Exercise Set 1

- Submit the answer via Moodle at latest on 17 November 2021 at 23:59.
- You can answer anonymously or write your name to the answer sheet, at your choice.
- The assignment should be completed by one person, but discussions and joint solving sessions with others are encouraged. Your final solution must however be your own. You are not not allowed to copy ready-made solutions or solutions made by other students. You are allowed to use external sources, web searches included.
- The problems can be discussed in the 15 November exercise session.
- Your answer will be peer-reviewed by you and randomly selected other students.
- The language of the assignments is English.
- The submitted report should be in a single Portable Document Format (pdf) file.
- Answer to the problems in the correct order.
- Read the general instructions and grading criteria in Moodle before starting with the problems.
- Main source material: James et al., Chapters 1-3 and 5. Please feel to use other resources as well. "James et al." refers to: James, Witten, Hastie, Tibshirani, 2021. An Introduction to Statistical Learning with applications in R, 2nd edition. Springer.
- Notice that you can submit your answer to the Moodle well before deadline and revise it until the deadline. Therefore: please submit your answer well in advance, already after you have solved some problems! Due to peer review process we cannot grant extensions to the deadline. Even though the Moodle submission may occasionally remain open for a while after 23:59, the submission system will eventually close. If you try to submit your answers late you will not get any points (including peer-reviwe points) from Exercise Set 1. You have been warned.
- Please double-check that the submitted pdf is formatted properly and, e.g., contains all figures. It seems to be especially difficult to produce properly formatted pdf files with Jupyter Notebooks: remember to check the produced pdf or consider using R Markdown.

Problem 1

[6 points]

Objective: familiarity with tools, basic description of the data set

Planned lecture: 5 November [Ch 1]

This exercise relates to a data set about new particle formation (NPF) of which you will do your term project. Read the data description and download the dataset file `npf_train.csv` from the Moodle term project final report page.

The instructions below are in R, but the examples are given also in Python, you can use either.

Before reading the data into R or Python, it can be viewed in Excel or a text editor.

Task a

Use the `read.csv()` function to read the data into R. Call the loaded data npf. Make sure that you have the directory set to the correct location for the data.

```
npf <- read.csv("npf_train.csv")
```

I use Python with RStudio. Others may want to use Jupyter notebook. However, I want to use the Python installation that came with Anaconda Individual Edition, which means that I do not have to set up virtual

environments etc. with duplicate package installations (even though some prefer that). The following does the trick on my Mac (replace the path with the Anaconda python path in your local installation):

```
Sys.setenv(RETICULATE_PYTHON="/usr/local/anaconda3/bin/python")
library(reticulate)
```

Python:

```
## You often want to import at least these:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
npf = pd.read_csv("npf_train.csv")
```

Task b

Look at the data using the `fix` function. In RStudio, instead of `fix`, you can use the nicer Data Viewer `View` (but if you do not have RStudio then just replace `View` with `fix`).

```
## fix(npf)
View(npf)
```

Python does not have nice built-in viewer, unless you install some packages:

```
npf
```

You should notice that the second column is the date and first column is the id. We don't really want R to treat this as data. However, it may be handy to have the dates. Try the following commands:

```
rownames(npf) <- npf[,"date"]
View(npf)
```

In Python:

```
npf = npf.set_index("date")
## Tell that "class4" is categorical variable. (R does this automatically.)
npf["class4"] = npf["class4"].astype("category")
npf
```

You should see that there is now a row.names column with the name of each day recorded. This means that R has given each row a name corresponding to the appropriate id. R will not try to perform calculations on the row names. However, we still need to eliminate the first column in the data where the id is stored as well as the date column (data is already a row name). Try

```
npf <- npf[,-(1:2)]
View(npf)
```

Python:

```
## Here date column was converted to index and we do not need to get rid of it.
npf = npf.drop("id",axis=1)
npf
```

Now you should see that the first data column is date. Note that another column labeled row.names now appears before the data column. However, this is not a data column but rather the name that R is giving to each row.

