3 Final Project Submission

- name: Leticia D Drasler (Fernandes)
- pace: Part time
- Scheduled project review data/time: November 16th, 2021, 08:00 AM (Mountain Time)
- Course Instructor: Abhineet

Applying PIPELINE

In [36]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')
```

In [37]:
```
df_values = pd.read_csv('training_set_values.csv', index_col='id')
df_labels = pd.read_csv('training_set_labels.csv', index_col='id')
```

In [38]:
```
df_training = pd.concat([df_labels, df_values], axis=1, join='inner')
df_training.head()
```

Out[38]:

id	status_group	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	w
69572	functional	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	
8776	functional	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	
34310	functional	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	
67743	non functional	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	N
19728	functional	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	

5 rows × 40 columns

In [39]:
```
to_drop =['num_private','date_recorded','longitude','latitude','subvillage',
          'region_code','district_code','lga','ward','recorded_by',
          'scheme_management','scheme_name', 'extraction_type_group',
          'extraction_type_class','management_group','payment',
          'quality_group','quantity_group','source_type',
          'source_class','waterpoint_type_group','wpt_name']
df_training.drop(to_drop,axis=1, inplace=True)
```

In [40]:
```
df_training.public_meeting.fillna(False,inplace=True)
```

In [41]:
```
df_training.permit.fillna(False,inplace=True)
```

In [42]:
```
cat_sub=['funder','installer','public_meeting','construction_year',
         'extraction_type','permit','basin']
```

In [43]:
```
categoricals=['funder','installer','management','public_meeting',
              'construction_year','extraction_type','permit','basin',
              'region','population','water_quality','quantity','source',
              'waterpoint_type','payment_type'
             ]
categoricals
```

Out[43]:
```
['funder',
 'installer',
 'management',
 'public_meeting',
 'construction_year',
 'extraction_type',
 'permit',
 'basin',
 'region',
 'population',
 'water_quality',
 'quantity',
 'source',
 'waterpoint_type',
 'payment_type']
```

In [44]:

```
def initial_drop(data):
        """
        Helper function that drops our duplacated data.
        """
        return data.drop(to_drop,axis=1)

def funder_transform(data):
    funder_bins=list(data.funder.value_counts().index[:8])
    funder_dict=dict(zip(funder_bins,range(1,len(funder_bins)+1)))
    data['funder']=data['funder'].apply(
        lambda x: funder_dict[x]if x in funder_bins else 0 )
    return data

def installer_transform(data):
    installers=list(data.installer.value_counts()[:10].index)
    installers.remove('0')
    installers_dict = dict(zip(installers,range(1,len(installers)+1)))
    data['installer']=data['installer'].apply(
        lambda x: installers_dict[x] if x in installers else 0 )
    return data

def management_transform(data):
    management=list(data.management.value_counts()[:4].index)
    management_dict = dict(zip(management,range(1,len(management)+1)))
    data['management']=data['management'].apply(
        lambda x: management_dict[x] if x in management else 0 )
    return data

def public_meeting_transform(data):
    data['public_meeting']=data['public_meeting'].fillna(False)
    binary_map={False:0, True:1}
    data['public_meeting']=data['public_meeting'].replace(binary_map)
    return data

def permit_tranform(data):
    data['permit']=data['permit'].fillna(False)
    return data

def construction_year_tranform(data):
    max_year = float(df_training['construction_year'].describe()['max'])
    min_year=float(df_training['construction_year'][
        df_training['construction_year']!=0].sort_values(ascending=True).iloc[0])
    year_bins=[np.round(x) for x in np.linspace(min_year,max_year,7) ]
    year_bins=[0,1]+year_bins[1:]
    data['construction_year']=pd.cut(data[
        'construction_year'],[0,1,1960,1969,1978,1987,1995,2004,2013],
       include_lowest=True,labels=[1,2,3,4,5,6,7,8])
    return data

def extractions_transform(data):
    extractions=list(df_training.extraction_type.value_counts()[0:4].index)
    extractions.remove('other')
    extractions_dict = dict(zip(extractions,range(1,len(extractions)+1)))
    data['extraction_type']=data['extraction_type'].apply(
        lambda x: extractions_dict[x] if x in extractions else 0 )
    return data

def population_transform(data):
    data['population']=data['population'].apply(lambda x: 1 if x>1 else 0)
```

```
    return data

def one_hot_encoder(data):
    data=pd.get_dummies(data,columns=categoricals,drop_first=True)
    return data
```

In [47]:

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler

status_map={'non functional':0,'functional':1,'functional needs repair':2}
y=df_labels.replace(status_map)
X=df_values

X_train, X_test, y_train, y_test = train_test_split (
    X, y, test_size = 0.25, random_state=42)

# Create a column transformer
col_transformer = ColumnTransformer(transformers=[
    ("ohe", OneHotEncoder(
        categories="auto", handle_unknown="ignore"),categoricals)],
                                    remainder="passthrough")



param_grid_optimal = {
    'learning_rate': [0.3],
    'max_depth': [6],
    'min_child_weight': [1],
    'subsample': [0.5],
    'n_estimators': [100],
}

pipe = Pipeline(steps=[
    ("initial column drop", FunctionTransformer(initial_drop)),
    ("Transform Funder into Bins",FunctionTransformer(funder_transform)),
    ("Transform Installer into Bins",FunctionTransformer(installer_transform)),
    ("Transform Management into Bins",FunctionTransformer(management_transform)),
    ("Fill Public Meeting missing values",FunctionTransformer(
        public_meeting_transform)),
    ("Fill Permit missing values",FunctionTransformer(permit_tranform)),
    ("Transform Construction Year into Bins",FunctionTransformer(
        construction_year_tranform)),
    ("Transform Extractions into Bins",FunctionTransformer(extractions_transform)),
    ("Transform Populations into Binary",FunctionTransformer(population_transform)),
    ("OHE",FunctionTransformer(one_hot_encoder)),
    ('scale', StandardScaler()),
    ("model", XGBClassifier(
        learning_rate=0.3, max_depth=6,min_child_weight=1, subsample=0.5,
        n_estimators=100))
])

# Use the pipeline to fit and transform the data
pipe.fit(X_train,y_train)
```

```
[10:51:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/
learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with t
he objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set
eval_metric if you'd like to restore the old behavior.
```

Out[47]:

```
Pipeline(steps=[('initial column drop',
                 FunctionTransformer(func=<function initial_drop at 0x00000210A3
9FB550>)),
```

```
                ('Transform Funder into Bins',
                 FunctionTransformer(func=<function funder_transform at 0x000002
10A9611EE0>)),
                ('Transform Installer into Bins',
                 FunctionTransformer(func=<function installer_transform at 0x000
00210A96115E0>)),
                ('Transform Management into Bins',
                 Func...
                              importance_type='gain',
                              interaction_constraints='', learning_rate=0.3,
                              max_delta_step=0, max_depth=6,
                              min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=100,
                              n_jobs=8, num_parallel_tree=1,
                              objective='multi:softprob', random_state=0,
                              reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
                              subsample=0.5, tree_method='exact',
                              validate_parameters=1, verbosity=None))])
```

In [46]:
```
pipe.score(X_test,y_test)
```

Out[46]:
```
0.7826936026936027
```