Exercises for the course
Machine Learning 1
Winter semester 2021/22

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 5

Exercise 1: Bias and Variance of Mean Estimators (20 P)

Assume we have an estimator $\hat{\theta}$ for a parameter θ. The bias of the estimator $\hat{\theta}$ is the difference between the true value for the estimator, and its expected value

$$\text{Bias}(\hat{\theta}) = \text{E}\big[\hat{\theta} - \theta\big].$$

If $\text{Bias}(\hat{\theta}) = 0$, then $\hat{\theta}$ is called unbiased. The variance of the estimator $\hat{\theta}$ is the expected square deviation from its expected value

$$\text{Var}(\hat{\theta}) = \text{E}\big[(\hat{\theta} - \text{E}[\hat{\theta}])^2\big].$$

The mean squared error of the estimator $\hat{\theta}$ is

$$\text{Error}(\hat{\theta}) = \text{E}\big[(\hat{\theta} - \theta)^2\big] = \text{Bias}(\hat{\theta})^2 + \text{Var}(\hat{\theta}).$$

Let $X_1, \ldots, X_N$ be a sample of i.i.d random variables. Assume that X_i has mean μ and variance σ^2. *Calculate* the bias, variance and mean squared error of the mean estimator:

$$\hat{\mu} = \alpha \cdot \frac{1}{N} \sum_{i=1}^{N} X_i$$

where α is a parameter between 0 and 1.

Exercise 2: Bias-Variance Decomposition for Classification (30 P)

The bias-variance decomposition usually applies to regression data. In this exercise, we would like to obtain similar decomposition for classification, in particular, when the prediction is given as a probability distribution over C classes. Let $P = [P_1, \ldots, P_C]$ be the ground truth class distribution associated to a particular input pattern. Assume a random estimator of class probabilities $\hat{P} = [\hat{P}_1, \ldots, \hat{P}_C]$ for the same input pattern. The error function is given by the expected KL-divergence between the ground truth and the estimated probability distribution:

$$\text{Error} = \text{E}\big[D_{\text{KL}}(P||\hat{P})\big] = \text{E}\big[\textstyle\sum_{i=1}^{C} P_i \log(P_i/\hat{P}_i)\big].$$

First, we would like to determine the mean of of the class distribution estimator $\hat{P}$. We define the mean as the distribution that minimizes its expected KL divergence from the the class distribution estimator, that is, the distribution R that optimizes

$$\min_R \ \text{E}\big[D_{\text{KL}}(R||\hat{P})\big].$$

(a) *Show* that the solution to the optimization problem above is given by

$$R = [R_1, \ldots, R_C] \quad \text{where} \quad R_i = \frac{\exp \text{E}\big[\log \hat{P}_i\big]}{\sum_j \exp \text{E}\big[\log \hat{P}_j\big]} \qquad \forall\ 1 \le i \le C.$$

(Hint: To implement the positivity constraint on R, you can reparameterize its components as $R_i = \exp(Z_i)$, and minimize the objective w.r.t. Z.)

(b) *Prove* the bias-variance decomposition

$$\text{Error}(\hat{P}) = \text{Bias}(\hat{P}) + \text{Var}(\hat{P})$$

where the error, bias and variance are given by

$$\text{Error}(\hat{P}) = \text{E}\big[D_{\text{KL}}(P||\hat{P})\big], \qquad \text{Bias}(\hat{P}) = D_{\text{KL}}(P||R), \qquad \text{Var}(\hat{P}) = \text{E}\big[D_{\text{KL}}(R||\hat{P})\big].$$

(Hint: as a first step, it can be useful to show that $\text{E}[\log R_i - \log \hat{P}_i]$ does not depend on the index i.)

Exercise 3: Programming (50 P)

Download the programming files on ISIS and follow the instructions.

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

Let $x_1, \ldots, x_N \in \mathrm{R}^d$ 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

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

(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 $\Sigma = I$.

In [1]:

```
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:

In [2]:

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

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**

In [3]:

```
def JS(X,s):
    # REPLACE BY YOUR CODE
    import solutions
    m_JS = solutions.JS(X,s)
    ###
    return m_JS
```

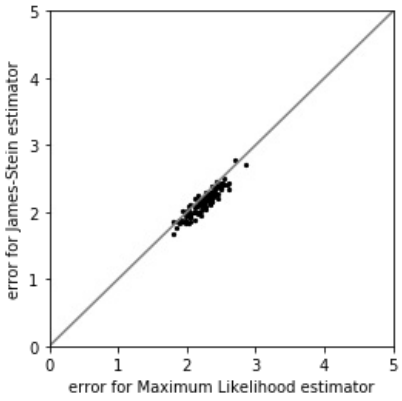
Comparing the ML and James-Stein Estimators (10 P)

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.**

In [4]:

```
%matplotlib inline
### REPLACE BY YOUR CODE
import solutions
solutions.compare_ML_JS()
###
```



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.

Regression Case (15 P)

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

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

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.

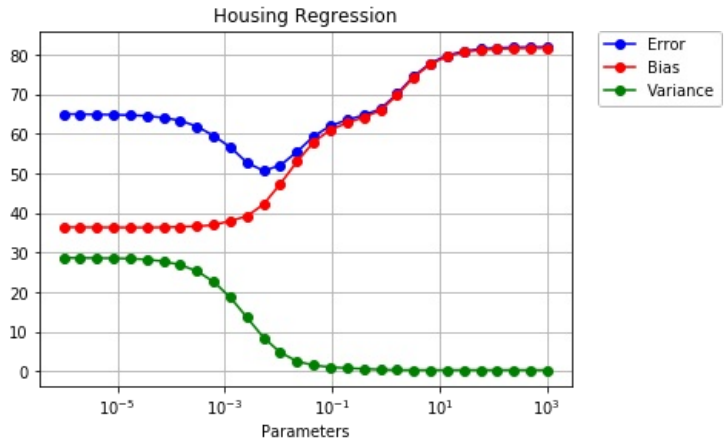
In [5]:

```
def biasVarianceRegression(sampler, predictor, X, T, nbsamples):

    # ---------------------------------
    # TODO: REPLACE BY YOUR CODE
    # ---------------------------------
    import solutions
    bias,variance = solutions.biasVarianceRegression(sampler, predictor, X, T, nbsamples=nbsamples)
    # ---------------------------------

    return bias,variance
```

Your implementation can be tested with the following code:

In [6]:

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



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:

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

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.

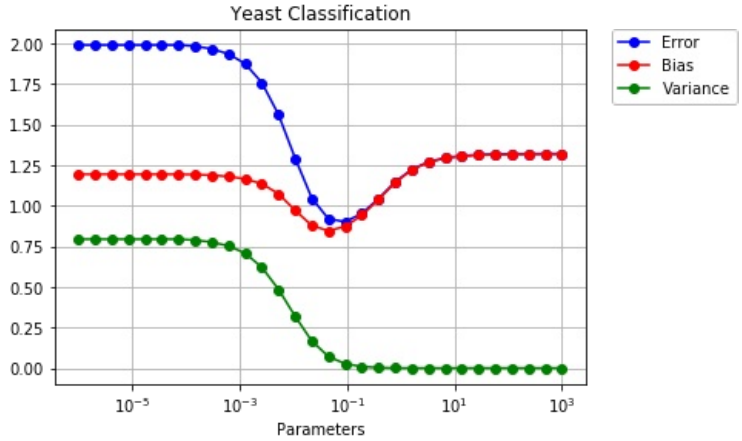
In [7]:

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

    # ---------------------------------
    # TODO: REPLACE BY YOUR CODE
    # ---------------------------------
    import solutions
    bias,variance = solutions.biasVarianceClassification(sampler, predictor, X, T, nbsamples=nbsamples)
    # ---------------------------------

    return bias,variance
```

Your implementation can be tested with the following code:

In [8]:

```
import utils,numpy
%matplotlib inline
utils.plotBVE(utils.Yeast,numpy.logspace(-6,3,num=30),utils.ParzenClassifier,biasVarianceClassification,'Yeast Cl
assification')
```



ML1 Sheet 5

1

a) The bias is defined as $\text{Bias}(\hat{\mu}) = E[\hat{\mu} - \mu]$

1) $\hat{\mu} = \frac{\alpha}{N} \sum_{i=1}^{N} x_i$

2) $E[\frac{\alpha}{N}] = \frac{\alpha}{N}$ and $E[\mu] = \mu$

3) $\mu = \frac{1}{N} \sum_{i=1}^{N} E[X_i]$

$\overset{1)}{=} E[\frac{\alpha}{N} \sum_{i=1}^{N} x_i - \mu]$

$\overset{2)}{=} \frac{\alpha}{N} \sum_{i=1}^{N} E[X_i] - \mu$

$\overset{3)}{=} \alpha\mu - \mu = (\alpha - 1)\mu$

b) $\text{Var}(\hat{\mu}) = \text{Var}(\frac{\alpha}{N} \prod_{i=1}^{N} x_i) = (\frac{\alpha}{N})^2 \prod_{i=1}^{N} \text{Var}(X_i)$