Task c

i. Use the `summary()` function to produce a numerical summary of the variables in the data set. We notice that the variable "partlybad" is always false and thereofore useless. We opt to remove it as well.

```
summary(npf)
npf <- npf[,-2]
```

Python:

```
npf.describe()
npf = npf.drop("partlybad",axis=1)
```

ii. Use the `pairs()` function to produce a scatterplot matrix of the first ten columns or variables of the data. Recall that you can reference the columns from 3 to 12 of a matrix A using A[,3:12].

```
pairs(npf[,2:11])
```

There are several libraries to make plots. `ggpairs` from `GGally` library can be used to make a prettier plot, even though some want to do everything with `ggplot2` library:

```
library(GGally)
ggpairs(npf[,1:11],aes(colour=class4,alpha=0.4))
```

Python and Seaborn:

```
plt.clf()
sns.pairplot(npf,hue="class4",vars=npf.columns[1:11],kind="reg")
plt.show()
```

iii. Use the `plot()` function to produce side-by-side boxplots of event vs. nonevent days.

```
boxplot(RHIRGA84.mean ~ class4,npf)
```

Python and Pandas:

```
plt.clf()
npf.boxplot(column="RHIRGA84.mean",by="class4")
plt.show()
```

Python and Seaborn:

```
plt.clf()
sns.boxplot(x="class4",y="RHIRGA84.mean",data=npf)
plt.show()
```

iv. Create a new qualitative variable, called class2, which is "event" if there was a NPF event and "non-event" otherwise.

```
npf$class2 <- factor("event",levels=c("nonevent","event"))
npf$class2[npf$class4=="nonevent"] <- "nonevent"
```

Python:

```
## If you don't use dtype="object" array will cut strings...
class2 = np.array(["event"]*npf.shape[0],dtype="object")
class2[npf["class4"]=="nonevent"] = "nonevent"
npf["class2"] = class2
npf["class2"] = npf["class2"].astype("category")
```

Use the summary() function to see how many event days there are. Now use the plot() function to produce side-by-side boxplots of CS.mean versus event.

```
boxplot(CS.mean ~ class2,npf)
```

Python and Pandas:

```
plt.clf()
npf.boxplot(column="CS.mean",by="class2")
plt.show()
```

Python and Seaborn:

```
plt.clf()
sns.boxplot(x="class2",y="CS.mean",data=npf)
plt.show()
```

```
npf.describe(include="all")
```

v. Use the `hist()` function to produce some histograms with differing numbers of bins for a few of the quantitative variables. You may find the command `par(mfrow=c(2,2))` useful: it will divide the print window into four regions so that four plots can be made simultaneously. Modifying the arguments to this function will divide the screen in other ways. In Python, you can start, e.g., by reading about matplolib histograms.

vi. Continue exploring the data, and provide a brief summary of what you discover. You should find at least one more or less interesting "thing" about the data which has not already been covered above.

Problem 2

[7 points]

Objective: learning linear regression, concrete use of validation set, k-fold cross validation

Planned lecture: 10 and 12 November [Ch. 3 & 5]

In this problem we study linear regression on a synthetically generated dataset, with underlying function $f(x) = 1 + x - x^2/2$. The idea here is also to apply the theory in the problem 5 into practice. **You should collect the results of tasks b-e into a table,** where rows corresponds to polynomial degrees from 0 to 10 and columns MSE on training set (task b), MSE on validation set (task c), CV loss (task d), and loss of a model trained on training+validation set on test set (task e).

The data item (pair) (x, y) can be sampled as follows. x is sampled uniformly from interval $[-3, 3]$ (R function `runif`) and ϵ is sampled from a normal distribution with zero mean and standard deviation of 0.4 (R function `rnorm`). y is then given by $y = f(x) + \epsilon$.

Task a

First create R or Python function that takes n as an input and which outputs dataset of size n. Use your function to create a *training set* of 20 pairs, *validation set* of 20 pairs, and *test set* of 1000 pairs. In total, you should now therefore have 1040 pairs.

Task b

Study the OLS linear regression with polynomials of degree p, where $p \in \{0, 1, \ldots, 10\}$. More specifically, fit polynomials $\widehat{y} = \sum_{k=0}^{p} w_k x^k$ to the training set (of 20 data items) for different orders p by using ordinary least squares (OLS) regression. For each 11 values of p produce a plot showing the points (x_i, y_i) in the training set and the fitted polynomial in interval $[-3, 3]$. Make sure to plot the polynomials using (almost) all values of x in $[-3, 3]$, e.g., `seq(from=-3,to=3,length.out=256)`, and not only the values of x appearing in your data! Calculate and report the mean squared error (MSE) on training set, where $\text{MSE}_{tr} = \sum_{i \in S_{tr}}^{n} (y_i - \widehat{y}_i)^2 / n_{tr}$, and compare the MSE for the different orders of polynomials.

Task c

Use the 11 polynomials you found above and compute and report the MSE of the polynomials on the validation and test sets as well.

Task d

Now, instead of using simple training-validation set split combine the training and validation sets and use 10-fold cross validation to select a model based on the training+validation set (see James et al., Sec. 5.1.3). Report the cross-validation loss (Eq. (5.3) of James et al.) for different polynomial orders.

