project1_Raya

June 16, 2021

[1]:

```
import numpy as np
import pandas as pd
# for dimensionality reduction
from sklearn.decomposition import PCA
```

[2]:

```
df_train = pd.read_csv('train.csv')
print('Size of training set: {} rows and {} columns'
      .format(*df_train.shape))
```

```
Size of training set: 4209 rows and 378 columns
```

[3]:

```
df_train.head()
```

[3]:

```
   ID       y  X0 X1  X2 X3 X4 X5 X6 X8  …  X375  X376  X377  X378  X379  \
0   0  130.81   k  v  at  a  d  u  j  o  …     0     0     1     0     0
1   6   88.53   k  t  av  e  d  y  l  o  …     1     0     0     0     0
2   7   76.26  az  w   n  c  d  x  j  x  …     0     0     0     0     0
3   9   80.62  az  t   n  f  d  x  l  e  …     0     0     0     0     0
4  13   78.02  az  v   n  f  d  h  d  n  …     0     0     0     0     0

   X380  X382  X383  X384  X385
0     0     0     0     0     0
1     0     0     0     0     0
2     0     1     0     0     0
3     0     0     0     0     0
4     0     0     0     0     0

[5 rows x 378 columns]
```

[4]:

```
y_train = df_train['y'].values
```

[5]:

```
cols = [c for c in df_train.columns if 'X' in c]
print('Number of features: {}'.format(len(cols)))

print('Feature types:')
df_train[cols].dtypes.value_counts()
```

```
Number of features: 376
Feature types:
```

[5]:

```
int64     368
object      8
dtype: int64
```

[6]:

```python
counts = [[], [], []]
for c in cols:
    typ = df_train[c].dtype
    uniq = len(np.unique(df_train[c]))
    if uniq == 1:
        counts[0].append(c)
    elif uniq == 2 and typ == np.int64:
        counts[1].append(c)
    else:
        counts[2].append(c)

print('Constant features: {} Binary features: {} Categorical features: {}\n'
      .format(*[len(c) for c in counts]))
print('Constant features:', counts[0])
print('Categorical features:', counts[2])
```

```
Constant features: 12 Binary features: 356 Categorical features: 8

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289',
'X290', 'X293', 'X297', 'X330', 'X347']
Categorical features: ['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

[7]:

```python
df_test = pd.read_csv('test.csv')
```

[8]:

```python
# remove columns ID and Y from the data as they are not used for learning
usable_columns = list(set(df_train.columns) - set(['ID', 'y']))
y_train = df_train['y'].values
id_test = df_test['ID'].values

x_train = df_train[usable_columns]
x_test = df_test[usable_columns]
```

[9]:

```python
#Check for null and unique values
def check_missing_values(df):
    if df.isnull().any().any():
        print("There are missing values in the dataframe")
    else:
        print("There are no missing values in the dataframe")
check_missing_values(x_train)
check_missing_values(x_test)
```

```
There are no missing values in the dataframe
There are no missing values in the dataframe
```

[10]:

```
#If for any column(s), the variance is equal to zero, then you need to remove␣
 ↪those variables
#Apply label encoder
for column in usable_columns:
    cardinality = len(np.unique(x_train[column]))
    if cardinality == 1:
        x_train.drop(column, axis=1) # Column with only one
        # value is useless so we drop it
        x_test.drop(column, axis=1)
    if cardinality > 2: # Column is categorical
        mapper = lambda x: sum([ord(digit) for digit in x])
        x_train[column] = x_train[column].apply(mapper)
        x_test[column] = x_test[column].apply(mapper)
x_train.head()
```

```
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:11:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/site-packages/ipykernel_launcher.py:12:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  if sys.path[0] == '':
```

[10]:

```
   X286  X350  X162  X190  X51  X277  X232  X312  X96  X257  …  X335  X288  \
0     0     0     0     0    0     0     0     0    0     0  …     0     0
1     0     0     0     0    1     0     0     0    1     0  …     0     0
2     1     1     1     0    1     0     1     0    1     0  …     0     0
3     1     1     1     0    0     0     1     0    1     0  …     0     0
4     1     1     1     0    1     0     1     0    1     0  …     0     0

   X164  X252  X81  X291  X161  X293  X212  X150
0     0     0    0     0     0     0     0     1
1     0     0    0     0     0     0     0     1
2     0     1    0     0     0     0     0     1
3     0     1    0     1     0     0     0     1
```

```
4      0     1    0     0     0     0     0     1

[5 rows x 376 columns]
```

[11]:

```
#Make sure the data is now changed into numericals
print('Feature types:')
x_train[cols].dtypes.value_counts()
```

```
Feature types:
```

[11]:

```
int64    376
dtype: int64
```

[12]:

```
#Perform dimensionality reduction
n_comp = 12
pca = PCA(n_components=n_comp, random_state=420)
pca2_results_train = pca.fit_transform(x_train)
pca2_results_test = pca.transform(x_test)
```

[13]:

