

Machine Learning 1 WS18/19

Submission for Exercise Sheet 1

Michael Hoppe 362514

Wai Tang Victor Chan 406094

Jonas Piotrowski 399334

Aki Saksala 399293

Sourabh Raj 397371

Exercise Sheet 7

$$\begin{aligned} 1a) \quad E(\hat{\mu}) &= E\left(\frac{1}{N} \sum_{i=1}^N X_i\right) \\ &= \frac{1}{N} \cdot (N\mu) \\ &= \mu \end{aligned}$$

$$\text{Bias}(\hat{\mu}) = \mu - \mu = 0$$

$$\text{Var}(\hat{\mu}) = \text{Var}\left(\frac{1}{N} \sum_{i=1}^N X_i\right)$$

$$\begin{aligned} &\stackrel{(X_i \text{ are iid})}{=} \frac{1}{N^2} \sum_{i=1}^N \text{Var}(X_i) \end{aligned}$$

$$= \frac{N\sigma^2}{N^2}$$

$$= \frac{\sigma^2}{N}$$

$$\text{Error}(\hat{\mu}) = 0^2 + \frac{\sigma^2}{N} = \frac{\sigma^2}{N}$$

$$1b) \quad E(\hat{\mu}) = 0$$

$$\text{Bias}(\hat{\mu}) = 0 - \mu = -\mu$$

$$\text{Var}(\hat{\mu}) = 0$$

$$\text{Error}(\hat{\mu}) = (-\mu)^2 + 0 = \mu^2$$

$$2) \text{Error}(\hat{f}(x)) = \text{Var}(\hat{f}(x))$$

$$= \mathbb{E}((\hat{f}(x) - f(x))^2) - \mathbb{E}((\hat{f}(x) - \mathbb{E}(\hat{f}(x)))^2)$$

$$= \mathbb{E}((\hat{f}(x))^2) - 2f(x)\mathbb{E}(\hat{f}(x)) + (f(x))^2 \\ - \mathbb{E}((\hat{f}(x))^2) + \underbrace{2\mathbb{E}(\hat{f}(x))\mathbb{E}(\hat{f}(x))}_{= (\mathbb{E}(\hat{f}(x)))^2} - (\mathbb{E}(\hat{f}(x)))^2$$

$$= (\mathbb{E}(\hat{f}(x)))^2 - 2f(x)\mathbb{E}(\hat{f}(x)) + (f(x))^2$$

$$= (\mathbb{E}(\hat{f}(x) - f(x)))^2$$

$$= (\mathbb{E}(\hat{f}(x) - f(x)))^2$$

$$= (\text{Bias}(\hat{f}(x)))^2$$

$$\Rightarrow \text{Error}(\hat{f}(x)) = (\text{Bias}(\hat{f}(x)))^2 + \text{Var}(\hat{f}(x))$$

$$3a) \quad E(D_{KL}(R \parallel \hat{P})) = -E\left(\sum_{i=1}^C R_i \log\left(\frac{\hat{P}_i}{R_i}\right)\right) \\ = \sum_{i=1}^C (R_i \log R_i - R_i E(\log \hat{P}_i))$$

Using Lagrange multiplier, let

$$L(R_1, \dots, R_C, \lambda) = \sum_{i=1}^C (R_i \log R_i - R_i E(\log \hat{P}_i)) - \lambda \left(\sum_{i=1}^C R_i - 1 \right)$$

$$\begin{cases} \frac{\partial L}{\partial R_i} = \log R_i + 1 - E(\log \hat{P}_i) - \lambda = 0 \quad \forall i=1, \dots, C & \text{--- (1)} \\ \frac{\partial L}{\partial \lambda} = \sum_{i=1}^C R_i - 1 = 0 & \text{--- (2)} \end{cases}$$

From (1): $R_i = e^{\lambda-1} e^{E(\log \hat{P}_i)}$ --- (3)

