COMP809 – Data Mining and Machine Learning

Lab 10

- **Part 1 Regression**
  - house prices prediction

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RANSACRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt

# Linear Regression Refer
https://scikitlearn.org/stable/modules/generated/sklearn.linear
_model.LinearRegression.html
# for more details. Loading housing dataset in to a dataframe.
This dataset has 14 features with 506samples. MEDV is
# the target variable

df = pd.read_csv('https://raw.githubusercontent.com/rasbt/'
                 'python-machine-learning-book-2nd-edition'
                 '/master/code/ch10/housing.data.txt',
header=None, sep='\s+')
df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM',
'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

print(df.head())

# Let's use the RM (number of rooms) variable from the Housing
dataset as the
# explanatory variable and train a model that can predict MEDV
(house prices).
# We only use one feature here, just to explain the concepts
and applicability of the model.

X = df[['RM']].values
y = df['MEDV'].values

# Applying Sklearn linear regression model
```

```
slr = LinearRegression()
slr.fit(X, y)
print('Slope: %.3f' % slr.coef_[0])
print('Intercept: %.3f' % slr.intercept_)


# A function that will plot a scatterplot of the training
samples and add the regression line
def lin_reg_plot(X, y, model):
    plt.scatter(X, y, c='steelblue', edgecolor='white', s=70)
    plt.plot(X, model.predict(X), color='black', lw=2)
    plt.xlabel('Average number of rooms [RM]')
    plt.ylabel('Price in $1000s [MEDV]')
    plt.show()


# Plotting MEDV against RM by calling the lin_regplot function
lin_reg_plot(X, y, slr)

# Now we use Random Sample Consensus (RANSAC) algorithm, which
identifies
# outliers and fits a regression model to a subset of the data,
the so-called inliers.

ransac = RANSACRegressor(LinearRegression(), max_trials=100,
min_samples=50,
                         loss='absolute_loss',
residual_threshold=5.0, random_state=0)
ransac.fit(X, y)
# After we fit the RANSAC model, let's obtain the inliers and
outliers from the fitted
# RANSAC-linear regression model and plot them together with
the linear fit:
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
line_X = np.arange(3, 10, 1)
line_y_ransac = ransac.predict(line_X[:, np.newaxis])
plt.scatter(X[inlier_mask], y[inlier_mask], c='steelblue',
edgecolor='white',
            marker='o', label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask], c='limegreen',
edgecolor='white',
            marker='s', label='Outliers')
plt.plot(line_X, line_y_ransac, color='black', lw=2)
plt.xlabel('Average number of rooms [RM]')

plt.ylabel('Price in $1000s [MEDV]')
```

```
plt.legend(loc='upper left')
plt.show()

# Now we use same housing dataset to do the actually implement
the model, using training
# and testing sets.
X = df.iloc[:, :-1].values
y = df['MEDV'].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)
slr = LinearRegression()
slr.fit(X_train, y_train)
y_train_pred = slr.predict(X_train)
y_test_pred = slr.predict(X_test)

# Since our model uses multiple explanatory variables, we can't
visualize the linear
# regression line (or hyperplane to be precise) in a two-
dimensional plot, but we can plot
# the residuals (the differences or vertical distances between
the actual and predicted
# values) versus the predicted values to diagnose our
regression model.

plt.scatter(y_train_pred, y_train_pred - y_train,
c='steelblue', marker='o',
            edgecolor='white', label='Training data')
plt.scatter(y_test_pred, y_test_pred - y_test, c='limegreen',
marker='s',
            edgecolor='white', label='Test data')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-10, xmax=50, color='black', lw=2)
plt.xlim([-10, 50])
plt.show()

# Using MSE(Mean Square Error to measure the performance)
print('MSE train: %.3f, test: %.3f' %
(mean_squared_error(y_train, y_train_pred),

mean_squared_error(y_test, y_test_pred)))
#############################################################
# You can refer
https://scikitlearn.org/stable/modules/generated/sklearn.linear
_model.LogisticRegression.html to
# understand on Logistic Regression
```

Lab 10

- **Part 2 LSTM vs Regression**
  – Data visualization

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df = pd.read_csv("TSLA.csv")
print('Number of rows and columns:', df.shape)
df.head(5)

# Visualising the results
plt.figure(figsize=(16, 8))
plt.plot(df["Date"], df["Close"], color='blue', label='Close
Price history')
plt.title('TESLA - Close Price history')
plt.xlabel('Time')
plt.ylabel('TESLA Stock Price')
plt.legend()

# x axix ticks may be too messy
x_ticks_length = np.arange(len(df['Date']))
plt.xticks(x_ticks_length[::30], df['Date'][::30], rotation=45)

plt.show()
plt.savefig('./output/TESLA_Stock_plot.png', dpi=800)
```

