Reservoir Description Machine Learning Challenge

In [22]:

```
### Your assignment is based on this SEG Machine Learning Contest https://github.com/seg/2016-ml-contest.
Read more on the example case here: https://github.com/seg/2016-ml-contest/blob/master/Facies_classification
```

```
  File "<ipython-input-22-4089ec26b745>", line 2
    Read more on the example case here: https://github.com/seg/2016-ml-contest/blob/master/Facies_classifica
tion.ipynb
            ^
SyntaxError: invalid syntax
```

You are given complete dataset of the field's Well Log Models generated from TNAV's software. There are 8 wells which dataset are not complete in some of the wells due to cost challenge imposed by the management as impact of Covid Crisis. Your senior engineer and Geologist have developed a Machine Learning model to predict Facies based Random Forest Algorithm. The accuracy of the model is 0.8678479783618837 (as shown at the end of the notebook).

However you are challenged to beat the benchmark of the accuracy result using your own choice of Machine Learning algorithm.

Instruction on How to Start

The instruction here is given to you to help you embark exploring your data and understanding how to build your first Machine Learning algorithm.

1. Download Python Software to run this notebook. You can use Anaconda or your own favorite Python Toolkit. Anaconda installation link: https://repo.anaconda.com/archive/Anaconda3-2020.11-Windows-x86_64.exe

1. After you complete the installation, if you're not familiar with Python or Anaconda you can watch series of Youtube video or other online resources on starting up your first Python program. This Youtube Video is one of the example on Anaconda tutorial: https://www.youtube.com/watch?v=beh7GE4FdnM. You are free to use other programming Tools such as R or Matlab if you are more proficient in using those programming language.

1. If you have experienced using Python and has already build ML applications before then congratulations. The committee will evluate the level of difficulty for Machine Learning problems for the next round based on the best performing team. There are a lot of resources to learn about Machine Learning algorithms in Python, for further reading you can learn from Scikit-Learn, one of the most popular ML library in python here: https://scikit-learn.org/stable/

1. If you dont need introductary Python resources as mentioned on step 1 to 3 above, you can just go straight ahead to Case Description bellow and lets begin to Code!

Invoke and install Python modules required for the work

Python has a lot of Open Source libraries of functions that you could use for any type of Data Science or programming purposes. The beauty of these open source Libraries is you dont need to write your own function for generic Machine Learning application. However, every Machine Learning applications are built differently and need to be tuned in order to improve its accuracy. The following lines of codes are necessary to call all the built in functions/Library for your ML project.s

In [ ]:

```
import pandas as pd
import re
import os
import datetime as dt
import xlrd
import numpy as np
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from mpl_toolkits.axes_grid1 import make_axes_locatable
import seaborn as sns

from pandas import set_option
# set_option("display.max_rows", 10)
pd.options.mode.chained_assignment = None

##### import stuff from scikit learn
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
from sklearn.model_selection import KFold, cross_val_score,LeavePGroupsOut, LeaveOneGroupOut, cross_val_pred
from sklearn.metrics import confusion_matrix, make_scorer, f1_score, accuracy_score, recall_score, precision
from sklearn.multiclass import OneVsOneClassifier,OneVsRestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
```

Set your working directory

Before we go any further, set your own working folder. This is where you stored all the CSV files from TNAV well logs model.

In [ ]:

```
os.chdir('C:/Users/rim3dh/POD_Competition')
```

Pro Tip: you will see a lot of hash (#) symbol. This indicate comment that I put to describe what I did to the code

