**Seminar 4**

**Put the data files in the same folder as your Jupyter Notebook files if there are data files.**

**We learn coding by imitation. Therefore, we start by copying example codes and run them. Based on the outputs, comments, and the codes, we understand what the codes need and what the codes produce. Then we can modify the codes and apply them to new data for solving new problems.**

1. **Try machine learning models: Start a new Jupyter Notebook and copy the following codes one (line/part) by one, followed by press the keys Shift + Enter.**

# Machine Learning

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

## Linear and logistic regression

### Linear regression

from sklearn.datasets import fetch\_california\_housing

cali = fetch\_california\_housing()

cali

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(cali.data,

cali.target, test\_size=0.2, random\_state=0)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

from sklearn.linear\_model import LinearRegression

regr = LinearRegression()

regr.fit(X\_train, Y\_train)

Y\_pred = regr.predict(X\_test)

from sklearn.metrics import mean\_absolute\_error

print ("MAE", mean\_absolute\_error(Y\_test, Y\_pred))

### Logistic regression

import numpy as np

avg\_price\_house = np.average(cali.target)

high\_priced\_idx = (Y\_train >= avg\_price\_house)

Y\_train[high\_priced\_idx] = 1

Y\_train[np.logical\_not(high\_priced\_idx)] = 0

high\_priced\_idx = (Y\_test >= avg\_price\_house)

Y\_test[high\_priced\_idx] = 1

Y\_test[np.logical\_not(high\_priced\_idx)] = 0

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(max\_iter=10000)

clf.fit(X\_train, Y\_train)

Y\_pred = clf.predict(X\_test)

from sklearn.metrics import classification\_report

print (classification\_report(Y\_test, Y\_pred))

# macro average says the function to compute precision / recall / f1-score for each label,

# and returns the average without considering the proportion for each label in the dataset.

# weighted average says the function to compute precision / recall / f1-score for each label,

# and returns the average considering the proportion for each label in the dataset.

# Evaluate the model by means of a Confusion Matrix

from sklearn.metrics import ConfusionMatrixDisplay

import matplotlib.pyplot as plt

matrix = ConfusionMatrixDisplay.from\_estimator(clf, X\_test, Y\_test)

plt.title('Confusion Matrix')

plt.show(matrix)

plt.show()

# Evaluate the model by means of a ROC Curve, which only works for a binary target.

from sklearn.metrics import RocCurveDisplay

log\_disp = RocCurveDisplay.from\_estimator(clf, X\_test, Y\_test)

## Naive Bayes

from sklearn import datasets

iris = datasets.load\_iris()

iris

target\_names=iris.target\_names

target\_names

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(iris.data,

iris.target, test\_size=0.2, random\_state=0)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

from sklearn.naive\_bayes import GaussianNB

clf = GaussianNB()

clf.fit(X\_train, Y\_train)

Y\_pred = clf.predict(X\_test)

from sklearn.metrics import classification\_report

print (classification\_report(Y\_test, Y\_pred))

# Evaluate the model by means of a Confusion Matrix

from sklearn.metrics import ConfusionMatrixDisplay

import matplotlib.pyplot as plt

matrix = ConfusionMatrixDisplay.from\_estimator(clf, X\_test, Y\_test, display\_labels=target\_names)

plt.title('Confusion Matrix')

plt.show(matrix)

plt.show()

1. **Try Case - trading strategies - 2: Open and run the Jupyter Notebook “Seminar 2.3 CLF-TradingStrategies-1” in the previous seminar. Click the menu “Cell -> Run All” and wait for it finishes running. In the end of the file, copy the following codes one (line/part) by one, followed by press the keys Shift + Enter.**

## EDA

plt.figure()

plt.plot(dataset['3day MA'])

plt.show()

import seaborn as sns

sns.histplot(dataset,x="3day MA", kde=True, stat="density")

## Data preprocessing

# ‘X’ stores the input features, the columns starting from the fifth column (or index 4) of the

# dataset till the second last column. The last column will be stored in the dataframe y,

# which is the value we want to predict, i.e. the price rise.

X = dataset.iloc[:, 4:-1]

y = dataset.iloc[:, -1]

X

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

X\_test

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

## Fit models

from sklearn.ensemble import ExtraTreesClassifier

classifier = ExtraTreesClassifier(random\_state=101)

classifier.fit(X\_train, Y\_train)

from sklearn.metrics import classification\_report

Y\_pred = classifier.predict(X\_test)

print (classification\_report(Y\_test, Y\_pred))

## Feature importance

feature\_names=X.columns

importance = classifier.feature\_importances\_

indices = np.argsort(importance)

range1 = range(len(importance[indices]))

plt.figure()

plt.title("ExtraTreesClassifier Feature Importance")

plt.barh(range1,importance[indices])

plt.yticks(range1, feature\_names[indices])

plt.ylim([-1, len(range1)])

plt.show()

## Trading strategies

# Next, we create a new column in the dataframe dataset with the column header ‘Y\_pred’

# and store NaN values in the column. We then store the values of Y\_pred into this new

# column, starting from the rows of the test dataset. This is done by slicing the dataframe

# using the iloc method as shown in the code above. We then drop all the NaN values

dataset['Y\_pred'] = np.NaN

dataset.iloc[(len(dataset) - len(Y\_pred)):,-1] = Y\_pred

trade\_dataset = dataset.dropna()

trade\_dataset