**Seminar 5**

**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 Case - trading strategies - full: Open and run the Jupyter Notebook “Seminar 4.2 CLF-TradingStrategies-2” 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.**

# Now that we have the predicted values of the stock movement. We can compute the

# returns of the strategy. We will be taking a long position when the predicted value of y is

# true and will take a short position when the predicted signal is False.

#

# We first compute the returns that the strategy will earn if a long position is taken at the

# end of today, and squared off at the end of the next day. We start by creating a new

# column named ‘Tomorrows Returns’ in the trade\_dataset and store in it a value of 0. We

# use the decimal notation to indicate that floating point values will be stored in this new

# column. Next, we store in it the log returns of today, i.e. logarithm of the closing price of

# today divided by the closing price of yesterday. Next, we shift these values upwards by one

# element so that tomorrow’s returns are stored against the prices of today.

trade\_dataset['Tomorrows Returns'] = 0.

trade\_dataset['Tomorrows Returns'] = np.log(trade\_dataset['Close']/trade\_dataset['Close'].shift(1))

trade\_dataset['Tomorrows Returns'] = trade\_dataset['Tomorrows Returns'].shift(-1)

# Next, we will compute the Strategy Returns. We create a new column under the header

# ‘StrategyReturns’ and initialize it with a value of 0. to indicate storing floating point

# values. By using the np.where() function, we then store the value in the column

# ‘Tomorrows Returns’ if the value in the ‘ypred’ column stores True (a long position), else

# we would store negative of the value in the column ‘Tomorrows Returns’ (a short

# position); into the ‘Strategy Returns’ column.

trade\_dataset['Strategy Returns'] = 0.

trade\_dataset['Strategy Returns'] = np.where(trade\_dataset['Y\_pred'] == True, trade\_dataset['Tomorrows Returns'], - trade\_dataset['Tomorrows Returns'])

# We now compute the cumulative returns for both the market and the strategy. These

# values are computed using the cumsum() function. We will use the cumulative sum to plot

# the graph of market and strategy returns in the last step.

trade\_dataset['Cumulative Market Returns'] = np.cumsum(trade\_dataset['Tomorrows Returns'])

trade\_dataset['Cumulative Strategy Returns'] = np.cumsum(trade\_dataset['Strategy Returns'])

import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

plt.plot(trade\_dataset['Cumulative Market Returns'], color='r', label='Market Returns')

plt.plot(trade\_dataset['Cumulative Strategy Returns'], color='g', label='Strategy Returns')

plt.legend()

plt.show()

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)

## K-nearest neighbours

from sklearn import datasets

iris = datasets.load\_iris()

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.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n\_neighbors=2)

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))

## Advanced non linear algorithms

### SVM for classification

from sklearn.svm import SVC

hypothesis = SVC(kernel='rbf', random\_state=101)

# cross validation

import numpy as np

from sklearn.model\_selection import cross\_val\_score

scores = cross\_val\_score(hypothesis, X\_train, Y\_train, cv=5, scoring='accuracy')

print ("SVC with rbf kernel -> cross validation accuracy: mean = %0.3f std = %0.3f" % (np.mean(scores), np.std(scores)))

from sklearn.svm import LinearSVC

hypothesis = LinearSVC()

scores = cross\_val\_score(hypothesis, X\_train, Y\_train, cv=5, scoring='accuracy')

print ("LinearSVC -> cross validation accuracy: mean = %0.3f std = %0.3f" % (np.mean(scores), np.std(scores)))

### SVM for regression

from sklearn.datasets import fetch\_california\_housing

cali = fetch\_california\_housing()

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)

import numpy as np

from sklearn.model\_selection import cross\_val\_score

from sklearn.svm import SVR

hypothesis = SVR()

scores = cross\_val\_score(hypothesis, X\_train, Y\_train, cv=3,

scoring='neg\_mean\_absolute\_error')

print ("SVR -> cross validation accuracy (neg\_mean\_absolute\_error): mean = %0.3f std = %0.3f" % (np.mean(scores), np.std(scores)))

1. **Try case – lending club - 1: Start a new Jupyter Notebook and copy the following codes one (line/part) by one, followed by press the keys Shift + Enter.**

# Predicting LendingClub Loan Status

# [LendingClub](https://www.lendingclub.com/) is a US peer-to-peer lending company and the world's largest peer-to-peer lending platform. In this project, we build machine learning models to predict the probability that a loan on LendingClub will charge off (kind of default). These models could help LendingClub investors make better-informed investment decisions.

#

# A charge-off or chargeoff is the declaration by a creditor (usually a credit card account) that an amount of debt is unlikely to be collected. This occurs when a consumer becomes severely delinquent on a debt. Traditionally, creditors will make this declaration at the point of six months without payment. A charge-off is a form of write-off.

#

# In training the models, we only use features that are known to investors before they choose to invest in the loan.

# Import the Data

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

pd.options.mode.chained\_assignment = None

# Close the waring about "A value is trying to be set on a copy of a slice from a DataFrame"

loans = pd.read\_csv('loans\_num.csv')

loans.head()

loans.shape

loans.describe()

loans['charged\_off'].value\_counts()