Lab 10

- **Part 2 LSTM vs Regression**
  – Linear Regression

Refer
https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html for more details.

```
import matplotlib.pyplot as plt
import pandas as pd
from fastai.tabular.core import add_datepart
from sklearn.linear_model import LinearRegression
import numpy as np

pd.options.mode.chained_assignment = None

data = pd.read_csv("TSLA.csv")
print('Number of rows and columns:', data.shape)
data.head(5)

# creating a separate dataset
new_data = data[['Date', 'Close']]
# reset index to the timestamp, instead of numeric sequential
values like : 1,2,3,4
new_data.index = new_data.Date

"""
Let's say, today is
2021,09,30

and you want to predict the stock price of TESLA on 2022,01,01

so you know the value of : Open,High,Low,Adj Close,Volume ?
you can't know it, right?

So, it means you don't have applicable features.
The only feature you can use, is the timestamp.

Thus, we have to create some other features from the timestamp.
"""

add_datepart(new_data, 'Date')
new_data.drop('Elapsed', axis=1, inplace=True)  # elapsed will
be the time stamp

"""
```

```
This creates features such as:

'Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear',
'Is_month_end', 'Is_month_start', 'Is_quarter_end',
'Is_quarter_start',
'Is_year_end', and  'Is_year_start'.

Note: I have used add_datepart from fastai library.
If you do not have it installed,
you can simply use the command pip install fastai.
Otherwise, you can create these feature using simple for loops
in python.
I have shown an example below.

Apart from this,
we can add our own set of features
that we believe would be relevant for the predictions.
 For instance,
 my hypothesis is that the first and last days of the week
 could potentially affect the closing price of the stock
 far more than the other days.
  So I have created a feature
   that identifies whether a
    given day is Monday/Friday
    or Tuesday/Wednesday/Thursday.
This can be done using the following lines of code:


new_data['mon_fri'] = 0
for i in range(0, len(new_data)):
    if (new_data['Dayofweek'][i] == 0 or
new_data['Dayofweek'][i] == 4):
        new_data['mon_fri'][i] = 1
    else:
        new_data['mon_fri'][i] = 0

"""

# split into train and validation
train = new_data[:987]
valid = new_data[987:]

x_train = train.drop('Close', axis=1)
y_train = train['Close']
x_valid = valid.drop('Close', axis=1)
y_valid = valid['Close']
```

```
model = LinearRegression()
model.fit(x_train, y_train)

# make predictions and find the rmse
preds = model.predict(x_valid)
rms = np.sqrt(np.mean(np.power((np.array(y_valid) -
np.array(preds)), 2)))
print(rms)

# for plotting
# valid.loc[0, valid.columns.get_loc('Predictions')] = preds

train = new_data[:987]
valid = new_data[987:]
valid['Predictions'] = preds

plt.figure(figsize=(16, 8))
plt.plot(train['Close'])
[plt_1, plt_2] = plt.plot(valid[['Close', 'Predictions']])
plt.legend([plt_1, plt_2], ['Close', 'Predictions'])

# x axix ticks may be too messy
x_ticks_length = np.arange(len(data['Date']))
plt.xticks(x_ticks_length[::30], data['Date'][::30],
rotation=45)
plt.show()

# Let's save the file, and insert it to your report!!
# 987 1258
plt.savefig('./output/TESLA_regression_plot.png', dpi=800)
```

Lab 10

- **Part 2 LSTM vs Regression**
  – LSTM

In order to use LSTM, you have to install TensorFlow.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import *
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.metrics import MeanAbsoluteError,
RootMeanSquaredError
from tensorflow_addons.metrics import RSquare

data = pd.read_csv("TSLA.csv")
print('Number of rows and columns:', data.shape)
data.head(5)

# creating a separate dataset
new_data = data[['Date', 'Close']]
# reset index to the timestamp, instead of numeric sequential
values like : 1,2,3,4
new_data.index = new_data.Date
new_data.drop('Date', axis=1, inplace=True)

# creating train and test sets
dataset = new_data.values

train = dataset[0:987, :]
valid = dataset[987:, :]

# It's a good idea to normalize the data before model fitting.
# This will boost the performance. You can read more here for
the Min-Max Scaler:
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
```

```
"""
Now the time-step value will be 60. Let's split the data X, Y.
In the 0th iteration the
first 60 elements goes as your first record
 and the 61 elements will be put up in the prediction.