Compile the wells dataset

In [ ]:

```
#Call all your Well Log CSV file as your dataframe

df1 = pd.read_csv("I1.csv",delimiter=r"\s+")
df1['WELLNAME'] = "I1"
df2  = pd.read_csv("I3.csv",delimiter=r"\s+")
df2['WELLNAME'] = 'I3'
df3 = pd.read_csv("W1.csv",delimiter=r"\s+")
df3['WELLNAME'] = 'W1'
df4 = pd.read_csv("W2.csv",delimiter=r"\s+")
df4['WELLNAME'] = 'W2'
df5 = pd.read_csv("W3.csv",delimiter=r"\s+")
df5['WELLNAME'] = 'W3'
df6 = pd.read_csv("W4.csv",delimiter=r"\s+")
df6['WELLNAME'] = 'W4'
df7 = pd.read_csv("W5.csv",delimiter=r"\s+")
df7['WELLNAME'] = 'W5'
df8 = pd.read_csv("X1.csv",delimiter=r"\s+")
df8['WELLNAME'] = 'x1'


df = pd.concat([df1,df2, df3, df4, df5, df6, df7, df8])
df =df.replace(-999.25, 0)
```

Exploring The Data

In [ ]:

```
df.describe()
```

In [ ]:

```
# Lets label your Facies according to geological description
# ============================================================

# 0 =Sale, 1=sandstone, 2=c_siltstone, 3=Coal
facies_colors = [ 'gray', '#F4D03F', 'green','black']
facies_labels = ['Shale', 'Sandstone', 'Silt', 'Coal']
facies_dict = { 0: 'Shale', 1: 'Sandstone', 2: 'Silt', 3: 'Coal'}

#facies_color_map is a dictionary that maps facies labels to their respective colors
facies_color_map = {}
for ind, label in enumerate(facies_labels):
    facies_color_map[label] = facies_colors[ind]

# Replace numeric "Facies" with "FaciesLabels"
df["FaciesLabels"] = df["Facies"].replace(facies_dict)
```

In [ ]:

```
# Count the number of unique entries for each facies, sort them by facies number (instead of by number of en
facies_counts = df['Facies'].value_counts().sort_index()

# Use facies labels to index each count
facies_counts.index = facies_labels

facies_counts.plot(kind='bar',color=facies_colors, title='Distribution of Training Data by Facies')
facies_counts
```

Visualize The Well Data

In [ ]:

```
import matplotlib.colors as colors
from mpl_toolkits.axes_grid1 import make_axes_locatable

def make_facies_log_plot(logs, facies_colors):
    #make sure logs are sorted by depth
    logs = logs.sort_values(by='DEPTH')
    cmap_facies = colors.ListedColormap(facies_colors[0:len(facies_colors)], 'indexed')

    ztop=logs.DEPTH.min(); zbot=logs.DEPTH.max()

    cluster=np.repeat(np.expand_dims(logs['Facies'].values,1), 100, 1)

    f, ax = plt.subplots(nrows=1, ncols=11, figsize=(15, 18))
    ax[1].plot(logs.SP, logs.DEPTH, '-g')
    ax[0].plot(logs.GR, logs.DEPTH, '-')
    ax[2].plot(logs.RES, logs.DEPTH, '-', color='0.5')
    ax[3].plot(logs.RHOB, logs.DEPTH, '-', color='orange')
    ax[4].plot(logs.NPHI, logs.DEPTH, '-', color='black')
    ax[5].plot(logs.VSH, logs.DEPTH, '-', color='purple')
    ax[6].plot(logs.PHIE, logs.DEPTH, '-', color='cyan')
    ax[7].plot(logs.SW, logs.DEPTH, '-', color='blue')
    ax[8].plot(logs.Core_Porosity, logs.DEPTH, '-', color='red')
    ax[9].plot(logs.Core_Permeability, logs.DEPTH, '-', color='gold')

    im=ax[10].imshow(cluster, interpolation='none', aspect='auto',
                    cmap=cmap_facies,vmin=1,vmax=4)

    divider = make_axes_locatable(ax[10])
    cax = divider.append_axes("right", size="20%", pad=0.05)
    cbar=plt.colorbar(im, cax=cax)
    cbar.set_label((17*' ').join(['Shale', 'SS', 'Silt', 'Coal']))
    cbar.set_ticks(range(0,1)); cbar.set_ticklabels('')

    for i in range(len(ax)-1):
        ax[i].set_ylim(ztop,zbot)
        ax[i].invert_yaxis()
        ax[i].grid()
        ax[i].locator_params(axis='x', nbins=3)

    ax[1].set_xlabel("SP")
    ax[1].set_xlim(logs.SP.min(),logs.SP.max())
    ax[0].set_xlabel("GR")
    ax[0].set_xlim(logs.GR.min(),logs.GR.max())
    ax[2].set_xlabel("RES")
    ax[2].set_xlim(logs.RES.min(),logs.RES.max())
    ax[3].set_xlabel("RHOB")
    ax[3].set_xlim(logs.RHOB.min(),logs.RHOB.max())
    ax[4].set_xlabel("NPHI")
    ax[4].set_xlim(logs.NPHI.min(),logs.NPHI.max())
    ax[5].set_xlabel("VSH")
    ax[5].set_xlim(logs.VSH.min(),logs.VSH.max())
    ax[6].set_xlabel("PHIE")
    ax[6].set_xlim(logs.PHIE.min(),logs.PHIE.max())
    ax[7].set_xlabel("SW")
    ax[7].set_xlim(logs.SW.min(),logs.SW.max())
    ax[8].set_xlabel("Core_Porosity")
    ax[8].set_xlim(logs.Core_Porosity.min(),logs.Core_Porosity.max())
    ax[9].set_xlabel("Core_Permeability")
    ax[9].set_xlim(logs.Core_Permeability.min(),logs.Core_Permeability.max())

    ax[10].set_xlabel('Facies')

    ax[1].set_yticklabels([]); ax[2].set_yticklabels([]); ax[3].set_yticklabels([])
    ax[4].set_yticklabels([]);  ax[5].set_yticklabels([]); ax[6].set_yticklabels([]);
    ax[7].set_yticklabels([]); ax[8].set_yticklabels([]); ax[9].set_yticklabels([]);
    ax[10].set_yticklabels([])

    ax[10].set_xticklabels([])
    f.suptitle('Well: %s'%logs.iloc[0]['WELLNAME'], fontsize=14,y=0.94)
```

In [ ]:

```
#Example Display of Well Log
make_facies_log_plot(df[df['WELLNAME'] == 'W4'], facies_colors)
```

Preparing the data for machine learning

In [ ]:

```
#Split data train and validation
# split the data into train and test set
from sklearn.model_selection import train_test_split
training_data, validation_data = train_test_split(df, test_size=0.7, random_state=42, shuffle=True)
```

Visualize the dataset

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.pairplot(training_data.drop(['WELLNAME', "Facies"], axis=1), hue='FaciesLabels', palette=facies_color_ma
             hue_order=list(reversed(facies_labels)), size=1.25);
```

Separate features from the labels for our training data

In [ ]:

```
X = training_data[["SP","GR","RES","RHOB","NPHI","VSH","PHIE","SW"]]
y = training_data["Facies"]
```

Do the same for our blind/validation data

In [ ]:

```
X_validation = validation_data[["SP","GR","RES","RHOB","NPHI","VSH","PHIE","SW"]]
y_validation = validation_data["Facies"]
```

Standardising/whitening the data

scikit-learn includes a preprocessing module that can 'standardise' the data (giving each variable zero mean and unit variance, also called whitening).

Many machine learning algorithms assume features will be standard normally distributed data (ie: Gaussian with zero mean and unit variance).

In [ ]:

```
from sklearn import preprocessing

# Standardise the training data
scaler = preprocessing.StandardScaler().fit(X)
X = scaler.transform(X)
```

The factors used to standardise the training set must be applied to any subsequent feature set that will be input to the classifier.

In [ ]:

```
# Standardise the validation data using the same transform
X_validation = scaler.transform(X_validation)
```

In [ ]:

```
# Choosing an estimator: (Example Algorithm) Random Forest
```

Now it's time to build your firs Random Forest ML Model

In [ ]:

```
Cl = RandomForestClassifier(n_estimators=100, max_features=0.1, min_samples_leaf=25,
                            min_samples_split=50, class_weight='balanced', random_state=42, n_jobs=-1)

OVR = OneVsRestClassifier(Cl,n_jobs=-1)
groups = training_data['WELLNAME']
methods = [Cl, OVR]
method_list = ['RF submission 3','One vs Rest']

lpgo = LeavePGroupsOut(n_groups=2)

scores = []

for method in methods:

    cv=lpgo.split(X, y, groups)
    validated = cross_val_score(method, X, y, scoring="f1_weighted", cv=cv, n_jobs=-1)
    scores.append(validated)

scores = np.array(scores)
scores = np.swapaxes(scores, 0, 1)
scores = pd.DataFrame(data=scores, columns=method_list)
```

In [ ]:

```
scores.head()
```

Applying Random Forest Algorithm to validation data

In [ ]:

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf2=RandomForestClassifier(n_estimators=100, max_features=0.1, min_samples_leaf=25,
                            min_samples_split=50, class_weight='balanced', random_state=42, n_jobs=-1)

OVR = OneVsRestClassifier(Cl,n_jobs=-1)
groups = training_data['WELLNAME']
methods = [Cl, OVR]
method_list = ['RF submission 3','One vs Rest']

lpgo = LeavePGroupsOut(n_groups=2)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf2.fit(X,y)

y_pred=clf2.predict(X_validation)
```

In [ ]:

```
# Code for printing a pretty confusion matrix - hidden during presentation

def display_cm(cm, labels, hide_zeros=False, display_metrics=False):
    """Display confusion matrix with labels, along with
       metrics such as Recall, Precision and F1 score.
       Based on Zach Guo's print_cm gist at
       https://gist.github.com/zachguo/10296432
    """

    precision = np.diagonal(cm)/cm.sum(axis=0).astype('float')
    recall = np.diagonal(cm)/cm.sum(axis=1).astype('float')
    F1 = 2 * (precision * recall) / (precision + recall)

    precision[np.isnan(precision)] = 0
    recall[np.isnan(recall)] = 0
    F1[np.isnan(F1)] = 0

    total_precision = np.sum(precision * cm.sum(axis=1)) / cm.sum(axis=(0,1))
    total_recall = np.sum(recall * cm.sum(axis=1)) / cm.sum(axis=(0,1))
    total_F1 = np.sum(F1 * cm.sum(axis=1)) / cm.sum(axis=(0,1))

    columnwidth = max([len(x) for x in labels]+[5]) # 5 is value length
    empty_cell = " " * columnwidth
    # Print header
    print("    " + " Pred", end=' ')
    for label in labels:
        print("%{0}s".format(columnwidth) % label, end=' ')
    print("%{0}s".format(columnwidth) % 'Total')
    print("    " + " True")
    # Print rows
    for i, label1 in enumerate(labels):
        print("    %{0}s".format(columnwidth) % label1, end=' ')
        for j in range(len(labels)):
            cell = "%{0}d".format(columnwidth) % cm[i, j]
            if hide_zeros:
                cell = cell if float(cm[i, j]) != 0 else empty_cell
            print(cell, end=' ')
        print("%{0}d".format(columnwidth) % sum(cm[i,:]))

    if display_metrics:
        print()
        print("Precision", end=' ')
        for j in range(len(labels)):
            cell = "%{0}.3f".format(columnwidth) % precision[j]
            print(cell, end=' ')
        print("%{0}.3f".format(columnwidth) % total_precision)
        print("   Recall", end=' ')
        for j in range(len(labels)):
            cell = "%{0}.3f".format(columnwidth) % recall[j]
            print(cell, end=' ')
        print("%{0}.3f".format(columnwidth) % total_recall)
        print("       F1", end=' ')
        for j in range(len(labels)):
            cell = "%{0}.3f".format(columnwidth) % F1[j]
            print(cell, end=' ')
        print("%{0}.3f".format(columnwidth) % total_F1)
```

In [ ]:

```
predicted_labels = clf2.predict(X_validation)

conf = confusion_matrix(y_validation, predicted_labels)
display_cm(conf, facies_labels, display_metrics=True, hide_zeros=True)
```

In [ ]:

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_validation, y_pred))
```

Your Task

In [ ]: