#\*\*PREDICT MRR OF WAFER BY CMP MODELLING\*\*

\*\*Programming Language Used\*\* : Python

\*\*Executed on\*\* : Hadoop Big Data Platform

\*\*Visualization Tool Used\*\* : Apache Zeppelin

## OVERVIEW

This learning lab can be used as a guide to predict MRR(Material Removal Rate) of wafers by modelling CMP(Chemical Mechanical Planarization). The main objective here is to get familiar with:

1. What is CMP (Chemical Mechanical Planarization)?

2. How to measure and inspect CMP?

3. How to predict MRR (Material Removal Rate) by modelling CMP?

We will be using DevNet Data Learning Platform referred as "DLP" during the course.

## PRE-REQUISITES

1. Install Chrome Browser.

2. Basic knowledge of Python.

3. Basic knowledge of Machine Learning.

4. Obtain access to Data Learning Platform -

Refer: https://github.com/CiscoDevNet/data-dev-learning-labs/blob/master/labs/WORD%20COUNT%20USING%20SCALA%20WITH%20APACHE%20SPARK/HowToAccessDTLP.md

## LEARNING OBJECTIVES

1. To get familiarized with the DLP (Data Learning Platform).

2. To get familiarized with writing and executing python code in Zeppelin.

## TERMINOLOGIES USED

### APACHE ZEPPELIN - AN INTRODUCTION

Zeppelin is a web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive and collaborative documents with SQL, Scala and more.

Zeppelin enables data engineers, data analysts and data scientists to be more productive by developing, organising, executing, and sharing data code and visualising results without referring to the command line or knowing the cluster details. It brings data exploration, visualization, sharing and collaboration features to Spark. It supports Python and also a growing list of programming languages such as Scala, Hive, SparkSQL, shell and markdown. The various languages are supported via Zeppelin language interpreters. ZeppelinΓÇÖs notebooks provides interactive snippet-at-time experience to data scientist.

###### Key Features:

1. Web based notebook style editor.

2. Built-in Apache Spark support.

###### Use Cases for Zeppelin:

1. Data Exploration and discovery

2. Data Visualization - Tables, graphs and charts

3. Interactive snippet-at-a-time experience

4. Collaboration and publishing

For more details, please refer: https://zeppelin.apache.org/

## PROCESS OVERVIEW

![alt-tag](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/WORD%20COUNT%20USING%20SCALA%20WITH%20APACHE%20SPARK/assets/images/ProcessXX.jpeg?raw=true)

Please follow the steps given below to launch the workspace and execute the lab.

### \*\*Step 1 : Select Learning Lab from DLP\*\*

After launching the DLP dashboard page, navigate to learning labs tab. Select the learning lab "Predict MRR of Wafer by CMP Modelling" and click on "Start" button as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR1.jpeg?raw=true)

### \*\*Step 2 : Workspace Page\*\*

On click of Start button, user will be navigated to a workspace page where all the components - IDE, Tools and Microservices required to execute the program are available. Please refer the screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR2.jpeg?raw=true)

### \*\*Step 3 : Start Service in Workspace\*\*

Points to Note:

\* Start the Zeppelin service.Please refer screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR3.jpeg?raw=true)

\* Once started, the colour of the icon changes to green and launch button is enabled.

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR4.jpeg?raw=true)

\* If the task is in stopped status then click on it to start again.

### \*\*Step 4 : Launch Zeppelin\*\*

Click on launch button and user will be navigated to Zeppelin home page.

Points to note:

(The pop-up may be blocked in browser configuration. Click on the red pop up blocker icon and select \*\*Always allow pop-up from http://xxx.xxx.xxx.xxx \*\* and click on \*\*Done\*\* button.)

![Figure](https://github.com/CiscoDevNet/data-dev-learning-labs/blob/master/labs/DATA%20EXPLORATION%2C%20ANALYSIS%20AND%20VISUALIZATION%20USING%20APACHE%20ZEPPELIN/assets/popUpBlockerAllowed1.jpeg?raw=true)

![Figure](https://github.com/CiscoDevNet/data-dev-learning-labs/blob/master/labs/DATA%20EXPLORATION%2C%20ANALYSIS%20AND%20VISUALIZATION%20USING%20APACHE%20ZEPPELIN/assets/popUpBlockerAllowed2.jpeg?raw=true)

On enabling the pop-up, Zeppelin would open in a new window taking the user to Zeppelin home page as shown in screenshot below.

