Machine Learning with the Equipment Failure Prediction.

Overview

Now I will take the selected/pre-processed/imputed datasets and feed this into different ML models by using cross validation.

Business Problem in machine learning terms:

Given the data points with their respective features, use classification to find out whether the data points belong to surface failure or downhole failure.

Metric to be used:

F2 Score : ((1 + (2)^2) * Precision * Recall) / (((2)^2 * Precision) + Recall)

Datasets:

3 datasets :

1)Median imputed Median dataset.
2)Median imputed Adasyn dataset.
3)Median imputed Smotetomek dataset.

I will feed these 3 datasets into MultiLayered Perceptron.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from sklearn import metrics
from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import fbeta_score
```

In [3]:
```
import warnings
warnings.simplefilter(action="ignore", category=UserWarning)
```

In [4]:
```
import datetime
import tensorflow as tf
from tensorflow.keras import models, layers
from tensorflow.python.keras.utils import tf_utils
from tensorflow.keras.models import Model,Sequential
from tensorflow.python.keras.utils.io_utils import path_to_string
```

In [5]:
```
import os
os.cpu_count()
```

Out[5]: 8

Get the data

In [6]:
```
new_standard_train_median_all_features_df=pd.read_pickle("new_standard_train_median_all_features_df.pickle")
y_train_median_all_features=pd.read_pickle("y_train_median_all_features.pickle")
new_standard_test_median_all_features_df=pd.read_pickle("new_standard_test_median_all_features_df.pickle")
y_test_median_all_features=pd.read_pickle("y_test_median_all_features.pickle")

new_x_adasyn_df=pd.read_pickle("new_x_adasyn_df.pickle")
y_adasyn=pd.read_pickle("y_adasyn.pickle")
new_standard_test_adasyn_all_features_df=pd.read_pickle("new_standard_test_adasyn_all_features_df.pickle")
y_test_adasyn_all_features=pd.read_pickle("y_test_adasyn_all_features.pickle")

new_x_smotetomek_df=pd.read_pickle("new_x_smotetomek_df.pickle")
y_smotetomek=pd.read_pickle("y_smotetomek.pickle")
new_standard_test_smotetomek_all_features_df=pd.read_pickle("new_standard_test_smotetomek_all_features_df.pi
y_test_smotetomek_all_features=pd.read_pickle("y_test_smotetomek_all_features.pickle")
```

I am taking 20% of the data as cv and the rest I will use it as train data.

In [7]:
```
from sklearn.model_selection import train_test_split
x_train, x_cv, y_train, y_cv = train_test_split(
    new_standard_train_median_all_features_df, y_train_median_all_features,
    test_size=0.20, stratify=y_train_median_all_features)
```

In [8]:
```
input_dim=new_standard_train_median_all_features_df.shape[1]
input_dim
```

Out[8]: 127

In [9]:
```
#We need to convert our pandas data into numpy before giving it to Tensorflow.
x_train=x_train.to_numpy()
x_cv=x_cv.to_numpy()
```

In [10]:
```
#pd.get_dummies() converts a list of categorical variables into one hot encoded variables.
#We need our categorical variables to be in the form of one hot encoded variables.
y_train = pd.get_dummies(y_train).values
y_cv = pd.get_dummies(y_cv).values
```

In [11]:
```
print("type(x_train) : ",type(x_train))
print("type(y_train) : ",type(y_train))
print("type(x_cv) : ",type(x_cv))
print("type(y_cv) : ",type(y_cv))
print("x_train.shape : ",x_train.shape)
print("y_train.shape : ",y_train.shape)
print("x_cv.shape : ",x_cv.shape)
print("y_cv.shape : ",y_cv.shape)
```

```
type(x_train) :  <class 'numpy.ndarray'>
type(y_train) :  <class 'numpy.ndarray'>
type(x_cv) :  <class 'numpy.ndarray'>
type(y_cv) :  <class 'numpy.ndarray'>
x_train.shape :  (38400, 127)
y_train.shape :  (38400, 2)
x_cv.shape :  (9600, 127)
y_cv.shape :  (9600, 2)
```

MLP

In [7]:
```
def fit_the_model(x_train,y_train,x_cv,y_cv,input_dim,epochs,batch_size,data_types,unit1,dropout1,unit2,drop

    tf.keras.backend.clear_session()

    try:
        del model
        del Model
        print("Old Model deleted.")
    except(NameError):
        print("No Model yet.")
        pass

    tf.keras.backend.clear_session()

    #For tensorflow.keras dropout rate, is a value between 0 and 1. Fraction of the input units to drop.
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(unit1, input_shape=(input_dim,),activation=act),
        tf.keras.layers.Dropout(dropout1),
        tf.keras.layers.Dense(unit2,activation=act),
        tf.keras.layers.Dropout(dropout2),
        tf.keras.layers.Dense(unit3,activation=act),
        tf.keras.layers.Dropout(dropout3),
        tf.keras.layers.Dense(unit4,activation=act),
        tf.keras.layers.Dropout(dropout4),
        tf.keras.layers.Dense(unit5,activation=act),
        tf.keras.layers.Dropout(dropout5),
        tf.keras.layers.Dense(2,activation='softmax')
    ])
    model.summary()


    #Setting a callback to save the best model weight value.
    checkpoint_filepath = str(data_types)+"_checkpoint_"+str(unit1)+"_"\
        +str(dropout1)+"_"\
        +str(unit2)+"_"\
        +str(dropout2)+"_"\
        +str(unit3)+"_"\
        +str(dropout3)+"_"\
        +str(unit4)+"_"\
        +str(dropout4)+"_"\
        +str(unit5)+"_"\
```

```
        +str(dropout5)+"_"+str(act)+"_"+str(rate)+"/checkpoint_"+ datetime.datetime.now().strftime("%Y%m%d-%


    #Compiling the model.
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=rate), loss='categorical_crossentropy')

    #Training the model.
    model.fit(x_train,y_train,epochs=40,verbose=0,batch_size=batch_size,
              callbacks=[custom_Metrics(validation_data=(x_cv, y_cv),
                                        filepath=checkpoint_filepath,
                                        save_freq='epoch')])
    return model
```

In [8]:
```
#This custom_metrics_class will inherit the methods of the Base class Callback ==> Inheritance
#class DerivedClassName(modname.BaseClassName)
#class DerivedClassName(BaseClassName)
class custom_Metrics(tf.keras.callbacks.ModelCheckpoint,tf.keras.callbacks.Callback):
    def __init__(self,validation_data,**kwargs):

        super(custom_Metrics, self).__init__(**kwargs)

        self.validation_data=validation_data
        print("type(validation_data)",type(self.validation_data))
        print('validation_data[0]', self.validation_data[0].shape)
        print('validation_data[1]', self.validation_data[1].shape)

    def on_epoch_begin(self, epoch, logs=None):
        self._current_epoch = epoch

    def on_epoch_end(self, epoch, logs):
        val_targ = self.validation_data[1]
        val_predict = (np.asarray(self.model.predict(self.validation_data[0]))).round()

        f2_score=np.round(fbeta_score(val_targ[:,1], val_predict[:,1], beta=2,average="binary", pos_label=1)

        print(f'Epoch : {epoch} \t\t\t val_f2: {f2_score}')

        self.epochs_since_last_save += 1
        if self.save_freq == 'epoch':
            self.save_model(epoch=epoch, logs=logs)

    def save_model(self, epoch, logs={}):
        """This is a method used for saving the model, this method has been customised accordingly.
        (original function: def _save_model in tf.keras.callbacks.ModelCheckpoint)
        """

        logs = tf_utils.to_numpy_or_python_type(logs)

        filepath = self._get_file_path(epoch, logs)+"_Epoch_%03d"% (epoch)