Read about the bias-variance trade-off for k-fold cross-validation. (James et al., Section 5.1.4).

Task e

Next train the regressors on combined training and validation set and report MSE on the test set. This should complete your table.

Task f

By using the table constructed, describe how you would choose the best model, if you would be given the 40 pairs in the training+validation data sets.

Hints

If you use Python you may want to experiment with ready-made scikit-learn cross-validation methods. R users may want to take a look at caret. With R, you can take a look at the hints of Problem 3 or James et al. Sect. 5.3.3. In fact you may want to do Problem 3 before problem 2, because Problem 3 has a quite comprehensive code example which may be of use here as well. :)

Problem 3

[6 points]

Learning objective: using regression models from ML libraries, generalisation, validation techniques.

Planned lecture: 12 November [Ch. 3 & 5]

Read Section 5.1 of James et al.

[v2] As you know, in supervised learning we try to find a function that produces a good estimate of the dependent variable. Consider the `Bias correction of numerical prediction model temperature forecast` dataset, available via the UCI machine learning repository: https://archive.ics.uci.edu/ml/datasets/Bias+correction+of+numerical+prediction+model+temperature+forecast and the task of predicting the next day maximum temperature (variable `Next_Tmax`) by using meteorological forecast and geographic data.

Instructions for preprocessing the data:

- Remove the following variables: categorical variables `stations` and `Date` as well as the alternate target variable `Next_Tmin`.
- Remove the rows with missing values.
- You can optionally also downsample the data, e.g., to 500 rows for speed.

[Alternatively, you can use the Boston dataset as in v1 of the problem described and available as instructed in James et al. and estimate the median value of owner-occupied homes in $1000s (variable `medv`). This dataset requires no preprocessing.]

Lets simulate missing data by choosing $n = 100$ variables in random (see the code below) by using which we train the regressor (called the *training set*). The remaining data items are used (*test set*).

You have heard that he following very good regressors that are all luckily available via SciPy or R:

- OLS linear regression
- Regression tree
- Random forest
- SVM

You can *optionally* add to your list a "dummy" regression model which always predicts the mean of the training data, irrespective of the values of the covariates. (More generally, dummy model can be thought a model that outputs a constant prediction such that the relevant loss on training data is minimized. It is often useful to include the dummy model to your list because this costs you almost nothing and it gives you a good baseline for the performance of your "real" regression models.)

Task a

Find still one more regression model implemented, e.g., in R (not the dummy model described above) and add it to your list.

Task b

Train all of the regressors on the training set and report the root mean squared errors (RMSE) on both the training and test sets. Which of the models performs best on the training set and on the test set? What does this tell about complexity of the respective model families?

Task c

Use the training set to compute the 10-fold cross-validation losses and report them as well.

Task d

Which of the four regressors is the best? How does the RMSE on the training data compare to the error on the test set? How does the error on the validation set compare to the error on the test set (if you computed the validation loss)? Could you do something with these regressors (on this training set) to make them perform better?