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR5.jpeg?raw=true)

The following notebooks are provided in zeppelin:

1. Analysis and modeling
2. Wafer dbn(Deep Belief Network) & xqboost
3. Wafer physical model

### \*\*Step 5 : Select Zeppelin notebook – “Analysis and Modeling”\*\*

The analysis is done in Pyspark, Spark ML and python. We use physical model, Linear regression Classifier, DBN and XGBoost to model CMP data.

\* Select pre-created notebook - "Analysis and modeling" from home page as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR6.jpeg?raw=true)

\* The Zeppelin interactive,development and visualisation environment is now ready for use. The pre-created notebook has all the codes required for executing this learning lab.

![Figure](<https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR7.jpeg?raw=true)>

### \*\*Step 6 : Execute the code in notebook – “Analysis and Modeling”\*\*

We are using code written in Pyspark, Spark ML and python.

Following functions are being done by the Code :

1. Read and analyze CMP data
2. Build and evaluate a linear regression model against CMP data
3. Compare the results between Statistical approach and physics-based modeling approach.

Execute the code by clicking on the icon highlighted as shown in screenshot below:.

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR7.jpeg?raw=true)

On successful execution of the code, the icon changes to "Finished" as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR8.jpeg?raw=true)

The code is as follows:

1. **Load Dependencies**

```

%dep

z.reset()

z.addRepo("Spark Packages Repo").url("http://dl.bintray.com/spark-packages/maven")

z.load("com.databricks:spark-csv\_2.10:1.2.0")

```

1. Load Raw Data

```

%pyspark

import os

train\_df = (sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true")

.load(os.environ['HDFS\_ROOT'] + "/cmp-train-all.csv"))

train\_rm\_df = (sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true")

.load(os.environ['HDFS\_ROOT'] + "/cmp-training-removalrate.csv"))

test\_df=(sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true")

.load(os.environ['HDFS\_ROOT'] + "/CMP\_test\_data.csv"))

test\_rm\_df=(sqlContext.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true")

.load(os.environ['HDFS\_ROOT'] + "/orig\_CMP-test-removalrate.csv"))

URI = sc.\_gateway.jvm.java.net.URI

Path = sc.\_gateway.jvm.org.apache.hadoop.fs.Path

FileSystem = sc.\_gateway.jvm.org.apache.hadoop.fs.FileSystem

Configuration = sc.\_gateway.jvm.org.apache.hadoop.conf.Configuration

IOUtils = sc.\_gateway.jvm.org.apache.hadoop.io.IOUtils

dbn\_a = os.environ['HDFS\_ROOT'] + "/dbn\_a.pkl" #'hdfs://172.16.1.11:8020/user/jiaocheng/wafer\_models/dbn\_a.pkl'

dbn\_b = os.environ['HDFS\_ROOT'] + "/dbn\_b.pkl" #'hdfs://172.16.1.11:8020/user/jiaocheng/wafer\_models/dbn\_b.pkl'

xgb\_a = os.environ['HDFS\_ROOT'] + "/xgb\_a.pkl" #'hdfs://172.16.1.11:8020/user/jiaocheng/wafer\_models/xgb\_a.pkl'

xgb\_b = os.environ['HDFS\_ROOT'] + "/xgb\_b.pkl" #'hdfs://172.16.1.11:8020/user/jiaocheng/wafer\_models/xgb\_b.pkl'

xgb\_lib = os.environ['HDFS\_ROOT'] + "/xgb\_lib.zip" #'hdfs://172.16.1.11:8020/user/jiaocheng/python\_lib/xgb\_lib.zip'

#copy data and library files from HDFS to local file system

def save\_to\_local(hdfs\_path):

local\_path = '/tmp/'

basename = os.path.basename(hdfs\_path)

if os.path.exists(local\_path + basename):

print(local\_path + basename + ' has existed')

return

fs = FileSystem.get(URI('hdfs://172.16.1.11:8020'), Configuration())

local\_fs = FileSystem.getLocal(Configuration())

fin = fs.open(Path(hdfs\_path))

fout = local\_fs.create(Path(local\_path + basename))

IOUtils.copyBytes(fin, fout, 4096, True)

print(local\_path + basename + ' has done')

save\_to\_local(dbn\_a)

save\_to\_local(dbn\_b)

save\_to\_local(xgb\_a)

save\_to\_local(xgb\_b)

save\_to\_local(xgb\_lib)

save\_to\_local(os.environ['HDFS\_ROOT'] + "/cmp-train-all.csv")

save\_to\_local(os.environ['HDFS\_ROOT'] + "/cmp-training-removalrate.csv")

save\_to\_local(os.environ['HDFS\_ROOT'] + "/CMP\_test\_data.csv")

save\_to\_local(os.environ['HDFS\_ROOT'] + "/orig\_CMP-test-removalrate.csv")

save\_to\_local(os.environ['HDFS\_ROOT'] + "/dbn\_lib.zip")