Using (2) $\sum_{j=1}^C R_j = e^{\lambda-1} \sum_{j=1}^C e^{E(\log \hat{P}_j)} = 1$

$$e^{\lambda-1} = \frac{1}{\sum_{j=1}^C e^{E(\log \hat{P}_j)}}$$

Putting this back into (3),

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

$$3b) \text{ Error}(\hat{P}) - \text{Var}(\hat{P})$$

$$= E(D_{KL}(P \parallel \hat{P})) - E(D_{KL}(R \parallel \hat{P}))$$

$$= \sum_{i=1}^c (P_i \log P_i - P_i E(\log \hat{P}_i)) - \sum_{i=1}^c \left(\underbrace{R_i \log R_i}_{= E(\log \hat{P}_i) - \log(\sum_{j=1}^c \exp E(\log \hat{P}_j))} - R_i E(\log \hat{P}_i) \right)$$

$$= \sum_{i=1}^c (P_i \log P_i - P_i E(\log \hat{P}_i)) + \log(\sum_{j=1}^c \exp E(\log \hat{P}_j)) \underbrace{\sum_{i=1}^c R_i}_{=1 = \sum_{i=1}^c P_i}$$

$$= \sum_{i=1}^c (P_i \log P_i - P_i \log(\exp E(\log \hat{P}_i))) + \sum_{i=1}^c P_i \log(\sum_{j=1}^c \exp E(\log \hat{P}_j))$$

$$= \sum_{i=1}^c \left(P_i \log P_i - P_i \log \left(\frac{\exp E(\log \hat{P}_i)}{\sum_{j=1}^c \exp E(\log \hat{P}_j)} \right) \right)$$

$$= \sum_{i=1}^c (P_i \log P_i - P_i \log R_i)$$

$$= D_{KL}(P \parallel R)$$

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

Programming Part

December 3, 2018

1 Model Selection

In this programming assignment we examine techniques for model selection on classification and regression tasks. In particular, we first explore the effect of model hyperparameters on the bias and variance of the prediction. In the second part of the assignment we utilize the bias-variance decomposition to perform automatic hyperparameter selection. Several classes and methods are provided in the `utils.py` file:

1.0.1 Datasets

- **`utils.Housing()`:** This regression dataset is available at <http://archive.ics.uci.edu/ml/datasets/Housing> and loaded from scikit-learn's inbuilt representation. This data is used for regression. A description of the dataset can be found here <http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names>. This data is in a 506x13 matrix and the labels in a array of length 506.
- **`utils.Yeast()`:** This classification dataset is available at <https://archive.ics.uci.edu/ml/datasets/Yeast> . This data is used for classification. A description of the dataset can be found here <https://archive.ics.uci.edu/ml/machine-learning-databases/yeast/yeast.names>. This data is in a 1484x8 matrix and the labels (class probabilities) are in a 1484x7 matrix where $\text{targets}[i, j] = 1$ if example i is of class j and 0 otherwise. For example, if we have a dataset of 4 examples which belong to following classes: [1, 0, 0, 2] the label matrix would look like this: $T = \begin{bmatrix} [0, 1, 0], \\ [1, 0, 0], \\ [1, 0, 0], \\ [0, 0, 1] \end{bmatrix}$.

1.0.2 Predictors

We provide two simple classes of predictors, one for regression and one for classification:

- **`utils.ParzenRegression`:** 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.ParzenClassification`:** 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 .

1.0.3 Bias Variance Decomposition

As we have seen in the theoretical exercise, there are several possible bias-variance decomposition for different tasks (e.g. classification, or regression).

- `utils.biasVarianceRegression()`: Perform the usual bias-variance decomposition of the mean square error. Reminder: given Y the (random) estimator and T the target, the decomposition is computed as follows:
 - $\text{Bias}(Y)^2 = (\mathbb{E}_Y[Y - T])^2$
 - $\text{Var}(Y) = \mathbb{E}_Y[(Y - \mathbb{E}_Y[Y])^2]$
 - $\text{Error}(Y) = \mathbb{E}_Y[(Y - T)^2]$

1.0.4 Sampler

To compute the bias and variance estimates, we require *multiple samples* from the training set for a single set of observation data. To accomplish 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.

