Sanaz Salari

Process Modelling Assignment- Reactive-ION Plasma Etching Process:

This problem is a multi-input and multi-output regression issue that we would like to solve by MLP algorithm. This is a regression problem since it involves four numerical continuous output variables which we want to predict as target values. The task here is to design and train an MLP to minimize the error between actual and predicted output values. We have 53 input-output data pairs, 6 features, and 4 outputs. The steps I did from beginning to end will be explained in the following.

1. **install and import the required libraries** such as TensorFlow and Keras and fix random seed for reproducibility:

```
import sklearn
import skopt
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import RepeatedKFold
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.models import Sequential
from keras.layers import Dense, Dropout
np.random.seed(5)
```

2. **Import the data:**
   from the local drive and do preprocess.

```
#Load data from local drive
from google.colab import files
import pandas as pd
import io
uploaded = files.upload()
```

```
Choose Files  Plasma_Dataset.csv
• Plasma_Dataset.csv(application/vnd.ms-excel) - 2500 bytes, last modified: 2/23/2022 - 100% done
Saving Plasma_Dataset.csv to Plasma_Dataset.csv
```

```
[3] df = pd.read_csv(io.BytesIO(uploaded['Plasma_Dataset.csv']))
    del df["run"] # drop "run" column
    df.head()
```

	pressure	rf_power	electrode_gap	cci_flow	he_flow	o2_flow	etch_rate	etch_uniformity	oxide_selectivity	photoresist_selectivity
0	300	300	1.8	100	200	20	3491	14.2	6.48	2.01
1	200	400	1.8	100	50	10	3884	3.9	5.98	1.91
2	200	400	1.2	150	200	20	4931	24.8	5.39	1.85
3	300	400	1.8	150	200	20	4726	6.6	5.97	2.11
4	200	400	1.2	150	50	10	5089	12.4	5.61	2.16

3. Check NA values in a dataset

```
#Clean the data
df.isna().sum()
```

```
pressure                0
rf_power                0
electrode_gap           0
cci_flow                0
he_flow                 0
o2_flow                 0
etch_rate               0
etch_uniformity         0
oxide_selectivity       0
photoresist_selectivity 0
dtype: int64
```

4. check the types of input and output values

To see whether we have any categorical data and if there are such values, convert them to numerical ones.

