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 [9]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.utils.class_weight import compute_sample_weight
import warnings
warnings.filterwarnings('ignore')
```

In [10]:
```
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 [11]:
```
df_training = pd.concat([df_labels, df_values], axis=1, join='inner')
df_training.head()
```

Out[11]:

s_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type	water_quality	quality_
1390	Roman	34.938093	-9.856322	none	0	...	annually	soft	
1399	GRUMETI	34.698766	-2.147466	Zahanati	0	...	never pay	soft	
686	World vision	37.460664	-3.821329	Kwa Mahundi	0	...	per bucket	soft	
263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu	0	...	never pay	soft	
0	Artisan	31.130847	-1.825359	Shuleni	0	...	never pay	soft	

In [12]:
```
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']
```

In [13]:
```
categoricals=['funder','installer','management','public_meeting',
              'construction_year','extraction_type','permit','basin',
              'region','population','water_quality','quantity','source',
              'waterpoint_type','payment_type'
             ]
```

In [14]:

```
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 [31]:

```
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=30)


sample_weights=compute_sample_weight(
    class_weight='balanced',
    y=y_train)

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(sample_weight=sample_weights,
        learning_rate=0.22, max_depth=10,min_child_weight=1, subsample=0.8,
        n_estimators=50))
])

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

```
[20:16:43] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/
learner.cc:573:
Parameters: { "sample_weight" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings
but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[20:16:43] 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[31]:

```
Pipeline(steps=[('initial column drop',
                 FunctionTransformer(func=<function initial_drop at 0x000001A223
F0E940>)),
                ('Transform Funder into Bins',
```

```
                 FunctionTransformer(func=<function funder_transform at 0x000001
A21DC3BC10>)),
                ('Transform Installer into Bins',
                 FunctionTransformer(func=<function installer_transform at 0x000
001A21DC3BD30>)),
                ('Transform Management into Bins',
                 Func...
                              min_child_weight=1, missing=nan,
                              monotone_constraints='()', n_estimators=50,
                              n_jobs=8, num_parallel_tree=1,
                              objective='multi:softprob', random_state=0,
                              reg_alpha=0, reg_lambda=1,
                              sample_weight=array([0.61648954, 0.61648954, 0.61
648954, ..., 0.86407541, 0.61648954,
       0.61648954]),
                              scale_pos_weight=None, subsample=0.8,
                              tree_method='exact', validate_parameters=1,
                              verbosity=None))])
```

In [32]:
```
pipe.score(X_test,y_test)
```

Out[32]:
```
0.7926599326599326
```

In [ ]:
```

```