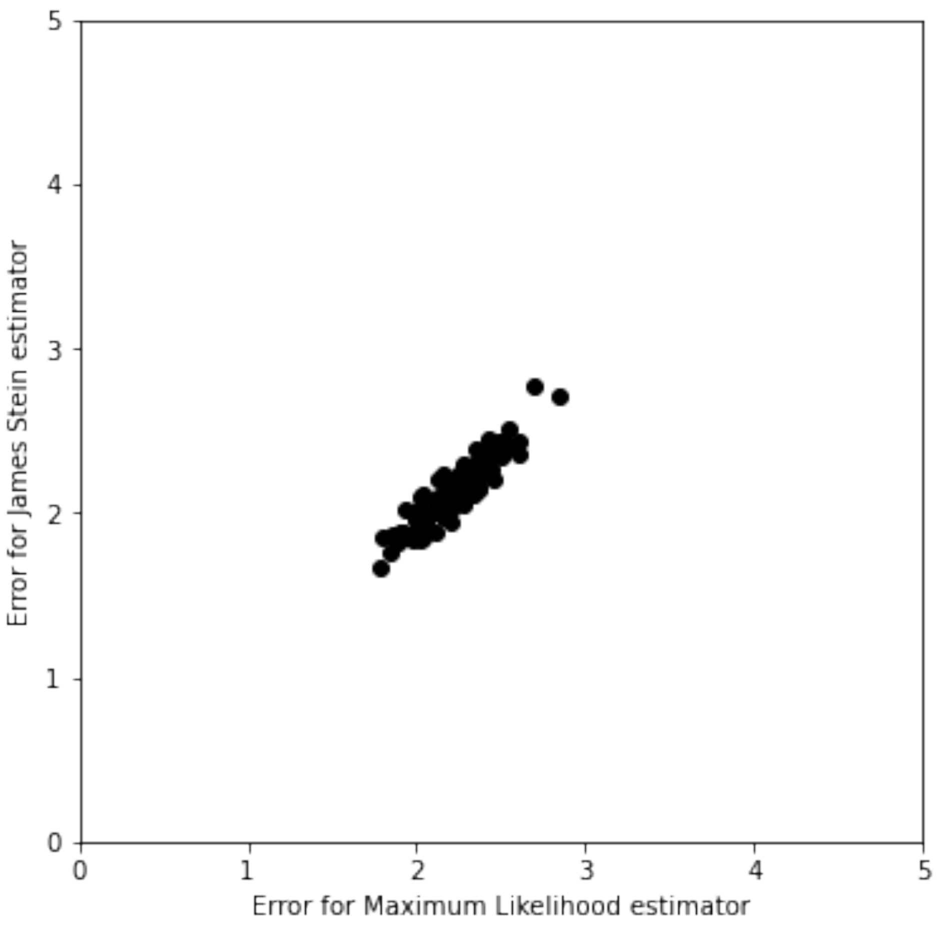
0.1.2 Comparing the ML and James-Stein Estimators (10 P)

We would like to compute the error of the maximum likelihood estimator and the James-Stein estimator for 100 different samples (where each sample consists of 10 draws generated by the function `getdata` with a different random seed). Here, for reproducibility, we use seeds from 0 to 99. The error should be measured as the Euclidean distance between the true mean vector and the estimated mean vector.

- **Compute the maximum-likelihood and James-Stein estimations.**
- **Measure the error of these estimations.**
- **Build a scatter plot comparing these errors for different samples.**

[126]:
```
%matplotlib inline
### REPLACE BY YOUR CODE
from matplotlib import pyplot as plt
Error_ml = []
Error_js = []
for seed in range(100):
    (X,m,s) = getdata(seed)
    m_ML = ML(X)
    m_JS = JS(X,s)

    Error_ml.append(numpy.linalg.norm((m_ML-1), 2))
    Error_js.append(numpy.linalg.norm((m_JS-1), 2))

Error_js = numpy.array(Error_js)
Error_ml = numpy.array(Error_ml)

plt.figure(figsize=(6,6))
plt.scatter(Error_ml, Error_js,c = 'k')
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.xlabel('Error for Maximum Likelihood estimator')
plt.ylabel('Error for James Stein estimator')
###
```

```
[126]: Text(0, 0.5, 'Error for James Stein estimator')
```



0.2 Part 2: Bias/Variance Decomposition (30 P)

In this part, we would like to implement a procedure to find the bias and variance of different predictors. We consider one for regression and one for classification. These predictors are available in the module utils.

- **`utils.ParzenRegressor`:** A regression method based on Parzen window. The hyperparameter corresponds to the scale of the Parzen window. A large scale creates a more rigid model. A small scale creates a more flexible one.

- **`utils.ParzenClassifier`:** A classification method based on Parzen window. The hyperparameter corresponds to the scale of the Parzen window. A large scale creates a more rigid model. A small scale creates a more flexible one. Note that instead of returning a single class for a given data point, it outputs a probability distribution over the set of possible classes.

Each class of predictor implements the following three methods:

- **`__init__(self,parameter)`:** Create an instance of the predictor with a certain scale parameter.
- **`fit(self,X,T)`:** Fit the predictor to the data (a set of data points `X` and targets `T`).
- **`predict(self,X)`:** Compute the output values arbitrary inputs `X`.

To compute the bias and variance estimates, we require *multiple samples* from the training set for a single set of observation data. To acomplish this, we utilize the **`Sampler`** class provided. The sampler is initialized with the training data and passed to the method for estimating bias and variance, where its function **`sampler.sample()`** is called repeatedly in order to fit multiple models and create an ensemble of prediction for each test data point.

0.2.1 Regression Case (15 P)

For the regression case, Bias, Variance and Error are given by:

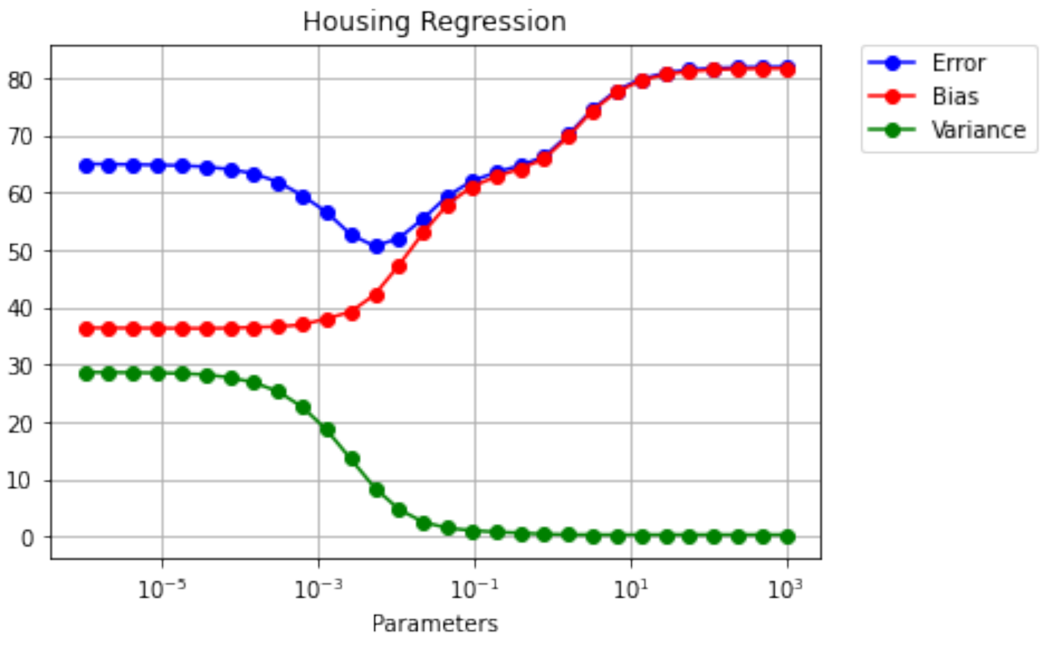
- $\text{Bias}(Y)^2 = (\mathbb{E}_Y[Y - T])^2$
- $\text{Var}(Y) = \mathbb{E}_Y[(Y - \mathbb{E}_Y[Y])^2]$
- $\text{Error}(Y) = \mathbb{E}_Y[(Y - T)^2]$

Task: Implement the KL-based Bias-Variance Decomposition defined above. The function should repeatedly sample training sets from the sampler (as many times as specified by the argument nbsamples), learn the predictor on them, and evaluate the variance on the out-of-sample distribution given by X and T.

[127]:
```python
def biasVarianceRegression(sampler, predictor, X, T, nbsamples):

    # --------------------------------
    # TODO: REPLACE BY YOUR CODE
    # --------------------------------
    Y_sample = []
    for _ in range(nbsamples):
        sample = sampler.sample()
        predictor.fit(sample[0],sample[1])
        Y_sample.append(predictor.predict(X))
    Y_sample = numpy.array(Y_sample)
    Y = numpy.mean(Y_sample,axis=0)

    variance = numpy.mean(numpy.mean((Y_sample - Y)**2,axis=0))
    error = numpy.mean(numpy.mean((Y_sample - T)**2,axis=0))
    bias = error - variance
    # --------------------------------

    return bias,variance
```

Your implementation can be tested with the following code:

```
[128]: import utils,numpy
%matplotlib inline
utils.plotBVE(utils.Housing,numpy.logspace(-6,3,num=30),utils.
 ↪ParzenRegressor,biasVarianceRegression,'Housing Regression')
```



0.2.2 Classification Case (15 P)

We consider here the Kullback-Leibler divergence as a measure of classification error, as derived in the exercise, the Bias, Variance decomposition for such error is:

- $\text{Bias}(Y) = D_{\text{KL}}(T||R)$
- $\text{Var}(Y) = \mathbb{E}_Y[D_{\text{KL}}(R||Y)]$
- $\text{Error}(Y) = \mathbb{E}_Y[D_{\text{KL}}(T||Y)]$

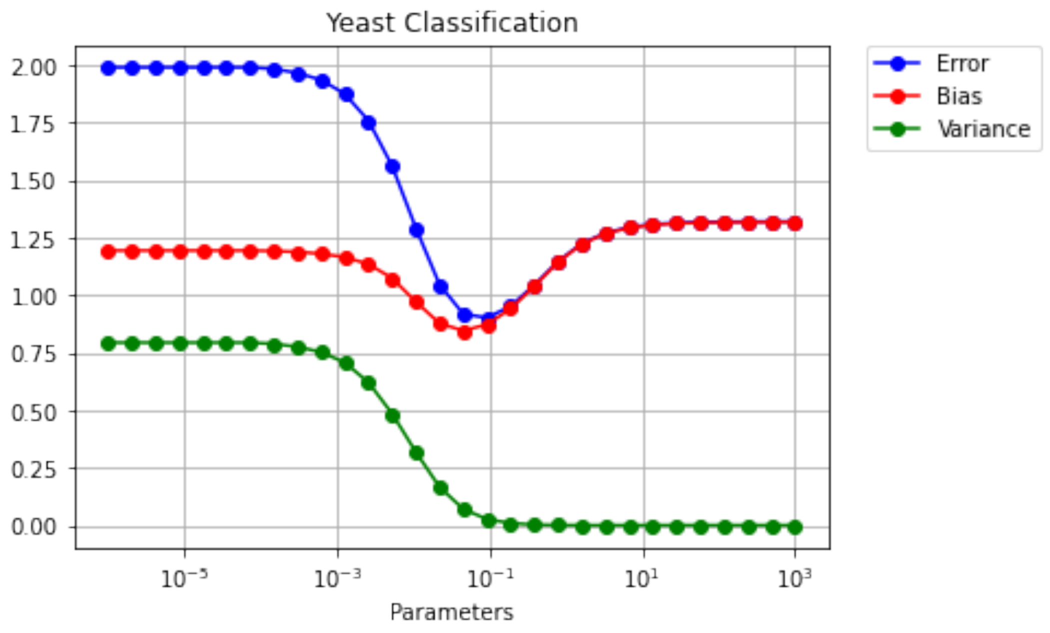
where R is the distribution that minimizes its expected KL divergence from the estimator of probability distribution Y (see the theoretical exercise for how it is computed exactly), and where T is the target class distribution.

Task: Implement the KL-based Bias-Variance Decomposition defined above. The function should repeatedly sample training sets from the sampler (as many times as specified by the argument nbsamples), learn the predictor on them, and evaluate the variance on the out-of-sample distribution given by X and T.

[129]:
```
def KL(P1,P2):
    return numpy.sum(P1 * numpy.log(P1 / P2), axis=2)

def biasVarianceClassification(sampler, predictor, X, T, nbsamples=25):

    # --------------------------------
    # TODO: REPLACE BY YOUR CODE
    # --------------------------------
    Y_sample = []
    for _ in range(nbsamples):
        sample = sampler.sample()
        predictor.fit(sample[0],sample[1])
        Y_sample.append(predictor.predict(X))
    Y_sample = numpy.array(Y_sample) #(20, 742, 7)

    numerator = numpy.exp(numpy.mean(numpy.log(Y_sample),axis=0)) #(742,7)
    denominator = numpy.sum(numerator,axis=1)[:,numpy.newaxis] #(742,1)
    R = numerator / denominator #(742,7)

    Error = numpy.mean(KL(T,Y_sample)) #scalar
    variance = numpy.mean(KL(R,Y_sample)) #inner KL is (20,742),variance is␣
↪scalar
    bias = Error - variance
    # --------------------------------

    return bias,variance
```

Your implementation can be tested with the following code:

[130]:
```
import utils,numpy
%matplotlib inline
utils.plotBVE(utils.Yeast,numpy.logspace(-6,3,num=30),utils.
↪ParzenClassifier,biasVarianceClassification,'Yeast Classification')
```


Yeast Classification
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
10^{-5}
10^{-3}
10^{-1}
10^{1}
10^{3}
Parameters
Error
Bias
Variance

```
[ ]:
```