1.1 Part 1: Implementing Bias-Variance Decomposition for Classification (20 P)

Implement a function which computes the bias, variance and error given the true labels of the training data and the predicted values. Bias, Variance and Error for classification are defined as:

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

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. Note that we consider here the Kullback-Leibler divergence as a measure of classification error, which is commonly done in practice in order to have a smooth objective function.

Tasks:

- **Implement the KL-based Bias-Variance Decomposition defined above (10 P)**

To get started, you can take inspiration from the readily implemented function `utils.biasVarianceRegression()`, which does the following:

- Iterate for a certain number of times the following:
 - * Acquire a subsample of the training data by invoking `sampler.sample()`
 - * Using the predictor (which will either be a Parzen Regressor or Parzen Classifier depending on the task), fit the model on the sample and determine the prediction for the observation data (N examples disjoint from the training data). Note that the dimension of the outputs matches the dimension of the targets, so for regression you will get an array of length N and for classification a matrix of shape $N \times \text{\#classes}$ containing the class distributions.
- Having computed a number of different predictions, determine the bias, variance and error comparing the predictions to the true labels. Check that the decomposition is correct (i.e. $\text{bias} + \text{variance} = \text{error}$) using an `assert` statement, and return the bias and variance.

- Once the method is implemented, run Test 1 and Test 2 provided below (10 P)

```
In [157]: import scipy
          from scipy import stats

          def biasVarianceClassification(sampler, predictor, X, T, nbsamples=25):
              Y = numpy.array([predictor.fit(*sampler.sample()).predict(X) for _ in range(nbsamples)])

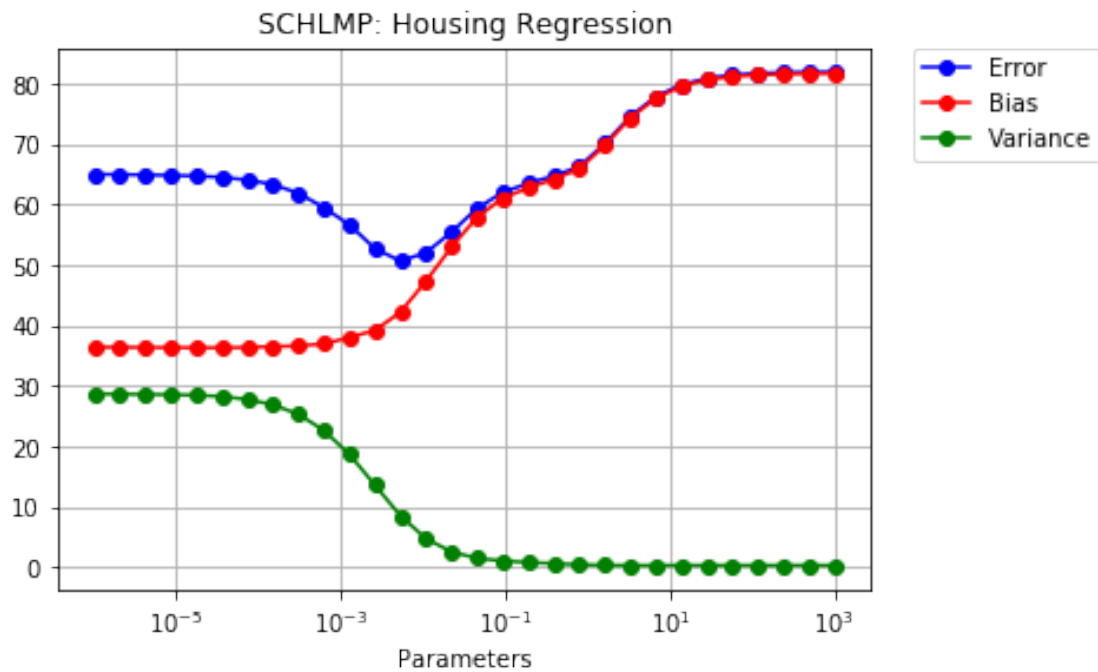
              # calculate R
              R_1 = numpy.exp(numpy.mean(numpy.log(Y), axis=0))
              R_2 = numpy.sum(R_1, axis=0)
              R = R_1 / R_2

              # calculate bias, variance, mean squared error
              bias = scipy.stats.entropy(T, R)
              variance = numpy.mean([scipy.stats.entropy(R, y) for y in Y], axis=0)
              error = numpy.mean([scipy.stats.entropy(T, y) for y in Y], axis=0)

              assert(numpy.all([d < 1e-4 for d in numpy.abs((bias + variance) - error)]))

              return numpy.mean(bias), numpy.mean(variance)

In [158]: ### TEST 1
          import utils,numpy
          %matplotlib inline
          utils.plotBVE(utils.Housing,numpy.logspace(-6,3,num=30),utils.ParzenRegressor,utils.ParzenClassifier)
```