 1 to 60 as a batch to fit the closing price on timestamp 61
 2 to 61 as a batch to fit the closing price on timestamp 62
 3 to 62 as a batch to fit the closing price on timestamp 63
 4 to 63 as a batch to fit the closing price on timestamp 64
 ........
"""
time_step = 60  # step = 100 ? 600 ? whatever

# converting dataset into x_train and y_train
x_train, y_train = [], []
for i in range(time_step, len(train)):
    x_train.append(scaled_data[i - time_step:i, 0])
    y_train.append(scaled_data[i, 0])
x_train, y_train = np.array(x_train), np.array(y_train)

x_train = np.reshape(x_train, (x_train.shape[0],
x_train.shape[1], 1))

# create and fit the very simple LSTM network
# Now, it's time to build the model.
# We will build the LSTM with 50 neurons and 2 hidden layers.
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(x_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(1))

"""
In next assignment,
for all the models, provide root mean square error (RMSE),

Mean Absolute Error (MAE) and correlation
coefficient (R2 ) to quantify the prediction performance of
each model.
mean_absolute_error, RootMeanSquaredError

from tensorflow_addons.metrics import RSquare
https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/R
Square
"""
```

```
model.compile(loss='mean_squared_error',
              optimizer=Adam(learning_rate=0.01),
              # metrics=MeanAbsoluteError()  # r_square(),
mean_absolute_error(), RootMeanSquaredError()
              )

model.fit(x_train,
          y_train,
          epochs=1,  # 100
          batch_size=4
          )

# Prepare the test data (reshape them):
# predicting values, using past time_step from the train data
inputs = new_data[len(new_data) - len(valid) -
time_step:].values
inputs = inputs.reshape(-1, 1)
inputs = scaler.transform(inputs)

X_test = []
for i in range(time_step, inputs.shape[0]):
    X_test.append(inputs[i - time_step:i, 0])

X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],
1))
closing_price = model.predict(X_test)
# because we have scaled the data, so we should inverse the
scale
closing_price = scaler.inverse_transform(closing_price)

# we can get the error, in this example, it is
mean_absolute_error:
# print('Mean Absolute Error:', model.evaluate(X_test))

# you can also calculate the error by formula, such as rmse
rms = np.sqrt(np.mean(np.power((valid - closing_price), 2)))
print('rmse:', rms)

correlation_matrix = np.corrcoef(
    np.reshape(valid, valid.shape[0]),
    np.reshape(closing_price, closing_price.shape[0])
)
correlation_xy = correlation_matrix[0, 1]
r_squared = correlation_xy ** 2
print('RSquare:', r_squared)
```

```
"""
 stock price is affected by the news about the company
  and other factors like demonetization or merger/demerger of
the companies.
  There are certain intangible factors as well which can often
be impossible to predict beforehand.
"""

# for plotting
train = new_data[:987]
valid = new_data[987:]
pd.options.mode.chained_assignment = None
valid['Predictions'] = closing_price

plt.figure(figsize=(16, 8))
plt.plot(train['Close'])
[plt_1, plt_2] = plt.plot(valid[['Close', 'Predictions']])
plt.legend([plt_1, plt_2], ['Close', 'Predictions'])

x_ticks_length = np.arange(len(data['Date']))
plt.xticks(x_ticks_length[::30], data['Date'][::30],
rotation=45)

plt.show()
plt.savefig('./output/TESLA_LSTM_plot.png', dpi=800)

"""
In the lab,
we don't have time to run a complex model.
If you are using CUDA-enabled graphics cards (such as 2070
super)
This may be finish in 5 minutes.
if your GPU is not very good, it may require 10 minutes

model = Sequential()
#Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True, input_shape
= (X_train.shape[1], 1)))
model.add(Dropout(0.2))
# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a fourth LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
```

```
model.add(Dropout(0.2))
# Adding the output layer
model.add(Dense(units = 1))
# Compiling the RNN
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
# Fitting the RNN to the Training set
model.fit(X_train, y_train, epochs = 100, batch_size = 32)
"""
```