```
#Training using xgboost

import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

x_train, x_valid, y_train, y_valid = train_test_split(
        pca2_results_train,
        y_train, test_size=0.2,
        random_state=4242)

d_train = xgb.DMatrix(x_train, label=y_train)
d_valid = xgb.DMatrix(x_valid, label=y_valid)
d_test = xgb.DMatrix(x_test)
d_test = xgb.DMatrix(pca2_results_test)

params = {}
params['objective'] = 'reg:linear'
params['eta'] = 0.02
params['max_depth'] = 4

def xgb_r2_score(preds, dtrain):
    labels = dtrain.get_label()
    return 'r2', r2_score(labels, preds)

watchlist = [(d_train, 'train'), (d_valid, 'valid')]

clf = xgb.train(params, d_train,
                1000, watchlist, early_stopping_rounds=50,
                feval=xgb_r2_score, maximize=True, verbose_eval=10)
```

```
[04:30:22] WARNING: /workspace/src/objective/regression_obj.cu:167: reg:linear
is now deprecated in favor of reg:squarederror.
[0]     train-rmse:99.14835     valid-rmse:98.26297     train-r2:-58.35295
valid-r2:-67.63754
Multiple eval metrics have been passed: 'valid-r2' will be used for early
stopping.

Will train until valid-r2 hasn't improved in 50 rounds.
[10]    train-rmse:81.27653     valid-rmse:80.36433     train-r2:-38.88428
valid-r2:-44.91014
[20]    train-rmse:66.71610     valid-rmse:65.77334     train-r2:-25.87403
valid-r2:-29.75260
[30]    train-rmse:54.86957     valid-rmse:53.88974     train-r2:-17.17752
valid-r2:-19.64401
[40]    train-rmse:45.24491     valid-rmse:44.21971     train-r2:-11.35979
valid-r2:-12.89996
[50]    train-rmse:37.44729     valid-rmse:36.37237     train-r2:-7.46666
valid-r2:-8.40428
[60]    train-rmse:31.14749     valid-rmse:30.01874     train-r2:-4.85757
valid-r2:-5.40571
[70]    train-rmse:26.08662     valid-rmse:24.90890     train-r2:-3.10872
valid-r2:-3.41053
[80]    train-rmse:22.04639     valid-rmse:20.83274     train-r2:-1.93458
valid-r2:-2.08514
[90]    train-rmse:18.84403     valid-rmse:17.60281     train-r2:-1.14397
valid-r2:-1.20265
[100]   train-rmse:16.33630     valid-rmse:15.08463     train-r2:-0.61131
valid-r2:-0.61753
[110]   train-rmse:14.40370     valid-rmse:13.14924     train-r2:-0.25262
valid-r2:-0.22909
[120]   train-rmse:12.92868     valid-rmse:11.69346     train-r2:-0.00921
valid-r2:0.02800
[130]   train-rmse:11.80810     valid-rmse:10.62254     train-r2:0.15816
valid-r2:0.19788
[140]   train-rmse:10.98601     valid-rmse:9.85892      train-r2:0.27130
valid-r2:0.30906
[150]   train-rmse:10.37397     valid-rmse:9.33236      train-r2:0.35023
valid-r2:0.38089
[160]   train-rmse:9.92029      valid-rmse:8.97081      train-r2:0.40582
valid-r2:0.42794
[170]   train-rmse:9.59072      valid-rmse:8.72660      train-r2:0.44464
valid-r2:0.45866
[180]   train-rmse:9.34406      valid-rmse:8.56794      train-r2:0.47284
valid-r2:0.47816
```

```
[190]   train-rmse:9.15732      valid-rmse:8.46753      train-r2:0.49370
valid-r2:0.49032
[200]   train-rmse:9.01338      valid-rmse:8.40309      train-r2:0.50949
valid-r2:0.49805
[210]   train-rmse:8.90767      valid-rmse:8.36135      train-r2:0.52093
valid-r2:0.50303
[220]   train-rmse:8.81830      valid-rmse:8.33586      train-r2:0.53049
valid-r2:0.50605
[230]   train-rmse:8.76528      valid-rmse:8.31968      train-r2:0.53612
valid-r2:0.50797
[240]   train-rmse:8.71715      valid-rmse:8.31027      train-r2:0.54120
valid-r2:0.50908
[250]   train-rmse:8.67918      valid-rmse:8.30535      train-r2:0.54519
valid-r2:0.50966
[260]   train-rmse:8.64532      valid-rmse:8.30037      train-r2:0.54873
valid-r2:0.51025
[270]   train-rmse:8.60983      valid-rmse:8.30106      train-r2:0.55243
valid-r2:0.51017
[280]   train-rmse:8.57386      valid-rmse:8.30142      train-r2:0.55616
valid-r2:0.51012
[290]   train-rmse:8.55184      valid-rmse:8.30257      train-r2:0.55844
valid-r2:0.50999
[300]   train-rmse:8.51677      valid-rmse:8.30632      train-r2:0.56205
valid-r2:0.50954
Stopping. Best iteration:
[259]   train-rmse:8.64905      valid-rmse:8.30028      train-r2:0.54834
valid-r2:0.51026
```

[14]:

```
#Predict your test_df values using xgboost

p_test = clf.predict(d_test)

sub = pd.DataFrame()
sub['ID'] = id_test
sub['y'] = p_test
sub.to_csv('xgb.csv', index=False)

sub.head()
```

[14]:

```
   ID           y
0   1   82.811882
1   2   97.109688
2   3   83.692238
3   4   77.155693
4   5  111.890274
```