Hints

If you use R you can use the training / test set split given below. I also give below pointers to suitable regression models that you may want to use.

```
## Do you want to use Boston data?
## useboston <- TRUE
useboston <- FALSE

## You should usually use random seed to have repeatable results.
## I always use 42, but you are of course free to choose your own way.
## You can try to vary your random seed to see if your conclusions
## change (obviously it might, because these are random algorithms).
set.seed(42)

library(MASS)
library(rpart)        # regression trees
library(randomForest) # random forest
library(e1071)        # SVM

if(useboston) {
  X <- Boston
  X_target <- "medv"
```

```
} else {
  X <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00514/Bias_correction_ucl.csv
  X_target <- "Next_Tmax"
  ## remove categorical columns named "station" and "Date" as well as
  ## secondary target variable "Next_Tmin" (we'll just use
  ## "Next_Tmax" here)
  X <- X[,!colnames(X) %in% c("stations","Date","Next_Tmin")]
  ## omit rows with NA values
  X <- na.omit(X)
  ## take a subset of 500 rows
  X <- X[sample.int(nrow(X),500),]
}


idx <- sample.int(nrow(X),100)
data_train <- X[ idx,] # train using this data
data_test  <- X[-idx,] # holdout data for final analysis

#' root mean squared error measure
rmse <- function(yhat,y) sqrt(mean((y-yhat)**2))

#' split n items into k random folds of roughly equal size
kpart <- function(n,k) {
  p <- sample.int(n)   # random permutation of integers from 1 to n
  s <- ((0:k)*n) %/% k # "boundaries" between k folds
  mapply(function(a,b) p[a:b],s[-k-1]+1,s[-1],SIMPLIFY=FALSE)
}

#' "normalize" column names to X1, X2, X3, etc.
normcolnames <- function(X) {
  colnames(X) <- sapply(1:ncol(X),function(i) sprintf("X%d",i))
  X
}

#' Find cross validation predictions
cv <- function(data,                # dataset
               target=X_target, # default target variable
               ## separate covariates...
               data_x=data[,!(colnames(data) %in% target),drop=FALSE],
               ## ...and dependent variable
               data_y=data[,target],
               ## "normalize" column names of covariates to X1, X2, ...
               X=normcolnames(data_x),
               XY=data.frame(X,Y=data_y), # rename dependent variable as Y
               n=nrow(X),         # number of rows in the data matrix
               k=min(n,10),       # number of cross-validation folds
               split=kpart(n,k),  # the split of n data items into k folds
               model=lm,          # model to use
               ## function to train a model on data XY
               train=function(XY) model(Y ~ .,data=XY),
               ## function to make predictions on trained model
               pred=function(m,X) predict(m,newdata=X),
               ## helper function to train a model in datapoints in itr and
```

```
              ### make a prediction on datapoints in iva
              f=function(itr,iva) pred(train(XY[itr,,drop=FALSE]),X[iva,,drop=FALSE])) {
  ## initialize yhat to something of correct data type,
  ## train model with full data and make a prediction
  yhat <- f(1:n,1:n)
  if(k>0) {
    ## go through all folds, train on other folds, and make a prediction
    for(iva in split) {
      yhat[iva] <- f(setdiff(1:n,iva),iva)
    }
  }
  yhat # finally, output cross-validation predictions
}

## Dummy model is here a model that ignores the covariates and always
## predicts the mean of the training data.
dummy <- function(formula,data) {
  target <- all.vars(formula[[2]])
  a <- list(mean=mean(data[,target]))
  attr(a,"class") <- "dummy"
  a
}
predict.dummy <- function(object,newdata) {
  rep(object$mean,nrow(newdata))
}

## Some regression models implemented in R. For documentation, just type
## ?lm, ?rpart etc. Notice that you need the above mentioned libraries to be
## able to use these models.
models <- list(dummy=dummy,
               OLS=lm,
               RegressionTree=rpart,
               RandomForest=randomForest,
               SVM=svm)

## use the same random split for all - this eliminates variance due
### to different splits for different models
split <- kpart(nrow(data_train),10)

a <- sapply(models,function(model) {
  ## with k=0 we just get predictions on training data
  c(
    ## with k=0 we just get predictions on training data
    train=rmse(cv(data_train,k=0,model=model),data_train[,X_target]),
    ## we concatenate test and train data and have them on separate folds,
    ## after which we get the fold out where
    ## we trained on training data and predicted on testing data.
    ## Looks ugly, but
    ## this allows us to hide all of the details in function cv...
    test=rmse(cv(rbind(data_test,data_train),
                 split=list(1:nrow(data_test),
                            (nrow(data_test)+1):(nrow(data_test)+nrow(data_train))),
                 model=model)[1:nrow(data_test)],data_test[,X_target]),
```

```
    ## we finally do the regular 10-fold cross-validation
    CV=rmse(cv(data_train,split=split,model=model),data_train[,X_target]))
})

knitr::kable(t(a),"simple",digits=3)
```

Problem 4

[7 points]

Learning objectives: bias and variance and model flexibility

Planned lecture: 12 November [Ch. 2]

Read Section 2.2 of James et al.

Consider the bias variance decomposition in the context of model selection.

Task a

Provide a sketch of typical (squared) bias, variance, training error, test error, and Bayes (or irreducible) error curves, on a single plot, as we go from less flexible statistical learning methods towards more flexible approaches. The x-axis should represent the amount of flexibility in the method, and the y-axis should represent the values for each curve. There should be five curves. Make sure to label each one.

Explain why each of the five curves has the shape displayed.

Task b

Test the bias variance tradeoff in practice with the data generation process described in Problem 2, at point $x = 0$. Generate 1000 new training sets of 20 pairs and train a polynomial regressor $\hat{f}(x)$ for each of the degrees from 0 to 10 on each of the training sets. For each of the 1000 training sets generate additionally a test data point at $x = 0$, i.e., $(0, y_0)$, where $y_0 = f(0) + \epsilon$ and ϵ is a random variable with zero mean and variance of $\sigma^2 = 0.4^2$.