        print('Epoch %03d: saving weights only to %s\n\n' % (epoch, filepath))
        self.model.save_weights(filepath, overwrite=True, options=self._options)
```

Trying with median based data.

In [14]:
```
epochs = 40
batch_size=1000
activations=[None,"relu","selu","elu"]
rates=[0.0001,0.001,0.01]
```

In [15]:
```
# Configuration of number of hidden units in a layer followed by the dropout rate for each layer.
l=[
[100,.5,50,.4,20,.3,10,.2,5,.1],
[200,.8,100,.4,50,.6,20,.2,5,.5],
[400,.8,200,.4,100,.6,50,.2,10,.5],
[400,.4,200,.6,100,.8,50,.4,10,.2],
[50,.5,40,.4,30,.3,20,.2,10,.1],
[200,.1,300,.2,400,.3,500,.4,600,.5]
]
```

In [ ]:
```
%%time
for config in l:
    for act in activations:
        for rate in rates:
            unit1=config[0]
            dropout1=config[1]
            unit2=config[2]
            dropout2=config[3]
            unit3=config[4]
            dropout3=config[5]
            unit4=config[6]
            dropout4=config[7]
            unit5=config[8]
            dropout5=config[9]

            model = fit_the_model(x_train,y_train,x_cv,y_cv,input_dim,epochs,
                                  batch_size,"median_data",
                                  unit1,dropout1,unit2,dropout2,unit3,dropout3,unit4,dropout4,unit5,dropout5
```

```
No Model yet.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               12800
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1020
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
```

Trying with adasyn based data.

In [12]:
```
from sklearn.model_selection import train_test_split
x_train, x_cv, y_train, y_cv = train_test_split(
    new_x_adasyn_df, y_adasyn,
    test_size=0.20, stratify=y_adasyn)
```

In [13]:
```
input_dim=new_x_adasyn_df.shape[1]
input_dim
```

Out[13]: 144

In [14]:
```
#We need to convert our pandas data into numpy before giving it to Tensorflow.
x_train=x_train.to_numpy()
x_cv=x_cv.to_numpy()
```

In [15]:
```
#pd.get_dummies() converts a list of categorical variables into one hot encoded variables.
#We need our categorical variables to be in the form of one hot encoded variables.
y_train = pd.get_dummies(y_train).values
y_cv = pd.get_dummies(y_cv).values
```

In [16]:
```
print("type(x_train) : ",type(x_train))
print("type(y_train) : ",type(y_train))
print("type(x_cv) : ",type(x_cv))
print("type(y_cv) : ",type(y_cv))
print("x_train.shape : ",x_train.shape)
print("y_train.shape : ",y_train.shape)
print("x_cv.shape : ",x_cv.shape)
print("y_cv.shape : ",y_cv.shape)
```

```
type(x_train) :  <class 'numpy.ndarray'>
type(y_train) :  <class 'numpy.ndarray'>
type(x_cv) :  <class 'numpy.ndarray'>
type(y_cv) :  <class 'numpy.ndarray'>
x_train.shape :  (66074, 144)
y_train.shape :  (66074, 2)
x_cv.shape :  (16519, 144)
y_cv.shape :  (16519, 2)
```

In [ ]:
```
%%time
for config in l:
    for act in activations:
        for rate in rates:
            unit1=config[0]
            dropout1=config[1]
            unit2=config[2]
            dropout2=config[3]
            unit3=config[4]
            dropout3=config[5]
            unit4=config[6]
            dropout4=config[7]
            unit5=config[8]
            dropout5=config[9]

            model = fit_the_model(x_train,y_train,x_cv,y_cv,input_dim,epochs,
                                  batch_size,"adasyn_data",
                                  unit1,dropout1,unit2,dropout2,unit3,dropout3,unit4,dropout4,unit5,dropout5
```

```
No Model yet.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               14500
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1020
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
```

Trying with smotetomek based data.

In [9]:
```
from sklearn.model_selection import train_test_split
x_train, x_cv, y_train, y_cv = train_test_split(
    new_x_smotetomek_df, y_smotetomek,
    test_size=0.20, stratify=y_smotetomek)
```

In [10]:
```
input_dim=new_x_smotetomek_df.shape[1]
input_dim
```

Out[10]: 117

In [11]:
```
#We need to convert our pandas data into numpy before giving it to Tensorflow.
x_train=x_train.to_numpy()
x_cv=x_cv.to_numpy()
```

In [12]:
```
#pd.get_dummies() converts a list of categorical variables into one hot encoded variables.
#We need our categorical variables to be in the form of one hot encoded variables.
y_train = pd.get_dummies(y_train).values
y_cv = pd.get_dummies(y_cv).values
```

In [13]:
```
print("type(x_train) : ",type(x_train))
print("type(y_train) : ",type(y_train))
print("type(x_cv) : ",type(x_cv))
print("type(y_cv) : ",type(y_cv))
print("x_train.shape : ",x_train.shape)
print("y_train.shape : ",y_train.shape)
print("x_cv.shape : ",x_cv.shape)
print("y_cv.shape : ",y_cv.shape)
```

```
type(x_train) :  <class 'numpy.ndarray'>
type(y_train) :  <class 'numpy.ndarray'>
type(x_cv) :  <class 'numpy.ndarray'>
type(y_cv) :  <class 'numpy.ndarray'>
x_train.shape :  (66078, 117)
y_train.shape :  (66078, 2)
x_cv.shape :  (16520, 117)
y_cv.shape :  (16520, 2)
```

In [ ]:
```
%%time
for config in l:
    for act in activations:
        for rate in rates:
            unit1=config[0]
            dropout1=config[1]
            unit2=config[2]
            dropout2=config[3]
            unit3=config[4]
            dropout3=config[5]
            unit4=config[6]
            dropout4=config[7]
            unit5=config[8]
            dropout5=config[9]

            model = fit_the_model(x_train,y_train,x_cv,y_cv,input_dim,epochs,
                                  batch_size,"smotetomek_data",
                                  unit1,dropout1,unit2,dropout2,unit3,dropout3,unit4,dropout4,unit5,dropout5
```

```
No Model yet.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               11800
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                1020
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                210
_________________________________________________________________
```

Trying with the highest median data weights

In [12]:
```
#median
np.sort([0.308511,0.379908,0.437428,0.496384,0.526499,0.562593,0.595813,0.635335,0.667921,0.684932,0.712058,
```

Out[12]:
```
array([0.910758, 0.895522, 0.892634, ..., 0.      , 0.      , 0.      ])
```

In [14]:
```
tf.keras.backend.clear_session()

try:
    del model
    del Model
    print("Old Model deleted.")
except(NameError):
    print("No Model yet.")
    pass

tf.keras.backend.clear_session()

#For tensorflow.keras dropout rate, is a value between 0 and 1. Fraction of the input units to drop.
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(200, input_shape=(127,),activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(300,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(400,activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(500,activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(600,activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2,activation='softmax')
])
model.summary()
```

```
No Model yet.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               25600
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               60300
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 400)               120400
_________________________________________________________________
dropout_2 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               200500
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 600)               300600
_________________________________________________________________
dropout_4 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 1202
=================================================================
Total params: 708,602
Trainable params: 708,602
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
# Getting the weight that got us the highest score
model.load_weights("median_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.001/checkpoint_202
```

Out[20]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f849fb79700>
```

In [21]:
```
x_test=new_standard_test_median_all_features_df.to_numpy()
y_test = pd.get_dummies(y_test_median_all_features).values
```

In [22]:
```
y_pred=np.round(model.predict(x_test))
```

In [23]:
```
f2_score=np.round(fbeta_score(y_test[:,1], y_pred[:,1], beta=2,average="binary",pos_label=1),6)
f2_score
```

Out[23]:
```
0.894584
```

In [14]:
```
# Utility function to print the confusion matrix
def confusion_matrix(Y_true, Y_pred):

    true = pd.Series(list(Y_true))
    predicted = pd.Series(list(Y_pred))

    return pd.crosstab(true, predicted, rownames=['True'], colnames=['Predicted'])
```

In [25]:
```
cm = confusion_matrix(y_test[:,1], y_pred[:,1])
cm
```

Out[25]:

Predicted	0.0	1.0
True		
0	11751	49
1	15	185

Trying with the highest adasyn data weights

In [34]:
```
#adasyn
np.sort([0.914496,0.924599,0.930197,0.932584,0.933419,0.934164,0.93466,0.934558,0.934519,0.935788,0.935768,0
```

Out[34]:
```
array([0.999831, 0.999831, 0.999802, ..., 0.      , 0.      , 0.      ])
```

In [35]:
```
tf.keras.backend.clear_session()

try:
    del model
    del Model
    print("Old Model deleted.")
except(NameError):
    print("No Model yet.")
    pass

tf.keras.backend.clear_session()

#For tensorflow.keras dropout rate, is a value between 0 and 1. Fraction of the input units to drop.
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(200, input_shape=(144,),activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(300,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(400,activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(500,activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(600,activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2,activation='softmax')
])
model.summary()
```

```
Old Model deleted.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               29000
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               60300
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
```

```
_________________________________________________________________
dense_2 (Dense)              (None, 400)               120400
_________________________________________________________________
dropout_2 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               200500
_________________________________________________________________
dropout_3 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_4 (Dense)              (None, 600)               300600
_________________________________________________________________
dropout_4 (Dropout)          (None, 600)               0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 1202
=================================================================
Total params: 712,002
Trainable params: 712,002
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
# Getting the weight that got us the highest score
model.load_weights("adasyn_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.001/checkpoint_202
```

Out[36]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fc26eefd340>
```

In [37]:
```
x_test=new_standard_test_adasyn_all_features_df.to_numpy()
y_test = pd.get_dummies(y_test_adasyn_all_features).values
```

In [38]:
```
y_pred=np.round(model.predict(x_test))
```

In [39]:
```
f2_score=np.round(fbeta_score(y_test[:,1], y_pred[:,1], beta=2,average="binary",pos_label=1),6)
f2_score
```

Out[39]:
```
0.895522
```

In [40]:
```
cm = confusion_matrix(y_test[:,1], y_pred[:,1])
cm
```

Out[40]:

Predicted	0.0	1.0
True		
0	11775	25
1	20	180

Trying with the highest smotetomek data weights

In [12]:
```
#smotetomek
np.sort([0.937062,0.945431,0.952316,0.955862,0.958897,0.961068,0.962165,0.962907,0.963394,0.964247,0.96489,0
```

Out[12]:
```
array([0.999689, 0.999661, 0.999661, ..., 0.739287, 0.724208, 0.      ])
```

In [14]:
```
tf.keras.backend.clear_session()

try:
    del model
    del Model
    print("Old Model deleted.")
except(NameError):
    print("No Model yet.")
    pass

tf.keras.backend.clear_session()

#For tensorflow.keras dropout rate, is a value between 0 and 1. Fraction of the input units to drop.
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(200, input_shape=(117,),activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(300,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(400,activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(500,activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(600,activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2,activation='softmax')
])
model.summary()
```

```
No Model yet.
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 200)               23600
_________________________________________________________________
dropout (Dropout)            (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 300)               60300
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0
```

```
dense_2 (Dense)              (None, 400)               120400
_________________________________________________________________
dropout_2 (Dropout)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               200500
```

In [16]:
```
# Getting the weight that got us the highest score
model.load_weights("smotetomek_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.001/checkpoint
```

Out[16]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7fc3f9a05eb0>
```

In [17]:
```
x_test=new_standard_test_smotetomek_all_features_df.to_numpy()
y_test = pd.get_dummies(y_test_smotetomek_all_features).values
```

In [18]:
```
y_pred=np.round(model.predict(x_test))
```

In [19]:
```
f2_score=np.round(fbeta_score(y_test[:,1], y_pred[:,1], beta=2,average="binary",pos_label=1),6)
f2_score
```

Out[19]:
```
0.910448
```

In [22]:
```
cm = confusion_matrix(y_test[:,1], y_pred[:,1])
cm
```

Out[22]:

Predicted	0.0	1.0
True		
0	11778	22
1	17	183

Observation :

I observe that I get the highest score on cross validation on adasyn data.