5 . Split the data into training and test sets

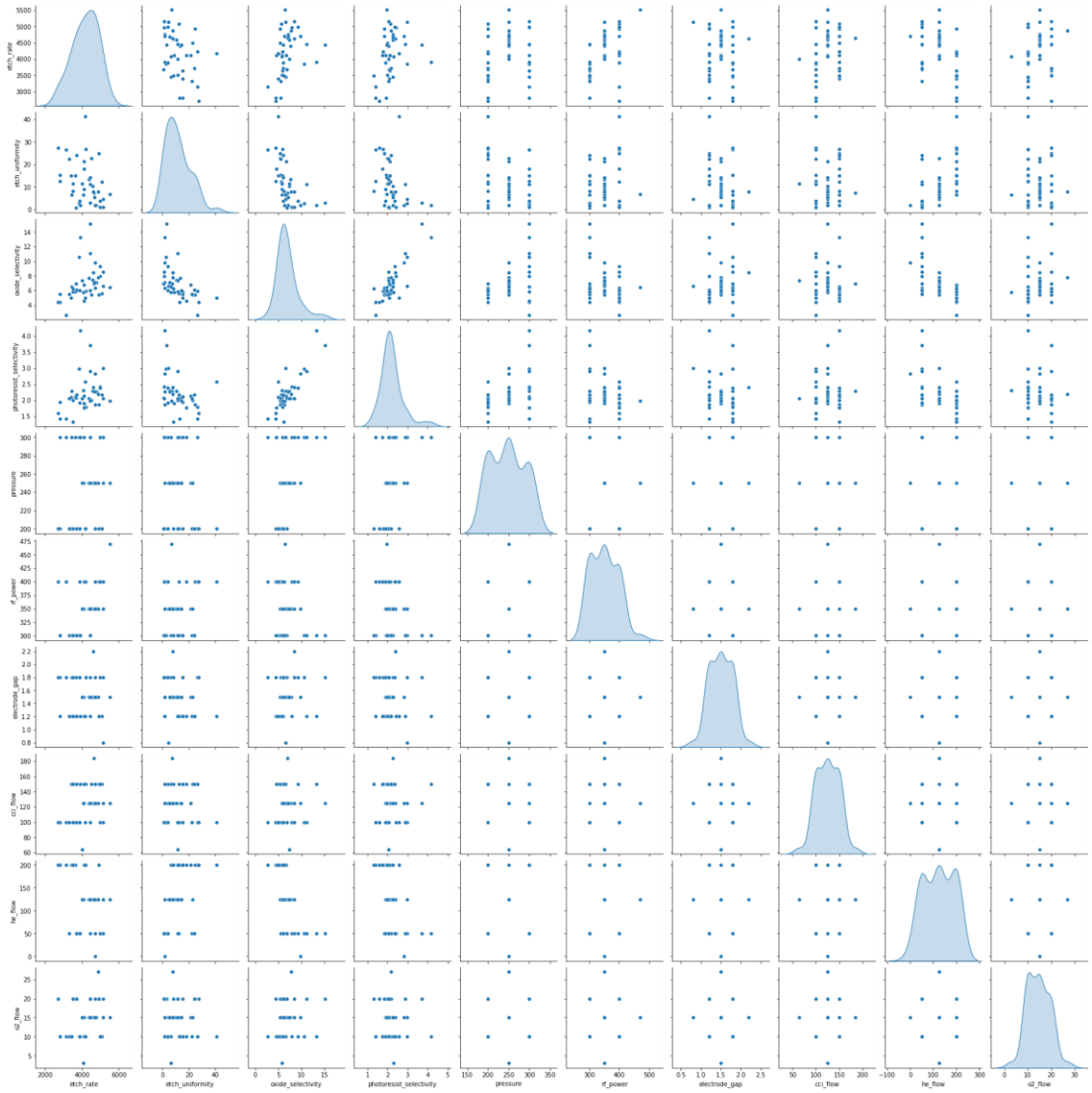
80% of data is considered as a training dataset and the remaining 20% as a test dataset.

```
df2 = df.copy()
#Split the data into training and test sets
df2_x = df.sample(frac=0.8, random_state=0)
#df_x = test_dataset.copy()
df2_y = df2_x[['etch_rate','etch_uniformity','oxide_selectivity','photoresist_selectivity']]
df2_x.drop(['etch_rate','etch_uniformity','oxide_selectivity','photoresist_selectivity'], axis = 1, inplace = True)

df2_x_test = df.drop(df2_x.index)
df2_y_test = df2_x_test[['etch_rate','etch_uniformity','oxide_selectivity','photoresist_selectivity']]
df2_x_test.drop(['etch_rate','etch_uniformity','oxide_selectivity','photoresist_selectivity'], axis = 1, inplace = True)

print(df2_x.shape, df2_y.shape)
print(df2_x_test.shape, df2_y_test.shape)
```

```
(42, 6) (42, 4)
(11, 6) (11, 4)
```

6. **Inspect the data and review the joint distribution of a few pairs of columns from the training set**

Inspect training data to see what the relationships are. By running pairs plot in Python using seaborn visualization library, we see that distribution of single variables such as “etch_uniformity”, “oxide_selectivity”, “pressure”, “rf_power“, and “he_flow” are not normalized. Besides, there is no clear relationship between input variables. In other words, the input variables are independent. So, none of them are redundant. Also, the relationship between all input and output variables are non-linear. There is almost a linear relationship between two output variables, “oxide_selectivity” and “photoresist_selectivity”.



7. To see whether we need normalization or not, in the table of statistics we see the ranges of features are different in comparison to each other's:
`train_dataset.describe().transpose()[['mean', 'std']]`

	mean	std
pressure	247.619048	39.743662
rf_power	350.452381	43.037984
electrode_gap	1.500000	0.284562
cci_flow	124.952381	23.916473
he_flow	123.809524	63.905509
o2_flow	14.523810	4.758756
etch_rate	4183.547619	695.546537
etch_uniformity	11.723810	9.312431
oxide_selectivity	6.948333	2.338508
photoresist_selectivity	2.208810	0.543728

8. To decrease the training model sensitivity to features scale, we need to normalize all training features (not outputs) which have different ranges to change their values to a common scale. Normalization is important because the input variables will be multiplied by the model weights and the scale of outputs will be affected by the scale of features. The goal is to predict output with the best accuracy, but we don't want to change the output.