According to Eq. (2.7) of James et al., the expected (expectation over your 1000 data sets, denoted by E_D) squared loss at $x = 0$, or loss $= E_D[(y_0 - \hat{f}(0))^2]$, can then be decomposed as a sum of irreducible error (here bayes $= E_D[(y_0 - f(0))^2]$), the bias term bias^2 $= (E_D[\hat{f}(0)] - f(0))^2$, and the variance term var $= E_D[(\hat{f}(0) - E_D[\hat{f}(0)])^2]$, or loss $=$ bayes $+$ bias^2 $+$ var. Compute and plot these 4 terms (squared loss, irreducible error, bias term, variance term) at $x = 0$ as a function of polynomial degrees from 0 to 10 by using your 1000 data sets (i.e., you should end up with 4 curves). Check that the terms sum to squared loss for different polynomial degrees, i.e., verify Eq. (2.7) of James et al. Do the terms behave as you would expect from the discussion in the task a above?

Problem 5

[6 points]

Topic: theoretical properties of generalization loss and OLS linear regression [Ch. 2-3]

Planned lecture: 10 and 12 November [Ch 3]

Consider a linear regression model $\hat{f}(\mathbf{x}) = \hat{\beta}^T\mathbf{x}$, where $\hat{\beta} \in \mathbb{R}^p$ is fit by ordinary least squares (OLS) to a set of training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ drawn at random from a population, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \mathbb{R}$ for $i \in \{1, \ldots, n\}$. Suppose we have a test data $(\overline{\mathbf{x}}_1, \overline{y}_1), \ldots, (\overline{\mathbf{x}}_m, \overline{y}_m)$ drawn from the same population. Denote

$$L_{train} = \frac{1}{n}\sum\nolimits_{i=1}^{n}\left(y_i - \hat{\beta}^T\mathbf{x}_i\right)^2$$

and

$$L_{test} = \frac{1}{m} \sum_{i=1}^{m} \left(\overline{y}_i - \hat{\beta}^T \overline{\mathbf{x}}_i \right)^2.$$

Task a

Prove that L_{test} is an unbiased estimate of the generalization error for the OLS regression.

Task b

Prove that $E[L_{train}] \leq E[L_{test}]$, where the expectations are over what is random in each expression.

Task c

Explain how the result of the task b above is related to generalization problem in machine learning.

Problem 6

[6 points]

Objective: properties of estimators [Ch 3]

Planned lecture: 10 November

Lets continue with the linear regression and the toy data of Problem 2. So far, we have tried only to estimate the loss (with the purpose of choosing the best model). It is also important to be able to estimate model parameters and give them confidence bounds.

Task a

Lets consider the simplest case of 0-degree polynomial, where you want to fit the constant line $\hat{y} = w_0$ to the data. In other words, lets first just compute mean $\hat{y} = w_0 = \sum_{i \in S_{tr}} y_i / n_{tr}$.

For this task sample a new data set of 100 data points. What is the t-statistics (James et al., Sect. 3.1.2) and the corresponding 95% confidence intervals for the mean when you use the 100 data points? Can you conclude that the true mean of the data is non-zero?

Task b

Use the 20 pair training data from Problem 2 and study the coefficients $w_0, w_1, \ldots, w_p$ at least of the polynomials of degrees $p \in \{1, 2, 3, 5\}$ with R or, e.g., with corresponding Python SciPy functions, as in Sect. 3.6 of James et al. Are the estimated coefficients and confidence limits consistent with what you know about the data generating process, i.e., that the data has been generated by using a degree-2 polynomial $1 + x - x^2/2$ plus random noise?

Task c

Study the degree-1 polynomial that you fitted to the data in Task b above. You may find that the slope is positive with a high confidence. Can you therefore safely conclude that when x increases also y tends to increase? Read the Sect. 3.3.3 of James et al., which lists 6 potential problems with linear regression models. Which of the 6 problems would apply here? Explain and demonstrate the tricks and plots you could use to detect and diagnose the problem(s) (out of the 6 listed) that apply to this data.

Hint

Regarding Task c, in R, you get useful diagnostic plots by plotting the `lm` object. For more information, type `?plot.lm` to the R command prompt. With Python obtaining nice diagnostic plots requires more effort.

Problem 7

[2 points]

Objectives: self-reflection, giving feedback of the course

Tasks

- Write a learning diary of the topics of lectures 1-4 and this exercise set.

Instructions

The length of your reply should be 1-3 paragraphs of text. Guiding questions: What did I learn? What did I not understand? Was there something relevant for other studies or (future) work? You can also give feedback of the course.