$= \frac{\alpha^2}{N^2} \prod_{i=1}^{N} \sigma^2 = \frac{\alpha^2}{N^2} \sigma^2$

c) $\text{Error}(\hat{\mu}) = (\text{Bias}(\hat{\mu}))^2 + \text{Var}(\hat{\mu})$

$= \mu^2(\alpha - 1)^2 + (\frac{\alpha}{N} \sigma)^2$

2

a) We need to solve $\min_R E[D_{KL}(R\|\hat{P})]$.

$$\min_R E[D_{KL}(R\|\hat{P})] = \min_R E\left[\sum_{i=1}^{c} R_i \log(R_i/\hat{P}_i)\right]$$

$$= \min_R E\left[\sum_{i=1}^{c} R_i \log R_i - R_i \log \hat{P}_i\right]$$

$$= \min_R \sum_{i=1}^{c} R_i \log R_i - R_i E[\log \hat{P}_i]$$

$$= \min_R \sum_{i=1}^{c} e^{z_i} z_i - e^{z_i} E[\log \hat{P}_i] \quad \text{using } R_i = e^{z_i}$$

Using the constraint $\sum_{i=1}^{c} e^{z_i} = 1$ we can set the Lagrangian:

$$\mathcal{L}(z,\lambda) = \sum_{i=1}^{c} e^{z_i} z_i - e^{z_i} E[\log \hat{P}_i] + \lambda\left(\sum_{i=1}^{c} e^{z_i} - 1\right)$$

$$\frac{\partial \mathcal{L}}{\partial z} = \sum_{i=1}^{c} e^{z_i} + e^{z_i} z_i - e^{z_i} E[\log \hat{P}_i] + e^{z_i}\lambda$$

$$= \sum_{i=1}^{c} e^{z_i}(1 + z_i - E[\log \hat{P}_i] + \lambda) \stackrel{!}{=} 0$$

In this product $p \cdot q = 0$, $p = e^{z_i}$ can never be zero, hence the other term must become 0:

$$1 + z_i - E[\log \hat{P}_i] + \lambda \stackrel{!}{=} 0 \iff z_i = E[\log \hat{P}_i] - 1 - \lambda$$

$$\iff R_i = e^{E[\log \hat{P}_i] - 1 - \lambda} = \exp(E[\log \hat{P}_i]) / \exp(1+\lambda)$$

This becomes $R_i = \dfrac{\exp E[\log \hat{P}_i]}{\sum_j \exp E[\log \hat{P}_j]}$

using the running constraint $\exp(1+\lambda) = \sum_{i=1}^{c} \exp(E[\log \hat{P}_i])$

b) Using $R_i = \exp(E[\log \hat{P}_i]) / \exp(1+\lambda)$ we write

$$E[\log R_i - \log \hat{P}_i] = E[\log(\exp(E[\log \hat{P}_i])) - \log(\exp(1+\lambda)) - \log \hat{P}_i]$$

$$= E[E[\log \hat{P}_i] - (1+\lambda) - \log \hat{P}_i]$$

$$= E[\log \hat{P}_i] - (1+\lambda) - E[\log \hat{P}_i]$$

$$= -1-\lambda \quad \text{which is independent of the index } i.$$

The error is given by

$$\text{Error}(\hat{P}) = E[D_{KL}(P \| \hat{P})] = E\left[\sum_{i=1}^{c} P_i \log(P_i / \hat{P}_i)\right]$$

$$= E\left[\sum_{i=1}^{c} P_i \log P_i - P_i \log \hat{P}_i\right] = \cancel{\text{illegible}}$$

$$= \underbrace{E\left[\sum_{i=1}^{c} P_i \log P_i - P_i \log R_i\right]}_{\text{Bias}(\hat{P})} + P_i \log R_i - P_i \log \hat{P}_i]$$

$$= \text{Bias}(\hat{P}) + E\left[\sum_{i=1}^{c} P_i \log R_i - P_i \log \hat{P}_i\right]$$

$$= \text{Bias}(\hat{P}) + \sum_{i=1}^{c} P_i \, E[\log R_i - \log \hat{P}_i]$$

Since $\sum_{i=1}^{c} R_i = \sum_{i=1}^{c} P_i = 1$

$$= \text{Bias}(\hat{P}) + \sum_{i=1}^{c} R_i \, E[\log R_i - \log \hat{P}_i]$$

$$= \text{Bias}(\hat{P}) + E\left[\sum_{i=1}^{c} R_i \log R_i - R_i \log P_i\right]$$

$$= \text{Bias}(\hat{P}) + \text{Var}(\hat{P})$$