9. **Normalization**

   To decrease the training model sensitivity to features scale, we need to normalize all training features (not outputs) which have different ranges to change their values to a common scale. Normalization is important because the input variables will be multiplied by the model weights and the scale of outputs will be affected by the scale of features. The goal is to predict output with the best accuracy, but we don't want to change the output.

```
# Normalization:
df2_x_n=(df2_x - df2_x.min())/(df2_x.max()-df2_x.min())
df2_y_n=(df2_y - df2_y.min())/(df2_y.max()-df2_y.min())
X2 = df2_x_n.to_numpy()
y2 = df2_y_n.to_numpy()
Y2 = df2_y.to_numpy()

# Normalization on test data
df2_x_test_n=(df2_x_test - df2_x.min())/(df2_x.max()-df2_x.min())
df2_y_test_n=(df2_y_test - df2_y.min())/(df2_y.max()-df2_y.min())


X2_test = df2_x_test_n.to_numpy()
y2_test = df2_y_test_n.to_numpy()
Y2_test = df2_y_test.to_numpy()
```

10. **Define & Compile Keras Model**

- Defining various initialization parameters for MLP model

```
num_features = df_x_n.shape[1]; num_classes = df_y.shape[1]
```

- Let's create a helper function that builds the model with various parameters. Builds a Sequential MLP model using Keras.
- The model has two hidden layers and one output layer.
- Activation function for each hidden layer is “Relu” and for the output layer is nothing which means linear since we have a regression problem.
- The regularization method is using the l1_l2 and two ‘dropout’ layers for each hidden layer.
- As we want to solve a regularization problem, the loss function is “MSE”.
- The optimizer is ‘adam’.
- The metric is “mse”.

```
# Define & Compile Keras Model w regulizers for Hyper-Parameter Optimization
from tensorflow.keras import regularizers
# Defining various initialization parameters for MLP model
num_features = df2_x_n.shape[1]; num_classes = df2_y.shape[1]

# Let's create a helper function first which builds the model with various parameters.
def get_model_reg(dense_0_neurons, dense_1_neurons, dropout_rate0, dropout_rate, input_dim, num_classes):  #dropout_rate0,
    # Builds a Sequential MLP model using Keras and returns it

    # Define the keras model
    model = Sequential()
    model.add(Dense(dense_0_neurons, input_dim=input_dim, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-4),
                    name="dense_1"))
    model.add(Dropout(dropout_rate0, name="dropout0"))
    model.add(Dense(dense_1_neurons, activation='relu',
                    kernel_regularizer=regularizers.l1_l2(l1=1e-4, l2=1e-4),
                    bias_regularizer=regularizers.l2(1e-4),
                    activity_regularizer=regularizers.l2(1e-4),
                    name="dense_2"))

    model.add(Dropout(dropout_rate, name="dropout"))
    model.add(Dense(num_classes, name="dense_3"))
    # Compile the keras model for a specified number of epochs.
    model.compile(loss='mean_squared_error',
                  optimizer='adam',
                  metrics=['mse'])
    return model
```