But on the test data , I get the highest score on test on smotetomek data.

But There are a few configurations like :
adasyn_data_checkpoint_400_0.8_200_0.4_100_0.6_50_0.2_10_0.5_selu_0.01/checkpoint_20201112-012718_Epoch_033,
smotetomek_data_checkpoint_400_0.4_200_0.6_100_0.8_50_0.4_10_0.2_elu_0.01/checkpoint_20201112-035913_Epoch_028,
smotetomek_data_checkpoint_400_0.4_200_0.6_100_0.8_50_0.4_10_0.2_elu_0.01/checkpoint_20201112-035913_Epoch_029,
smotetomek_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.01/checkpoint_20201112-041915_Epoch_024,
smotetomek_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.01/checkpoint_20201112-041915_Epoch_039,
smotetomek_data_checkpoint_400_0.4_200_0.6_100_0.8_50_0.4_10_0.2_selu_0.01/checkpoint_20201112-035540_Epoch_025,
smotetomek_data_checkpoint_400_0.4_200_0.6_100_0.8_50_0.4_10_0.2_elu_0.01/checkpoint_20201112-035913_Epoch_020
and many more that are giving me,
a perfect detection of downhole failures but at the cost of high misclassification of surface failures and a low f2 score.

In [20]:
```
try:
    del model
    del Model
    print("Old Model deleted.")
except(NameError):
    print("No Model yet.")
    pass
```

```
No Model yet.
```

In [21]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(200, input_shape=(117,),activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(300,activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(400,activation="relu"),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(500,activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(600,activation="relu"),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(2,activation='softmax')
])
```

In [23]:
```
# Getting the weight that got us the highest score
model.load_weights("smotetomek_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.01/checkpoint_
```

Out[23]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f6c001b43a0>
```

In [24]:
```
x_test=new_standard_test_smotetomek_all_features_df.to_numpy()
y_test = pd.get_dummies(y_test_smotetomek_all_features).values
```

In [25]:
```
x_test.shape
y_test.shape
```

Out[25]:
```
(12000, 2)
```

In [26]:
```
y_pred=np.round(model.predict(x_test))
```

Observe a low F2 Score

In [27]:
```
f2_score=np.round(fbeta_score(y_test[:,1], y_pred[:,1], beta=2,average="binary",pos_label=1),6)
f2_score
```

Out[27]:
```
0.52356
```

Observe below perfect classification of downhole failures, but a huge misclassification of surface failures.

In [29]:
```
cm = confusion_matrix(y_test[:,1], y_pred[:,1])
cm
```

Out[29]:

Predicted	0.0	1.0
True		
0	10890	910
1	0	200

Conclusion : I will get a good F2 score only when there are both low downhole and low surface failures misclassification. Both are needed to be low. If I concentrate on removing the downhole misclassifications, then I observe a tradeoff between the

downhole and surface misclassification.
I will tried running both adasyn test data and smotetomek test data but with 2 configurations(one favouring F2 score and one focussing on removing the downhole failures), and got the below results.

adasyn_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.001/checkpoint_20201112-020820_Epoch_023:
f2 score : 0.895522

Predicted	0.0	1.0
True		
0	11775	25
1	20	180

smotetomek_data_checkpoint_200_0.1_300_0.2_400_0.3_500_0.4_600_0.5_relu_0.01/checkpoint_20201112-041915_Epoch_039:
f2 score : 0.52356

Predicted	0.0	1.0
True		
0	10890	910
1	0	200