MachineLearningProject

January 15, 2022

[1]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder,MinMaxScaler,StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score,mean_squared_error,r2_score,␣
↪mean_absolute_error
from sklearn.model_selection import train_test_split
from math import sqrt
import numpy as np
import xgboost
```

[2]:
```
from sklearn.model_selection import GridSearchCV
```

[3]:
```
merc_train_data = pd.read_csv('/home/labsuser/Datasets/train.csv',index_col=0)
print(merc_train_data.shape)

merc_test_data = pd.read_csv('/home/labsuser/Datasets/test.csv',index_col=0)
print(merc_test_data.shape)
```

```
(4209, 377)
(4209, 376)
```

[4]:
```
merc_train_data.isnull().sum()
merc_train_data.isna().sum()
```

[4]:
```
y       0
X0      0
X1      0
X2      0
X3      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 377, dtype: int64
```

0.1 Label Encoding - Train Test

[5]:
```
#Categorical Variables Split
categorical_columns = ['X0','X1','X2','X3','X4','X5','X6','X8']
cat_subset_train = pd.DataFrame(merc_train_data,columns=categorical_columns)
numeric_subset_train = pd.DataFrame(merc_train_data,columns = merc_train_data.
 ↪drop(categorical_columns,axis = 1).columns)

#Label Encoder
encoder = LabelEncoder()
cat_subset_train_encoded = cat_subset_train

for i in range(0,len(cat_subset_train_encoded.columns)):
    if(i==7):
        i+=1
    encoder.fit(cat_subset_train_encoded['X'+str(i)])
    #print(encoder.classes_ + "for the column - "+cat_subset.columns[i])
    cat_subset_train_encoded['X'+str(i)] = encoder.
 ↪transform(cat_subset_train_encoded['X'+str(i)])


final_train_data = pd.
 ↪concat([cat_subset_train_encoded,numeric_subset_train],axis=1)
y_train = final_train_data['y']
X_train = final_train_data.drop('y',axis=1)
```

[6]:
```
#-----------------------TEST -----------------

cat_subset_test = pd.DataFrame(merc_test_data,columns=categorical_columns)
numeric_subset_test = pd.DataFrame(merc_test_data,columns = merc_test_data.
 ↪drop(categorical_columns,axis = 1).columns)

#Label Encoder
encoder_test = LabelEncoder()
cat_subset_test_encoded = cat_subset_test

for i in range(0,len(cat_subset_test_encoded.columns)):
    if(i==7):
        i+=1
    encoder_test.fit(cat_subset_test_encoded['X'+str(i)])
    #print(encoder.classes_ + "for the column - "+cat_subset.columns[i])
    cat_subset_test_encoded['X'+str(i)] = encoder_test.
 ↪transform(cat_subset_test_encoded['X'+str(i)])
```

```
final_test_data = pd.
 ↪concat([cat_subset_test_encoded,numeric_subset_test],axis=1)
X_test_validation = final_test_data
```

0.2 Removing variables with 0 variance

[7]:
```
numeric_data_train = X_train.loc[:,'X10':]


to_drop = []
for i in numeric_data_train.columns:

    if(numeric_data_train[i].var() == float(0)):
        to_drop.append(i)

print(to_drop)
numeric_data_train = X_train.drop(to_drop,axis=1)


# Create correlation matrix
corr_matrix = numeric_data_train.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.
 ↪bool))

# Find features with correlation greater than 0.8
to_drop_corr = [column for column in upper.columns if any(upper[column] > 0.8)]

print('Total correlated columns are', len(to_drop_corr),'\n')
print('Total variance 0 to drop', len(to_drop))

to_drop_final = np.concatenate([to_drop,to_drop_corr])
X_train = X_train.drop(to_drop_final,axis=1)
X_test_validation = X_test_validation.drop(to_drop_final,axis=1)
print('Shape of Data after dropping features - ',X_train.shape)
print('Shape of Test Data after dropping features - ',X_test_validation.shape)
```

```
['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X290', 'X293', 'X297',
'X330', 'X347']
Total correlated columns are 128

Total variance 0 to drop 12
Shape of Data after dropping features -  (4209, 236)
Shape of Test Data after dropping features -  (4209, 236)
```

[8]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test_validation.shape)
```

```
(4209, 236)
(4209,)
(4209, 236)
```

0.3 PCA

[9]:
```
X_train, X_test, y_train, y_test=␣
↪train_test_split(X_train,y_train,random_state=120,train_size=0.8)

pca = PCA(n_components=4)

pca.fit(X_train)
```

[9]:
```
PCA(n_components=4)
```

[10]:
```
X_train_trans = pca.transform(X_train)
```

[11]:
```
X_train_trans.shape
```

[11]:
```
(3367, 4)
```

[12]:
```
X_test_trans = pca.transform(X_test)
```

[13]:
```
X_test_trans.shape
```

[13]:
```
(842, 4)
```

0.4 XGBoost GridSearch

[14]:
```
xgb = xgboost.XGBRegressor()
```

[15]:
```
parameters ={ 'nthread': [10], 'objective': ['reg:linear'], 'learning_rate': [0.
↪3,0.4,0.5], 'max_depth': [5,6,7], 'n_estimators': [100,300,500]}
```

[16]:
```
xgb_grid = GridSearchCV(xgb,parameters,cv=2,n_jobs=10,verbose=True)
```

[17]:
```
xgb_grid.fit(X_train,y_train)
```

```
Fitting 2 folds for each of 27 candidates, totalling 54 fits
[16:06:23] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linear
is now deprecated in favor of reg:squarederror.
```

[17]:
```
GridSearchCV(cv=2,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
                                    gpu_id=None, importance_type='gain',
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight=None,
                                    missing=nan, monotone_constraints=None,
                                    n_estimators=100, n_jobs=None,
                                    num_parallel_tree=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None,
                                    scale_pos_weight=None, subsample=None,
                                    tree_method=None, validate_parameters=False,
                                    verbosity=None),
             n_jobs=10,
             param_grid={'learning_rate': [0.3, 0.4, 0.5],
                         'max_depth': [5, 6, 7],
                         'n_estimators': [100, 300, 500], 'nthread': [10],
                         'objective': ['reg:linear']},
             verbose=True)
```

[18]:
```
print(xgb_grid.best_params_)
```

```
{'learning_rate': 0.3, 'max_depth': 5, 'n_estimators': 100, 'nthread': 10,
'objective': 'reg:linear'}
```

0.5 Using Best Params to fit the Final Model

[19]:
```
xgb = xgboost.
 ↪XGBRegressor(max_depth=5,n_estimators=100,n_thread=10,learning_rate=0.3)
```

[20]:
```
xgb.fit(X_train,y_train)
```

[20]:
```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.3, max_delta_step=0, max_depth=5,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=0, n_thread=10, num_parallel_tree=1,
             random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
             subsample=1, tree_method=None, validate_parameters=False,
             verbosity=None)
```

[21]:
```
y_preds = xgb.predict(X_test)
```

[27]:
```
print("RMSE : "+str(sqrt(mean_squared_error(y_test,y_preds))))
print("Absolute Error:"+str(mean_absolute_error(y_test,y_preds)))
```

```
RMSE : 8.714052694250595
Absolute Error:5.7536925968704775
```

0.6 Predictions on Validation Data

[23]:
```
y_preds_validation = xgb.predict(X_test_validation)
```

[24]:
```
y_preds_validation
```

[24]:
```
array([ 90.50865 , 106.822655,  89.77226 , …,  90.94823 , 112.84172 ,
        94.79426 ], dtype=float32)
```

[ ]:
```

```