11. Setup Keras Model with layer Regularizers and cross validation for Scikit-Optimizer to implement on train data (X2, y2) { we use test data(x2_test, y2_test) to evaluate the model}

```
# Setup Keras Model with layer Regularizers and cross validation for Scikit-Optimizer to implement on train data (X2, y2) { we use test data(x2_test, y2_test)
import skopt
from skopt import gp_minimize
# Specify `Static' Parameters
STATIC_PARAMS = {num_features, num_classes}

# Bounded region of parameter space
# The list of hyper-parameters we want to optimize. For each one we define the
# bounds, the corresponding scikit-learn parameter name, as well as how to
# sample values from that dimension (`'log-uniform'` for the dropout_rate)
SPACE = [skopt.space.Integer(2, 12, name='dense_0_neurons'),
         skopt.space.Integer(2, 12, name='dense_1_neurons'),
         skopt.space.Real(0.0, 0.4, name='dropout_rate'),
         skopt.space.Real(0.0, 0.4, name='dropout_rate0')]

# This decorator allows your objective function to receive a the parameters as
# keyword arguments. This is particularly convenient when you want to set
# scikit-learn estimator parameters
@skopt.utils.use_named_args(SPACE)

# Define objective for optimization
def objective3(**params):

    # All parameters:
    #all_params = {**params, **STATIC_PARAMS}
    results = list()
    # define evaluation procedure
    cv = RepeatedKFold(n_splits=5, n_repeats=4, random_state=1)
    # enumerate folds
    for train_ix, test_ix in cv.split(X2):
      # prepare data
```

```
    for train_ix, test_ix in cv.split(X2):
      # prepare data
      X_train, X_test = X2[train_ix], X2[test_ix]
      y_train, y_test = y2[train_ix], y2[test_ix]
    # define model
      model = get_model_reg(params["dense_0_neurons"], params["dense_1_neurons"], params["dropout_rate0"], params["dropout_rate"],num_features, num_classes)  #pa

      # Fit keras model
      history = model.fit(X_train, y_train, epochs=32, verbose=0)
      loss_epoch = history.history['loss']

    #sklearn.metrics.r2_score(y_true, y_pred)
    # Evaluate the model with the eval dataset.
      score = model.evaluate(X_test, y_test, verbose=0 ) ###, batch_size=4, verbose=0) # thei evaluation is a preliminary evaluation because it is done on the sa

      # store result
      #print('>%.3f' % score[0])
      results.append(score[0])

    #test_loss = model.evaluate(X_test, y_test)
      print('CV MSE: %.3f:' % score[0])
    print('Cross validation average MSE is %.3f:' % np.mean(results))
    # Return the accuracy.
    return np.mean(results)

########################################################################
########################################################################
#Run Scikit-Optimizer
import warnings
warnings.filterwarnings('ignore')

results_gp = skopt.gp_minimize(objective3,     # the function to minimize
                               SPACE,              # the bounds on each dimension of x
```

The result would be as follows:

```
CV MSE: 0.111:
CV MSE: 0.117:
CV MSE: 0.083:
CV MSE: 0.168:
CV MSE: 0.122:
CV MSE: 0.247:
CV MSE: 0.242:
CV MSE: 0.094:
CV MSE: 0.171:
CV MSE: 0.206:
CV MSE: 0.124:
Cross validation average MSE is 0.173:
CV MSE: 0.124:
CV MSE: 0.130:
CV MSE: 0.111:
CV MSE: 0.196:
CV MSE: 0.075:
CV MSE: 0.177:
CV MSE: 0.175:
CV MSE: 0.214:
CV MSE: 0.085:
CV MSE: 0.094:
CV MSE: 0.155:
CV MSE: 0.117:
CV MSE: 0.161:
CV MSE: 0.108:
CV MSE: 0.152:
CV MSE: 0.128:
CV MSE: 0.112:
CV MSE: 0.237:
CV MSE: 0.127:
CV MSE: 0.178:
Cross validation average MSE is 0.143:
```

The value of “MSE” decreased by each epoch on training dataset and using K-fold to train the model.

12. Get Best Parameter Set

```
#Get Best Parameter Set
"Best score=%.4f" % results_gp.fun

print("""Best parameters:
- dense_0_neurons=%d
- dense_1_neurons=%d
- dropout_rate0=%.6f
- dropout_rate=%.6f""" %
      (results_gp.x[0], results_gp.x[1],
                        results_gp.x[2], results_gp.x[3]))
# Model evaluation:
model_opt = get_model_reg(results_gp.x[0], results_gp.x[1], results_gp.x[2], results_gp.x[3], num_features, num_classes)
score_test = model_opt.evaluate(X2_test, y2_test, verbose=0 )
print(' MSE on test dataset:', score_test[0])
```

```
Best parameters:
- dense_0_neurons=9
- dense_1_neurons=8
- dropout_rate0=0.214949
- dropout_rate=0.303446
 MSE on test dataset: 0.33775049448013306
```

13. Results

Average of MSE on normalized training data is small (0.171) and MSE on n ormalized test data is reasonably small as well although larger as expected .