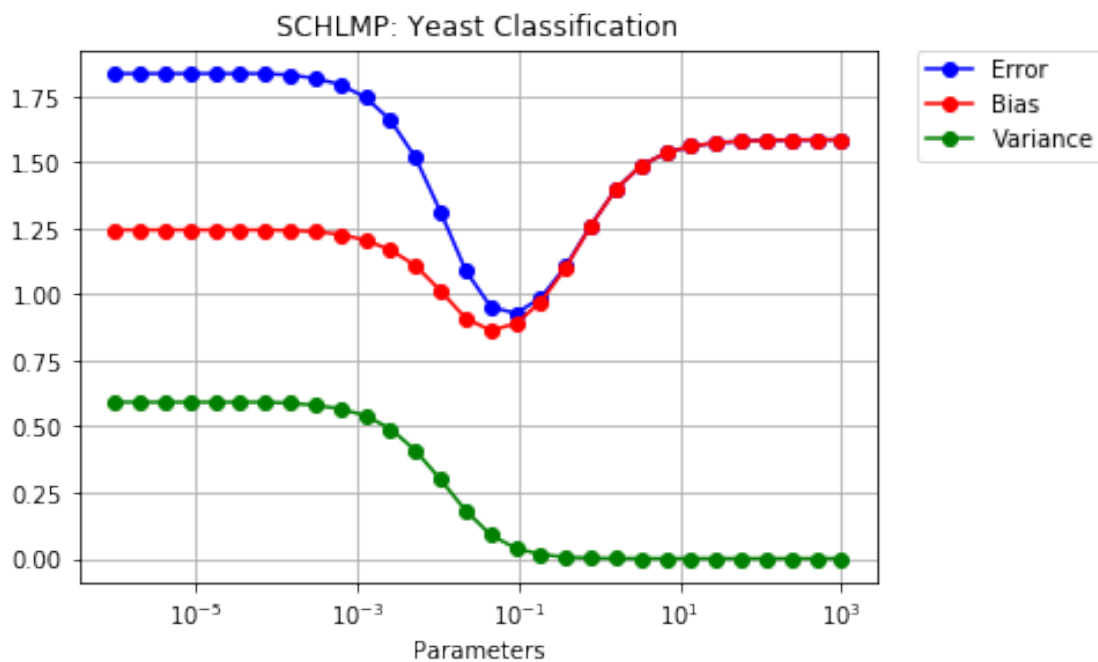



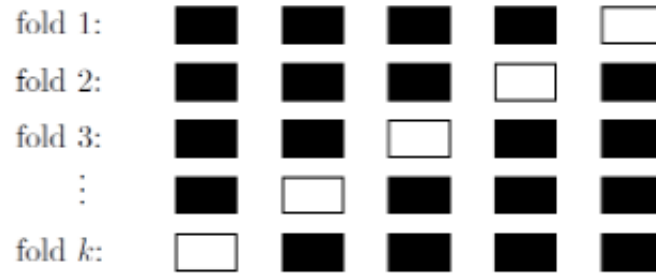
In [159]: `### TEST 2`

`import utils,numpy`

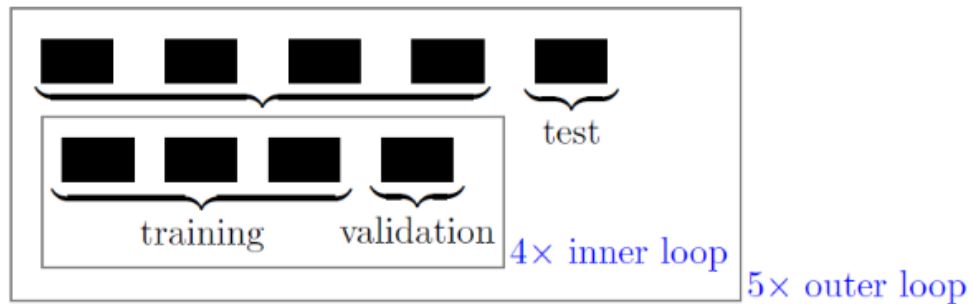
`%matplotlib inline`

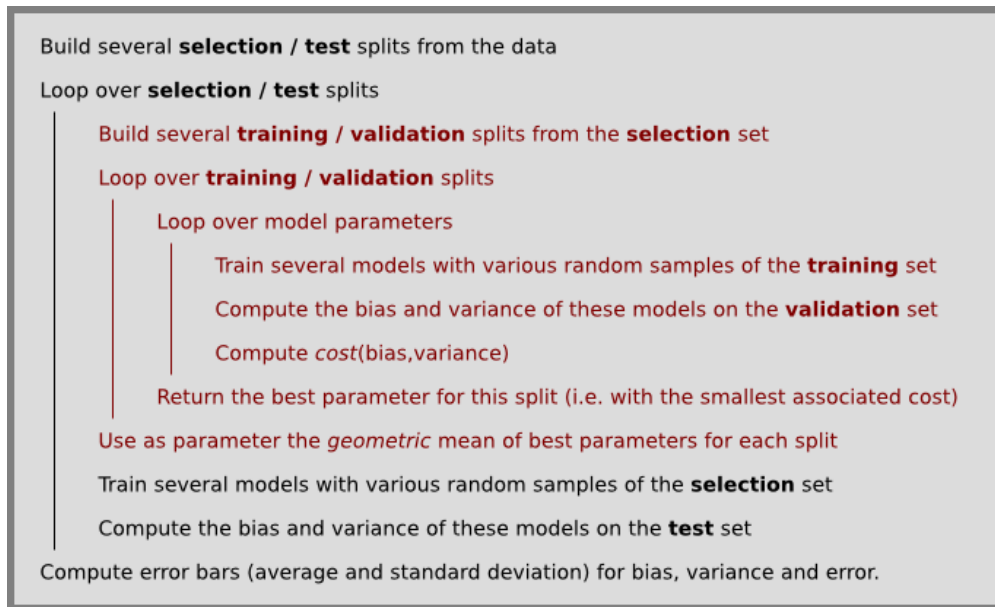
`utils.plotBVE(utils.Yeast,numpy.logspace(-6,3,num=30),utils.ParzenClassifier,biasVar=1)`





Loop





Procedure

- The best parameter for each split is the one that minimizes the costfunction specified as argument.
- The bias and variance estimates are obtained by sampling 10 times from the training distribution.

