CODE

SMOTE:

```
import numpy as np
import pandas as pd

# import dataset
data=pd.read_csv(r"E:\origindata.csv", header=0)
data.name.value_counts ()
x=data [["Sand", "Silt", "Clay", "QZ", "pF", "theta", "n", "bulk density", "lambda"]]
y = data[["name"]]

# SMOTE
from imblearn.over_sampling import SMOTE
smo=SMOTE(sampling_strategy={7:279,8:279,9:279,10:279,11:279,12:279,13:279,1
4:279,15:279,16: 279,17279,18:279,19:279,20:279,21: 279},
        k_neighbors=2,
        random_state=99)
x_smo, y_smo = smo.fit_sample(x, y)
data_smote = pd.concat([y_smo, x_smo], axis=1)

#RUS
from imblearn.under_sampling import RandomUnderSampler
rus=RandomUnderSampler(sampling_strategy={1:279,2: 279,3: 279,4: 279,5: 279,6:
279}, random_state=99)
x_rus, y_rus = rus.fit_sample(x_smo, y_smo)
data_un_smote= pd.concat([y_rus,x_rus], axis=1)
data_un_smote.name.value_counts()

#output
data_un_smote.to_excel(2_279.xlsx)
```

Division of Training set, Validation set and Testing set (6:2:2)

```
import numpy as np
import pandas as pd
# Trainv set and Testing set
# import dataset
tpt_data = pd.read_csv(r"E:\ 2_279.csv", header=0)
tpt_data.drop_duplicates(inplace=True)
tpt_data =tpt_data.sample(len(tpt_data), random_state=0)
tpt_data.name.value_counts()

tpt_gbr = tpt_data.groupby("name")
tpt_gbr.groups
trainv_rate = 0.8
num_tup=np.array([279, 279, 279, 279, 279, 279, 279, 279, 279, 279, 279, 279, 279,
279,279, 279, 279, 279, 279, 279, 279])
num_trainv_tup = np.array([(int)(round(j * trainv_rate)) for j in num_tup])
num_test_tup = num_tup – num_trainv_tup
print(num_trainv_tup)
print(num_test_tup)

typicalNDict_training_validation = {1: num_trainv_tup[0], 2: num_trainv_tup[1], 3:
num_trainv_tup[2], 4: num_trainv_tup[3], 5: num_trainv_tup[4], 6: num_trainv_tup[5],
7: num_trainv_tup[6], 8: num_trainv_tup[7], 9: num_trainv_tup[8], 10:
num_trainv_tup[9], 11: num_trainv_tup[10], 12: num_trainv_tup[11], 13:
num_trainv_tup[12], 14: num_trainv_tup[13], 15: num_trainv_tup[14], 16:
num_trainv_tup[15], 17: num_trainv_tup[16], 18: num_trainv_tup[17], 19:
num_trainv_tup[18], 20: num_trainv_tup[19], 21: num_trainv_tup[20]}
typicalNDict_testing = {1: num_test_tup[0], 2: num_test_tup[1], 3: num_test_tup[2], 4:
num_test_tup[3], 5: num_test_tup[4], 6: num_test_tup[5], 7: num_test_tup[6], 8:
num_test_tup[7], 9: num_test_tup[8], 10: num_test_tup[9], 11: num_test_tup[10], 12:
num_test_tup[11], 13: num_test_tup[12], 14: num_test_tup[13], 15: num_test_tup[14],
```

```
16: num_test_tup[15], 17: num_test_tup[16], 18: num_test_tup[17], 19: num_test_tup[18], 20: num_test_tup[19], 21: num_test_tup[20]}

def typicalsamling(group, typicalNDict):
    name = group.name
    n = typicalNDict[name]
    return group.sample(n=n)

result_tpt_training_validation = tpt_data.groupby("name").apply(typicalsamling,
    typicalNDict_ training_validation)
result_tpt_ training_validation.to_csv("E:\\ train_v.csv", index=False)
result_tpt_testing=tpt_data.append(result_tpt_training_validation).drop_duplicates(ke
ep=False)
result_tpt_ testing.to_csv("E:\\ testing.csv", index=False)
print(result_tpt_ training_validation ["name"].value_counts())
print(result_tpt_testing["name"].value_counts())

# Training set and Validation set
tpt_data = pd.read_csv(r"E:\train_v.csv", header=0)
tpt_data.drop_duplicates(inplace=True)
tpt_sample =tpt_data.sample(len(tpt_data), random_state=0)
train_rate = 0.75
num_train_tup = np.array([(int)(round(len(tpt_sample) * train_rate))])
num_validation_tup = len(tpt_sample) – num_train_tup
print(num_train_tup)
print(num_validation_tup)
TPT_test_X = result_tpt_testing.iloc[:1171, 2:-1]
TPT_test_Y = result_tpt_testing.iloc[1171:, -1]
TPT_train_X = tpt_sample.iloc[:3512, 2:-1]
TPT_train_Y = tpt_sample.iloc[:3512, -1]
TPT_validation_X = tpt_sample.iloc[3512:,2:-1]
```

```
TPT_validation_Y=tpt_sample.iloc[3512:,-1]
```

Machine Learning

```
import numpy as np
import scipy
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import explained_variance_score
from sklearn.metrics import roc_curve, auc, roc_auc_score
tlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from math import sqrt
import keras
from keras import layers
from keras import models
from keras.layers import LeakyReLU
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Dropout
from tensorflow.python.keras.utils.multi_gpu_utils import multi_gpu_model
from keras import regularizers
from sklearn.model_selection import train_test_split
from keras.utils.vis_utils import plot_model
```

LR

```

lr121 = linear_model.LinearRegression()
lr121.fit (TPT_train_X, TPT_train_Y)
```

```
lr_va_result = lr121.predict(TPT_validation_X)
lr_test_result=lr121.predict(TPT_test_X)

SVM
rbf_svr=SVR
#Gridsearch
NOTE: SVM, RF and GBDT were all using this method to find the optimal hyperparameters. Thus, we only displayed Girdsearch one time, the process of Girdsearch is no longer shown in RF and GBDT.
param_test1= {"kernel":["linear","poly","rbf"]}
gsearch1= GridSearchCV(estimator = SVR(gamma=0.1,C=30),param_grid=
param_test1, scoring=None,cv=10)
gsearch1.fit(TPT_train_X,TPT_train_Y)
print(gsearch1.best_params_, gsearch1.best_score_)

param_test2= { "C":list(range(1,100,1))}
gsearch2= GridSearchCV(estimator = SVR(gamma=0.1, kernel='rbf'), param_grid=
param_test2, scoring=None,cv=10)
gsearch2.fit(TPT_train_X,TPT_train_Y)
print(gsearch2.best_params_, gsearch2.best_score_)

param_test3={ "gamma":[0.1,0. 2,0. 3,0.4,0.5,0.6,0.7,0.8,0.9,1,2,3,4,5,6,7,8,9,10]}
gsearch3= GridSearchCV(estimator = SVR(C=99, kernel='rbf'), param_grid=
param_test3, scoring=None,cv=10)
gsearch3.fit(TPT_train_X,TPT_train_Y)
print(gsearch3.best_params_, gsearch3.best_score_)


rbf_svr121=SVR(kernel='rbf', gamma=0.3.7, C=99)
rbf_svr121.fit(TPT_train_X,TPT_train_Y)
svr_va_result = rbf_svr121.predict(TPT_validation_X)
```

```
svr_test_result = rbf_svr121.predict(TPT_test_X)
RF
rfc121=RandomForestRegressor(n_estimators=55,max_depth=19,
min_samples_split=2,min_samples_leaf=1,random_state=10)
rfc121 = rfc121.fit(TPT_train_X, TPT_train_Y)
rf_va_result = rfc121.predict(TPT_validation_X)
rf_test_result = rfc121.predict(TPT_test_X)

GBDT
gbdt121=GradientBoostingRegressor(learning_rate=0.2,n_estimators=89,subsample=
0.8,max_depth=16,max_features='sqrt',random_state=10,min_samples_split=13,min_
samples_leaf=1)
gbdt121.fit(TPT_train_X, TPT_train_Y)
gbdt_va_result = gbdt121.predict(TPT_validation_X)
gbdt_test_result = gbdt121.predict(TPT_test_X)

BPNN
bpnn121 = Sequential()
bpnn121.add(Dense(units = 8,
                  activation='LeakyReLU',
                  input_shape=(TPT_train_X.shape[1],)
                 )
           )
bpnn121.add(Dropout(0.005))
bpnn121.add(Dense(units = 20,
                  kernel_regularizer = keras.regularizers.l2(0.005),
                  activity_regularizer = keras.regularizers.l1(0.01),
                  activation='LeakyReLU'
                  #bias_regularizer = keras.regularizers.l1,l2(0.01)
                  )
           )
```

```
bpnn121.add(Dense(units = 1,
                  activation='relu'
                 )
           )
print(bpnn121.summary())
bpnn121.compile(loss='mse',
                optimizer='Nadam',
               )

history = bpnn121.fit(TPT_train_X, TPT_train_Y,
          epochs=600,
          batch_size=40,
          verbose=2,
          validation_data = (TPT_validation_X, TPT_validation_Y)
         )

import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

bpnn_va_result = bpnn121.predict(TPT_validation_X)
bpnn_test_result = bpnn121.predict(TPT_test_X)
```

Models accuracy

NOTE: All models applied these codes to calculate accuracy, LR was used for display:

```
lr_vaNSE = r2_score(TPT_validation_Y,lr_va_result)
```

```
lr_vaMSE =  mean_squared_error(TPT_validation_Y,lr_va_result)
lr_vaMAE = mean_absolute_error(TPT_validation_Y,lr_va_result)
lr_vaRMSE = np.sqrt(lr_vaMSE)
lr_vaAD = np.mean(lr_va_result - TPT_validation_Y)

lr_testNSE = r2_score(TPT_test_Y,lr_test_result)
lr_testMSE =  mean_squared_error(TPT_test_Y,lr_test_result)
lr_testMAE = mean_absolute_error(TPT_test_Y,lr_test_result)
lr_testTRMSE = np.sqrt(lr_testMSE)
lr_testAD = np.mean(lr_test_result - TPT_test_Y)
```