```

1. **Description of time series data:**

%pyspark

new\_train=train\_df.na.drop()

z.show(new\_train.describe())

```

1. **Description of Average MRR data:**

```

%pyspark

new\_train\_rm=train\_rm\_df.na.drop()

z.show(new\_train\_rm.describe())

```

1. **Remove unnecessary features:**

```

%pyspark

new\_train=new\_train.drop('machine\_id').drop('machine\_data')

new\_test=test\_df.drop('machine\_id').drop('machine\_data')

print "data schema after removing unnecessary features"

new\_test.printSchema()

```

1. **Understand Data:**

* **Stage – wafer count**

```

%pyspark

from pyspark.sql.functions import countDistinct

z.show(new\_train\_rm.groupBy('stage').agg(countDistinct('wafer\_id').alias('wafer\_cnt')))

```

* **Chamber – wafer count**

```

%pyspark

z.show(new\_train.groupBy('chamber').agg(countDistinct('wafer\_id').alias('wafer\_cnt')).orderBy('chamber'))

```

* **Stage – Chamber count**

```

%pyspark

z.show(new\_train.groupBy('stage','chamber').agg(countDistinct('wafer\_id').alias('wafer\_cnt')).orderBy('stage','chamber'))

```

1. **Drop outliers**

```

%pyspark

z.show(new\_train\_rm.filter(new\_train\_rm['avg\_removal\_rate']>170))

train\_rm1=new\_train\_rm.filter(new\_train\_rm['avg\_removal\_rate']<170)

```

1. **MRR difference between stage A and Stage B**

```

%pyspark

from pyspark.sql import functions as F

z.show(train\_rm1.groupBy('stage').agg(F.min("avg\_removal\_rate"), F.avg('avg\_removal\_rate'),F.max('avg\_removal\_rate')))

```

1. **Join Average MRR to average features**

```

%pyspark

new\_avg\_train=new\_train.groupBy(['wafer\_id', 'stage']).avg().drop('avg(wafer\_id)').drop('avg(timestamp)')

new\_avg\_train\_rm=train\_rm1.groupBy(['wafer\_id', 'stage']).avg().drop('avg(wafer\_id)')

new\_avg\_test=new\_test.groupBy(['wafer\_id', 'stage']).avg().drop('avg(wafer\_id)').drop('avg(timestamp)').drop('avg()')

new\_avg\_test\_rm=test\_rm\_df.groupBy(['wafer\_id', 'stage']).avg().drop('avg(wafer\_id)')

df=new\_avg\_train.join(new\_avg\_train\_rm, [new\_avg\_train.wafer\_id==new\_avg\_train\_rm.wafer\_id,new\_avg\_train.stage==new\_avg\_train\_rm.stage], 'inner').drop(new\_avg\_train\_rm.wafer\_id).drop(new\_avg\_train\_rm.stage)

df\_t=new\_avg\_test\_rm.join(new\_avg\_test, [new\_avg\_test.wafer\_id==new\_avg\_test\_rm.wafer\_id, new\_avg\_test.stage==new\_avg\_test\_rm.stage]).drop(new\_avg\_test\_rm.wafer\_id).drop(new\_avg\_test\_rm.stage)

z.show(df.describe())

```

1. **Chamber vs MRR**

```

%pyspark

z.show(df.select('avg(avg\_removal\_rate)','avg(chamber)','stage'))

```

1. **Retainer Ring Pressure vs MRR**

```

%pyspark

z.show(df.select('avg(avg\_removal\_rate)','avg(retainer\_ring\_pressure)','stage'))

```

1. **Usage of Backing Film vs MRR**

```

%pyspark

z.show(df.select('avg(avg\_removal\_rate)','avg(usage\_of\_backing\_film)','stage'))

```

1. **Train model - linear regression**
   * Logarithm transformation on training & test dataset

```

%pyspark

from pyspark.sql.functions import log2

df1=df

n=len(df.columns)

for names in df1.columns[2:]:

df1=df.withColumn(names,log2(df[names]+1))

df2=df1.na.drop()

df1\_t=df\_t

for names in df.columns[2:]:

df1\_t=df\_t.withColumn(names,log2(df\_t[names]+1))

df2\_t=df1\_t.na.drop()

```

* + Show log transformed training data

```

%pyspark

z.show(df2.describe())

```

* + Show log transformed test data

```

%pyspark

z.show(df\_t.describe())

```

1. **Split training data into Group I (i.e., MRR<7) and Group II (i.e., MRR>=7)**

```

%pyspark

from pyspark.ml.feature import StringIndexer

from pyspark.mllib.regression import LabeledPoint

import pyspark.ml.regression as lm

# transform categorical feature 'stage' into numeric

indexer = StringIndexer(inputCol='stage', outputCol='stageIndex')

indexed = indexer.fit(df2).transform(df2)

indexed\_t = indexer.fit(df2\_t).transform(df2\_t)

# build labelpoint features

#train= indexed.rdd.map(lambda r:LabeledPoint(r[n-1],r[2:(n-2)]+r[n:n])).toDF(["features","label"])

train1= indexed.filter(indexed['avg(AVG\_REMOVAL\_RATE)']<7).rdd.map(lambda r:LabeledPoint(r[n-1],r[2:(n-2)]+r[n:n])).toDF(["features","label"])

train2=indexed.filter(indexed['avg(AVG\_REMOVAL\_RATE)']>=7).rdd.map(lambda r:LabeledPoint(r[n-1],r[2:(n-2)]+r[n:n])).toDF(["features","label"])

test=indexed\_t.rdd.map(lambda r:LabeledPoint(r[0],r[3:(n-1)]+r[n:n])).toDF(["features","label"])

```

1. **Train two linear regression models on Group I and II**

```

%pyspark

lr = lm.LinearRegression(featuresCol = 'features', labelCol = 'label', maxIter=10, regParam=0.3)

#lrModel=lr.fit(train)

lrModel1=lr.fit(train1)

lrModel2=lr.fit(train2)

```

1. **Predict all test data by Model 1**

```

%pyspark

from pyspark.sql.functions import pow

Pred=lrModel1.transform(test).select('label','prediction','features')

z.show(Pred)

```

1. **Predict Group 1 test data by Model 1**

```

%pyspark

from pyspark.sql.functions import pow

# split test data into Group I and II

# splitting value 6.1 is based on the observation of model 1 prediction

test1=Pred.filter(Pred.prediction>6.1).select('label','features')

test2=Pred.filter(Pred.prediction<6.1).select('label','features')

# predict Group I by model1

Pred1=lrModel1.transform(test1).select('label','prediction','features')

for names in Pred1.columns[0:2]:

Pred1=Pred1.withColumn(names,pow(2,Pred1[names])-1)

z.show(Pred1)

```

1. **Predict Group 2 test data by model 2**

```

%pyspark

from pyspark.sql.functions import pow

Pred2=lrModel2.transform(test2).select('label','prediction','features')

for names in Pred2.columns[0:2]:

Pred2=Pred2.withColumn(names,pow(2,Pred2[names])-1)

z.show(Pred2)

```

1. **Coefficients of Linear model 1 and model 2**

```

%pyspark

import numpy as np

names=indexed.columns

names=['intercept']+names[2:(n-2)]+names[n:n]

value=np.abs(np.append(lrModel1.weights,lrModel1.intercept))\*10000

value=value.tolist()

l = [value]

coe1=sqlContext.createDataFrame(l,names)

z.show(coe1)

value2=np.abs(np.append(lrModel2.weights,lrModel2.intercept))\*10000

value2=value2.tolist()

l2 = [value2]

coe2=sqlContext.createDataFrame(l2,names)

z.show(coe2)

#print("Coefficients of model 1: %s" % str(lrModel1.weights))

#print("Intercept of model 1: %s" % str(lrModel1.intercept))

#print("Coefficients of model 2: %s" % str(lrModel1.weights))

#print("Intercept of model 2: %s" % str(lrModel1.intercept))

# Summarize the model over the training set and print out some metrics

#trainingSummary = lrModel.summary

#print("numIterations: %d" % trainingSummary.totalIterations)

#print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))

#trainingSummary.residuals.show()

#print("RMSE: %f" % trainingSummary.rootMeanSquaredError)

#print("r2: %f" % trainingSummary.r2)

```

1. **RMSE**

```

%pyspark

Pre=Pred1.unionAll(Pred2)

pred=Pre.toPandas().to\_csv('/tmp/pred.csv')

RMSE=Pre.selectExpr("((label-prediction)\*(label-prediction)) as SE").groupBy().agg(F.avg('SE').alias('MSE'))

z.show(RMSE.groupBy('MSE').agg(F.sqrt('MSE').alias('RMSE')))

```

1. **Prediction vs Ground Truth**

```

%python

import pandas

import matplotlib.pyplot as plt

import io

from numpy import \*

import pandas as pd

pred=pandas.read\_csv('/tmp/pred.csv')

x=pred.label

y=pred.prediction

line = arange(0,300)

plt.plot(line, line, '-')

plt.plot(x, y, 'o')

plt.xlim(50, 200)

plt.ylim(50, 200)

plt.suptitle('Quantitative comparison between predicted\n and measured MRR distributions',fontsize=15, fontweight='bold')

plt.xlabel('Measured average MRR [nm/min]')

plt.ylabel('Predicted average MRR [nm/min]')

img1 = io.BytesIO() #Create StringIO object which named img

plt.savefig(img1, format='svg') #Save the figure in svg format

img1.seek(0) #Get the memory of the figure

print("%html <div style='width:500pt'>"+img1.getvalue() + "</div>")

img1.close()

plt.close()

```

### \*\*Step 7 : View output\*\*

In this step, we have built and evaluated a linear regression model against CMP data.On successful execution of code, you can view the ouput as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR9.jpeg?raw=true)

### \*\*Step 8 : Select Zeppelin notebook – “Wafer dbn(Deep Belief Network) & xqboost”\*\*

We are using code written in Pyspark, Spark ML and python.

Following functions are being done by the Code :

1. Model using Deep Belief Network
2. Model using extreme gradient boosting
3. Compare the results

Execute the code by clicking on the icon highlighted as shown in screenshot below:.

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR10.jpeg?raw=true)

On successful execution of the code, the icon changes to "Finished" as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR11.jpeg?raw=true)

The code is as follows:

1. **Prepare Data:**

* **Make packages directory**

```

%sh

mkdir /tmp/python\_lib

```

* **Extract related packages from zip file**

```

%python

import zipfile

zip\_ref = zipfile.ZipFile('/tmp/xgb\_lib.zip')

zip\_ref.extractall('/tmp/python\_lib/')

zip\_ref.close()

zip\_ref = zipfile.ZipFile('/tmp/dbn\_lib.zip')

zip\_ref.extractall('/tmp/python\_lib/')

zip\_ref.close()

```

* **Python package needs \_\_init\_\_.py file**

```

%sh

touch /tmp/python\_lib/dbn\_raw/\_\_init\_\_.py

```

1. **Model Deep Belief Network**

```

%python

from \_\_future\_\_ import print\_function, division

import sys

sys.path.insert(0, '/tmp/python\_lib')

import numpy as np

import pickle

from sklearn.metrics.regression import mean\_squared\_error

from dbn\_raw.models\_raw import SupervisedDBNRegression

from math import sqrt

from sklearn.preprocessing import MinMaxScaler

import pandas as pd

def normalize(X\_train, X\_test):

scaler = MinMaxScaler(copy=False)

scaler.fit\_transform(X\_train)

scaler.transform(X\_test)

def get\_data(x\_path, y\_path):

df\_x = pd.read\_csv(x\_path)

df\_y = pd.read\_csv(y\_path)

df\_avg = df\_x.groupby(['WAFER\_ID','STAGE']).mean().reset\_index()

df\_merge = df\_avg.merge(df\_y, on = ['WAFER\_ID', 'STAGE'])

cols = ['USAGE\_OF\_BACKING\_FILM',

'USAGE\_OF\_DRESSER','USAGE\_OF\_POLISHING\_TABLE',

'USAGE\_OF\_DRESSER\_TABLE','USAGE\_OF\_MEMBRANE','USAGE\_OF\_PRESSURIZED\_SHEET',

'PRESSURIZED\_CHAMBER\_PRESSURE','MAIN\_OUTER\_AIR\_BAG\_PRESSURE',

'CENTER\_AIR\_BAG\_PRESSURE','RETAINER\_RING\_PRESSURE','RIPPLE\_AIR\_BAG\_PRESSURE',

'SLURRY\_FLOW\_LINE\_A','SLURRY\_FLOW\_LINE\_B',

'SLURRY\_FLOW\_LINE\_C','WAFER\_ROTATION',

'STAGE\_ROTATION','HEAD\_ROTATION','EDGE\_AIR\_BAG\_PRESSURE','AVG\_REMOVAL\_RATE']

a\_idx = df\_merge[df\_merge.AVG\_REMOVAL\_RATE <= 100].index.tolist()

b\_idx = df\_merge[(df\_merge.AVG\_REMOVAL\_RATE >= 140)&(df\_merge.AVG\_REMOVAL\_RATE <= 165)].index.tolist()

data = np.array(df\_merge[cols])

a = data[a\_idx]

b = data[b\_idx]

return a[:,0:-1], a[:,-1],b[:,0:-1], b[:,-1]

X\_train\_a,Y\_train\_a,X\_train\_b,Y\_train\_b = get\_data('/tmp/cmp-train-all.csv', '/tmp/cmp-training-removalrate.csv')

X\_test\_a,Y\_test\_a,X\_test\_b,Y\_test\_b = get\_data('/tmp/CMP\_test\_data.csv', '/tmp/orig\_CMP-test-removalrate.csv')

normalize(X\_train\_a, X\_test\_a)

normalize(X\_train\_b, X\_test\_b)

regressor\_a = SupervisedDBNRegression.load('/tmp/dbn\_a.pkl')

Y\_pred\_a = regressor\_a.predict(X\_test\_a)

regressor\_b = SupervisedDBNRegression.load('/tmp/dbn\_b.pkl')

Y\_pred\_b = regressor\_b.predict(X\_test\_b)

print('Done.\nRMSE a: %f' % sqrt(mean\_squared\_error(Y\_test\_a, Y\_pred\_a)))

print('Done.\nRMSE b: %f' % sqrt(mean\_squared\_error(Y\_test\_b, Y\_pred\_b)))

overall\_rmse = sqrt((mean\_squared\_error(Y\_test\_a, Y\_pred\_a) \* len(Y\_test\_a) + mean\_squared\_error(Y\_test\_b, Y\_pred\_b) \* len(Y\_test\_b)) / (len(Y\_test\_a) + len(Y\_test\_b)))

print('overall\_rmse: %f' % overall\_rmse)

```

1. **Deep Belief Network Model Results:**

```

%python

import matplotlib.pyplot as plt

import io

x = np.concatenate((Y\_test\_a, Y\_test\_b), axis=0)

y = np.concatenate((Y\_pred\_a, Y\_pred\_b), axis=0)

line = np.arange(0,300)

plt.plot(line, line, '-')

plt.plot(x, y, 'o')

plt.xlim(50, 200)

plt.ylim(50, 200)

plt.suptitle('Quantitative comparison between predicted\n and measured MRR distributions',fontsize=15, fontweight='bold')

plt.xlabel('Measured average MRR [nm/min]')

plt.ylabel('Predicted average MRR [nm/min]')

img1 = io.BytesIO() #Create StringIO object which named img

plt.savefig(img1, format='svg') #Save the figure in svg format

img1.seek(0) #Get the memory of the figure

print("%html <div style='width:500pt'>" + img1.getvalue() + "</div>")

img1.close()

plt.close()

```

1. **Model of Extreme Gradient Boosting**

```

%python

import xgboost as xgb

#regressor\_a = pickle.load(open("/tmp/xgb\_a.pkl", "rb"))

#regressor\_b = pickle.load(open("/tmp/xgb\_b.pkl", "rb"))

regressor\_a = xgb.XGBRegressor(max\_depth=6, n\_estimators=1000)

regressor\_a.fit(X\_train\_a, Y\_train\_a)

regressor\_b = xgb.XGBRegressor(max\_depth=6, n\_estimators=1000)

regressor\_b.fit(X\_train\_b, Y\_train\_b)

Y\_pred\_a = regressor\_a.predict(X\_test\_a)

Y\_pred\_b = regressor\_b.predict(X\_test\_b)

print('Done.\nRMSE a: %f' % sqrt(mean\_squared\_error(Y\_test\_a, Y\_pred\_a)))

print('Done.\nRMSE b: %f' % sqrt(mean\_squared\_error(Y\_test\_b, Y\_pred\_b)))

overall\_rmse = sqrt((mean\_squared\_error(Y\_test\_a, Y\_pred\_a) \* len(Y\_test\_a) + mean\_squared\_error(Y\_test\_b, Y\_pred\_b) \* len(Y\_test\_b)) / (len(Y\_test\_a) + len(Y\_test\_b)))

print('overall\_rmse: %f' % overall\_rmse)

```

1. **XGBoost Results**

```

%python

import matplotlib.pyplot as plt

import io

x = np.concatenate((Y\_test\_a, Y\_test\_b), axis=0)

y = np.concatenate((Y\_pred\_a, Y\_pred\_b), axis=0)

line = np.arange(0,300)

plt.plot(line, line, '-')

plt.plot(x, y, 'o')

plt.xlim(50, 200)

plt.ylim(50, 200)

plt.suptitle('Quantitative comparison between predicted\n and measured MRR distributions',fontsize=15, fontweight='bold')

plt.xlabel('Measured average MRR [nm/min]')

plt.ylabel('Predicted average MRR [nm/min]')

img1 = io.BytesIO() #Create StringIO object which named img

plt.savefig(img1, format='svg') #Save the figure in svg format

img1.seek(0) #Get the memory of the figure

print("%html <div style='width:500pt'>" + img1.getvalue() + "</div>")

img1.close()

plt.close()

```

1. **Features Importance**

```

%python

print('xx')

feature\_names = ['USAGE\_OF\_BACKING\_FILM',

'USAGE\_OF\_DRESSER','USAGE\_OF\_POLISHING\_TABLE',

'USAGE\_OF\_DRESSER\_TABLE','USAGE\_OF\_MEMBRANE','USAGE\_OF\_PRESSURIZED\_SHEET',

'PRESSURIZED\_CHAMBER\_PRESSURE','MAIN\_OUTER\_AIR\_BAG\_PRESSURE',

'CENTER\_AIR\_BAG\_PRESSURE','RETAINER\_RING\_PRESSURE','RIPPLE\_AIR\_BAG\_PRESSURE',

'SLURRY\_FLOW\_LINE\_A','SLURRY\_FLOW\_LINE\_B',

'SLURRY\_FLOW\_LINE\_C','WAFER\_ROTATION',

'STAGE\_ROTATION','HEAD\_ROTATION','EDGE\_AIR\_BAG\_PRESSURE']

feature\_names = np.array(feature\_names)

feature\_importance\_a = regressor\_a.feature\_importances\_

feature\_importance\_b = regressor\_b.feature\_importances\_

feature\_importance\_a = 100.0 \* (feature\_importance\_a / feature\_importance\_a.max())

feature\_importance\_b = 100.0 \* (feature\_importance\_b / feature\_importance\_b.max())

sorted\_idx\_a = np.argsort(feature\_importance\_a)

sorted\_idx\_b = np.argsort(feature\_importance\_b)

pos = np.arange(len(feature\_names)) + .5

plt.figure(figsize=(15,9))

plt.subplot(1, 2, 1)

plt.barh(pos, feature\_importance\_a[sorted\_idx\_a], 0.5, align='center')

plt.yticks(pos, feature\_names[sorted\_idx\_a])

plt.xlim(0, 105)

plt.xlabel('Relative Importance')

plt.title('Group I Variable Importance')

plt.subplot(1, 2, 2)

plt.barh(pos, feature\_importance\_b[sorted\_idx\_b], 0.5, align='center')

plt.yticks(pos, feature\_names[sorted\_idx\_b])

plt.xlim(0, 105)

plt.xlabel('Relative Importance')

plt.title('Group II Variable Importance')

plt.tight\_layout()

img1 = io.BytesIO() #Create StringIO object which named img

plt.savefig(img1, format='svg') #Save the figure in svg format

img1.seek(0) #Get the memory of the figure

print("%html <div style='width:800pt'>" + img1.getvalue() + "</div>")

img1.close()

plt.close()

```

1. ### \*\*Step 8 : Select Zeppelin notebook – “Wafer physical model”\*\*

We are using code written in Pyspark, Spark ML and python.

Following functions are being done by the Code :

1. Train physical model
2. Compare the physical model results

Execute the code by clicking on the icon highlighted as shown in screenshot below:.

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR10.jpeg?raw=true)

On successful execution of the code, the icon changes to "Finished" as shown in screenshot below:

![Figure](https://raw.githubusercontent.com/CiscoDevNet/data-dev-learning-labs/master/labs/TENSORFLOW101/assets/images/MRR11.jpeg?raw=true)

The code is as follows:

1. **Train Physical Model:**

```

%python

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import division

import pandas as pd

from scipy.optimize import fsolve

import math

from numpy import \*

import matplotlib.pyplot as plt

import io

def get\_rmse(m, test\_data):

dev = []

for pr,rt,y in test\_data:

y\_pred = predict(m, (pr,rt))

dev.append((y, y\_pred))

#print('%f\t%f' % (y, y\_pred))

rmse = estimateVal(dev)

print('RMSE: %f' % rmse)

dev = array(dev)

return dev[:,0], dev[:,1], rmse

#Preston equation: MRR = K\*(P\*\*a)(V\*\*b)

def get\_data2(x\_path, y\_path):

df\_train = pd.read\_csv(x\_path)

df\_rm = pd.read\_csv(y\_path)

df\_train['PRESSURE'] = (df\_train.PRESSURIZED\_CHAMBER\_PRESSURE + df\_train.MAIN\_OUTER\_AIR\_BAG\_PRESSURE + df\_train.CENTER\_AIR\_BAG\_PRESSURE

+ df\_train.RETAINER\_RING\_PRESSURE

+ df\_train.RIPPLE\_AIR\_BAG\_PRESSURE + df\_train.EDGE\_AIR\_BAG\_PRESSURE )

df\_train['ROTATION'] = df\_train.WAFER\_ROTATION + df\_train.STAGE\_ROTATION

df\_avg = df\_train.groupby(['WAFER\_ID','STAGE']).agg({'PRESSURE': 'mean','ROTATION': 'mean'}).reset\_index()

df\_merge = df\_avg.merge(df\_rm, on = ['WAFER\_ID', 'STAGE'])

cols = ['PRESSURE', 'ROTATION', 'AVG\_REMOVAL\_RATE']

a\_idx = df\_merge[df\_merge.AVG\_REMOVAL\_RATE <= 100].index.tolist()

# b\_idx = df\_merge[(df\_merge.AVG\_REMOVAL\_RATE >= 140)&(df\_merge.AVG\_REMOVAL\_RATE <= 165)&

# (df\_merge.PRESSURE != 0)].index.tolist()

b\_idx = df\_merge[(df\_merge.AVG\_REMOVAL\_RATE >= 140)&(df\_merge.AVG\_REMOVAL\_RATE <= 165)].index.tolist()

data = array(df\_merge[cols])

a = data[a\_idx]

b = data[b\_idx]

return a, b

def myFunction(z, param):

k, a, b = z

#func.

f = empty((3))

i = 0

for (x,y,z) in param:

f[i] = k \* math.pow(x, a) \* math.pow(y, b) - z

i = i + 1

return f

def build\_model2(dataList):

kArray = []

aArray = []

bArray = []

zGuess = array([1,1,1])

tupleList = []

i = 0

for arr in dataList:

if 0 == int(arr[0]) or 0 == int(arr[1]):

continue

i = i + 1

tupleList.append(arr)

if i%3 == 0:

k, a, b = fsolve(myFunction, zGuess, tupleList)

if ( k >= 150) and ( k<= 700):

kArray.append(k)

aArray.append(a)

bArray.append(b)

tupleList = []

kAvg = sum(kArray)/len(kArray)

aAvg = sum(aArray)/len(aArray)

bAvg = sum(bArray)/len(bArray)

return (kAvg, aAvg, bAvg)

def predict(m, d):

k, a, b = m

x, y = d

a = 0 if 0 == int(x) else a

b = 0 if 0 == int(y) else b

# if 0 == int(x) or 0 == int(y):

# return 0

return k \* math.pow(x, a) \* math.pow(y, b)

def estimateVal(tList):

sumVal = 0

for p, a in tList:

#print('p, a',p,a)

sumVal = sumVal + math.pow((p -a), 2)

#print('p, a, sum:',p,a,sumVal)

return math.sqrt(sumVal/len(tList))

train\_a, train\_b = get\_data2('/tmp/cmp-train-all.csv', '/tmp/cmp-training-removalrate.csv')

test\_a, test\_b = get\_data2('/tmp/CMP\_test\_data.csv', '/tmp/orig\_CMP-test-removalrate.csv')

m\_a = (356.63721473818919, -0.016890878358881443, -0.32380617498813929)

m\_b = (225.69876624236517, -0.03755748580984368, -0.016337749763835042)

#m = build\_model2(train\_b)

print('formula MRR = K\*(P\*\*a)(V\*\*b) group A coneffient (k, a, b) :', m\_a)

print('formula MRR = K\*(P\*\*a)(V\*\*b) group B coneffient (k, a, b) :', m\_b)

xa,ya,rmse\_a = get\_rmse(m\_a, test\_a)

xb,yb,rmse\_b = get\_rmse(m\_b, test\_b)

overall\_rmse = math.sqrt(((rmse\_a\*\*2) \* len(test\_a) + (rmse\_b\*\*2) \* len(test\_b)) / (len(test\_a) + len(test\_b)))

print('overall\_rmse: %f' % overall\_rmse)

```

1. Physical Model Results

%python

x = concatenate((xa, xb), axis=0)

y = concatenate((ya, yb), axis=0)

line = arange(0,300)

plt.plot(line, line, '-')

plt.plot(x, y, 'o')

plt.xlim(50, 300)

plt.ylim(50, 300)

plt.suptitle('Quantitative comparison between predicted\n and measured MRR distributions',fontsize=15, fontweight='bold')

plt.xlabel('Measured average MRR [nm/min]')

plt.ylabel('Predicted average MRR [nm/min]')

img1 = io.BytesIO() #Create StringIO object which named img

plt.savefig(img1, format='svg') #Save the figure in svg format

img1.seek(0) #Get the memory of the figure

print("%html <div style='width:500pt'>" + img1.getvalue() + "</div>")

img1.close()

plt.close()

## LESSONS LEARNT

1. How to write and execute python code using Apache Zeppelin.
2. How to predict MRR (Material Removal Rate) of wafers by modelling CMP(Chemical Mechanical Planarization).

# \*\*Congratulations! You have successfully completed the Learning Lab!\*\*