• Verify your implementation by running Test 3 (10 P)

In [160]: `import numpy,utils`

```

def getbestparameter(Xselect,Tselect,costfunction):

    splits = [( [1,2,3], 0), ([0,2,3], 1), ([0,1,3], 2), ([0,1,2], 3)]

    costs = list()
    for inds_train, ind_val in splits:

        Xtrain = numpy.concatenate([Xselect[ind] for ind in inds_train], axis=0)
        Xval = Xselect[ind_val]

        Ttrain = numpy.concatenate([Tselect[ind] for ind in inds_train], axis=0)
        Tval = Tselect[ind_val]

        splitcosts = list()
        params = numpy.logspace(-5, 5, 15)
        for param in params:
            predictor = utils.ParzenRegressor(param)
            sampler = utils.Sampler(Xtrain, Ttrain)
            bias, variance = utils.biasVarianceRegression(sampler, predictor, Xval, Tval)

```

```

        splitcosts.append(costfunction(bias, variance))

    costs.append(params[numpy.argmin(splitcosts)])

    return scipy.stats.mstats.gmean(costs)

In [161]: import numpy,utils

def evaluateModel(X,T,costfunction):
    # X: partitioned input
    # T: partitioned targets
    # costfunction: the function for evaluate how good/bad a hyperparameter is

    # Create splits
    splits = [ ([1,2,3,4],0) , ([0,2,3,4],1) , ([0,1,3,4],2) , ([0,1,2,4],3) , ([0,1,2,3],4) ]

    testbiases,testvariances,testerrors,bestparameters = [],[],[],[]

    #Loop over selection/test splits
    for inds_select,ind_test in splits:

        Xselect = [X[ind] for ind in inds_select]
        Tselect = [T[ind] for ind in inds_select]

        Xtest = X[ind_test]
        Ttest = T[ind_test]

        bestparam = getbestparameter(Xselect,Tselect,costfunction)

        # Evaluate bias and variance with this best parameter
        predictor = utils.ParzenRegressor(bestparam)
        sampler = utils.Sampler(numpy.concatenate(Xselect,axis=0),numpy.concatenate(Tselect,axis=0))
        bias,variance = utils.biasVarianceRegression(sampler,predictor,Xtest,Ttest,1000)

        testbiases += [bias]
        testvariances += [variance]
        testerrors += [bias+variance]
        bestparameters += [bestparam]

    # Output results of model evaluation
    print('bias:      %8.5f +/- %8.5f'%(numpy.mean(testbiases),numpy.std(testbiases)))
    print('variance:  %8.5f +/- %8.5f'%(numpy.mean(testvariances),numpy.std(testvariances)))
    print('error:      %8.5f +/- %8.5f'%(numpy.mean(testerrors),numpy.std(testerrors)))
    print('parameter: %8.5f +/- %8.5f'%(numpy.mean(bestparameters),numpy.std(bestparameters)))

In [162]: ### TEST 3

import numpy,utils

```

```

costfunctions = [
    ('Parameter Selection Criterion: favor low bias', lambda b,v: 9*b+v),
    ('Parameter Selection Criterion: favor low error', lambda b,v: b+v),
    ('Parameter Selection Criterion: favor low variance', lambda b,v: b+9*v),
]

# Load and partition the data
X,T = utils.Housing()
n = len(X)
X = [X[n*i//5:n*(i+1)//5] for i in range(5)]
T = [T[n*i//5:n*(i+1)//5] for i in range(5)]

print "SCHLMP"
for name,costfunction in costfunctions:
    print('\n\n%s\n'%name)
    evaluateModel(X,T,costfunction)

```

SCHLMP

Parameter Selection Criterion: favor low bias

```

bias:      39.11131 +/-  6.45247
variance:  17.59493 +/-  5.22909
error:     56.70625 +/-  7.29255
parameter:  0.00043 +/-  0.00050

```

Parameter Selection Criterion: favor low error

```

bias:      46.60062 +/-  6.36601
variance:   4.06933 +/-  1.56882
error:     50.66995 +/-  6.55526
parameter:  0.00664 +/-  0.00261

```

Parameter Selection Criterion: favor low variance

```

bias:      63.86182 +/- 10.34832
variance:   0.52321 +/-  0.09478
error:     64.38502 +/- 10.30351
parameter:  0.15402 +/-  0.03189

```

In []: