Practical Session 5

1. Aim of the Session

   In this session, you will do one more principal component analysis on the wine dataset. You will learn how to train a simple linear regression model in Python. Moreover, you will go through the model selection procedure using the boston house-prices dataset. Finally, you will learn more about data manipulation with **Pandas.**

2. Task1: do a PCA analysis on the wine dataset (Please have a look at Section 4 in the notes of Practical Session 2, if you have forgotten on how to do PCA using Python.)
   1) Load wine data
   ```
   from sklearn.datasets import load_wine
   ```
   2) Investigate data: how many data points are there? How many features are there? And what are they? How many classes are included? Plot the boxplot for each feature.
   3) Pre-processing the data as follows:
   ```
   from sklearn.preprocessing import StandardScaler
   x = StandardScaler().fit_transform(data)
   ```
   4) Do PCA on the normalised wine dataset (**x**).
   Project the normalised wine data on the first two principal components. Plot the PCA visualisation plot using the scatter function. Label the data using their class label information. How much variance has been captured by the first two principal components?
   5) Plot the scree plot. To keep 80% of the total variance, how many principal components should be used?

3. Task 2: pre-process the data for a linear regression training

   1) Download the boston house-prices data by typing the following:
   ```
   from sklearn.datasets import load_boston
   boston = load_boston()
   ```
   Get the inputs and target values, and check the size of the data.

   2) Divide the data into a training set (60%), validation set (20%) and a test set (20%). For example, you may use the following code to divide the data into a training set and test set.
   ```
   from sklearn.model_selection import train_test_split

   X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.2, random_state=421, shuffle=True)
   ```
   How do you further divide the training set into a smaller training set and a validation set?

   3) Plot 13 subplots in one figure. Each subplot is a scatter plot of each feature against the target values. Use the dataset including both the training set and the validation set. You may start with the following code:

```
"""Create a plot figure"""
plt.figure(figsize=(200,100))
fig, axes = plt.subplots(nrows=3, ncols=5)
#spacing with many subplots
fig.tight_layout() # Or equivalently,  "plt.tight_layout()"

"""Create the first of two panels and set current axis"""
plt.subplot(3,5,1)
plt.plot(X_train[:,0], y_train, 'ro')
plt.xlabel('CRIM')
plt.ylabel('Target')
```

4. Task 3: run 3 linear regression models on the smaller training set and validate them on the validation set. The 3 linear regression models are: simple linear regression model, simple linear regression model with Stochastic gradient descent, simple linear regression model with polynomial features.

   1) Scaling the data first by starting with the following code and finishing it. Note that you need to use the scaling parameters obtained from the training set to scale both the training set and the validation set.

```
from sklearn import preprocessing

"""Note it should be fit on the training set only"""
scaler = preprocessing.StandardScaler().fit(X_trn)
"""Apply scaling parameters on both the training set and the validation set"""
```

   2) Simple linear regression model
   Type the following code and obtain the results. Be sure that you understand the meaning of each line.

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import numpy as np

# instantiate and fit
lm1 = LinearRegression()
lm1.fit(scaled_trnX, y_trn)

# print the coefficients
print('The intercept =',lm1.intercept_)
print('The trained coefficients are:',lm1.coef_)

# predict for the validation set
y_val_pred1=lm1.predict(scaled_valX)

# RMSE
print('RMSE =',np.sqrt(metrics.mean_squared_error(y_val, y_val_pred1)))
```

   What is the linear regression equation you have obtained?

   3) Stochastic gradient descent (SGD)

   To perform Linear Regression using SGD in **sklearn**, you can use the **SGDRegressor** class, which defaults to optimizing the squared error cost function. To see the meaning of each argument of **SGDRegressor** and understand the mathematical model you will use, you need to check the following link:
   https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html

   Type the following code and finish the training and the validation parts. Comparing the results with those obtained from **lm1**.

```
from sklearn.linear_model import SGDRegressor

# instantiate and fit
lm2 = SGDRegressor(max_iter=50, tol=1e-3, penalty=None, eta0=0.1)
```

4) Using polynomial features

Type the following code and finish the training and the validation parts. Comparing the results with those obtained from **lm1** and **lm2**.

```
from sklearn.preprocessing import PolynomialFeatures

#transforms the existing features to higher degree features.
poly_reg=PolynomialFeatures(degree=2)
scaler=poly_reg.fit(scaled_trnX)
scaled_trnX_poly = scaler.transform(scaled_trnX)
scaled_valX_poly = scaler.transform(scaled_valX)

## fit the transformed features to Linear Regression
lm3 = LinearRegression()
```

If you want to check which features have been used, you may type the following

```
scaler.get_feature_names()
```

Which model gives the best result on the validation set?

5. Task 4: run a linear regression model with the best model you have identified in Task3 on the training set, that is the combination of the smaller training set and the validation set, and then test it on the test set. Report the obtained RMSE (root mean square error) on the test set.

Note that you need to re-scale the data first. The scaling parameters should be obtained from the dataset including both the training set and the validation set. Use the same scaling parameters to scale the test set.

Reflection: In tasks 3 and 4, you have gone through a basic model selection procedure you need to apply in machine learning applications. Take some time to recall how you did it and extract the general steps you need to follow when analysing different datasets..

6. Data Manipulation with **Pandas** (continuous)

- Dealing with problematic data

In practice, it is very common to have messy data including missing values and dates. You need to download a_messy_example.csv from Canvas and open it using pandas as follows:

```
import pandas

messy_data = pandas.read_csv('a_messy_example.csv', sep=',')
messy_data
```

	Date	Temperature_city_1	Temperature_city_2	Destination
0	20171210	5.0	20	1
1	20171211	2.0	23	1
2	20171212	3.0	19	2
3	20171213	8.0	21	1
4	20171214	5.0	20	2
5	20171215	NaN	21	1
6	20171216	3.0	18	2

As can be seen, all the data, even the dates, has been parsed as integers. You may try the following code to convert dates into the proper format.

```
messy_data = pandas.read_csv('a_messy_example.csv', sep=',', parse_dates=[0])
```

In addition, to get rid of the missing data that is indicated as NaN, you may either fill in a proper value, or drop off the row including the missing values. For example,

	Date	Temperature_city_1	Temperature_city_2	Destination
0	2017-12-10	5.0	20	1
1	2017-12-11	2.0	23	1
2	2017-12-12	3.0	19	2
3	2017-12-13	8.0	21	1
4	2017-12-14	5.0	20	2
5	2017-12-15	4.0	21	1
6	2017-12-16	3.0	18	2

Task5:

Drop off the row from the messy data using the method.dropna() and check the resulting data.

Note that another possible problem when handling real world datasets is the loading of a dataset with errors or bad lines. A possible workaround, which is not always feasible, is to ignore this line [2]. For example, suppose that you have a badly formatted data file called a_messy_data2.csv. What you may try to do is as follows:

```
bad_data = pandas.read_csv('a_messy_data2.csv', error_bad_lines=False)
```

- Data selection
  - Removing the index column

    As you may have noticed, the first column of data which are read into Python is index. This index column should not be used by mistake as a feature. If you need to remove it, you may do the following:

```
messy_data = pandas.read_csv('a_messy_example.csv', sep=',', index_col =0)
messy_data
```

Date	Temperature_city_1	Temperature_city_2	Destination
20171210	5.0	20	1
20171211	2.0	23	1
20171212	3.0	19	2
20171213	8.0	21	1
20171214	5.0	20	2
20171215	NaN	21	1
20171216	3.0	18	2

Task6:

Please remove the index column when load the Iris data using pandas.

- Access the value of a cell

You can simply specify the column and the line you are interested in. Note that it is not a matrix. This is actually a Panda's DataFrame, and the [ ] operator works first on columns and then on the element of the resulting pandas Series.

```
iris_data['petal_width'][10]
```

```
0.2
```

If you want to access the data using something similar to access matrices, you may use the .loc() method.

```
iris_data.loc[10, 'petal_width']
```

```
0.2
```

Task7:

Access the fifth value under the column of 'sepal_length' using the both methods described above.

- Access values of sub-matrixes

You may access values of sub-matrixes as follows:

```
iris_data[['sepal_length', 'petal_width']][0:10]
```

Or

```
iris_data.loc[range(0,10), ['sepal_length', 'petal_width']]
```

- Access values using .loc() with conditions

For example, if you want to find all patterns with a value of **sepal_width** less than 5cm, you may type the following:

```
iris_data.loc[iris_data.sepal_width<5]
```

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

.
.
.

Furthermore, type the following to see what you get:

```
iris_data.loc[(iris_data.sepal_width<5) & (iris_data.sepal_width>4) ]
```

7. References

[1] Aurelien Geron: Hands-on Machine Learning with Scikit-Learn and Tensorflow (Chapter 4), 2017, O'Relly.

[2] Jake VanderPlas: Python Data Science Handbook, Page 390-405, 2016, O'Relly.

[3] Evaluating a Linear Regression model: https://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/