trunSVD_xgboost_shap_windows

June 11, 2023

[ ]:
```
import warnings

warnings.filterwarnings("ignore",category=DeprecationWarning)
```

[1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
import shap
```

[2]:
```
# Load a local dataset
# Replace 'path_to_your_dataset' with the actual path to your local dataset
df = pd.read_excel(r'C:\Users\huangchuhuan\Downloads\    .xlsx', header = 0,␣
 ↪skiprows = [0])
# focus on
df = df.dropna(subset = ['   '])
# ignore rows with nan
df = df.dropna(axis=0)
# convert % symbol to real values
for column in df.columns:
    if df[column].dtype == np.object:
        if df[column].str.contains('%').any():
            df[column] = df[column].str.rstrip('%').astype(float)*100.0/(100.
 ↪0**2)
```

```
`np.object` is a deprecated alias for the builtin `object`. To silence this
warning, use `object` by itself. Doing this will not modify any behavior and is
safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

[3]:
```
# Select a random sample of 100 rows
df_sample = df.sample(n=100, random_state=1)
```

[4]:
```
df_sample.head()
```

[4]:
```
                                                          \n(     )  \
4413  040016.OF                         2010-05-11        2018-11-12
2098  008208.OF                         2019-12-19        2019-12-19
4659  163807.OF                 A      2009-04-03        2015-05-28
1497  005260.OF   , ,          A      2017-12-15        2017-12-15
4435  070002.OF                          2003-07-09        2018-12-04

     Wind        ( )              …     .4    .11            \
4413         6.315900              0.0  …    Beta   Beta
2098        12.784945              0.0  …    Beta   Beta
4659        34.999295             0.0  …    Beta   Beta
1497         0.321281              0.0  …    Beta   Beta
4435        30.490955              0.0  …    Beta    Beta



4413              ,  ,          0
2098              ,  ,          0
4659              ,  ,          0
1497              ,  ,          0
4435              ,  ,

[5 rows x 83 columns]
```

[5]:
```
# Autonomously choose a numerical variable as the target
# Here we choose 'ROE_TTM%' as an example
y = df_sample[['ROE_TTM%']]
# Drop the target variable from the dataframe
df_sample = df_sample.drop('ROE_TTM%', axis=1)
```

[6]:
```
# Get lists of numerical and categorical columns
num_cols = [col for col in df_sample.columns if df_sample[col].dtype in␣
 ↪['int64', 'float64']]
cat_cols = [col for col in df_sample.columns if df_sample[col].dtype ==␣
 ↪'object']
```

[7]:
```
# Convert all categorical columns to string type
df_sample[cat_cols] = df_sample[cat_cols].astype(str)

# Now define the preprocessor
preprocessor = make_column_transformer(
    (StandardScaler(), num_cols),  # standardize numerical features
    (OneHotEncoder(handle_unknown='ignore', sparse=False), cat_cols)  # one-hot␣
 ↪encode categorical features
)
```

```
# Preprocess the data
X = preprocessor.fit_transform(df_sample)
```

[8]:
```
from sklearn.decomposition import TruncatedSVD
```

[9]:
```
num_cols = 16
num_rows = 100
```

[10]:
```
data = [[np.nan]*num_cols for _ in range(num_rows)]
train_score_df = pd.DataFrame(data)
test_score_df = pd.DataFrame(data)
```

[11]:
```
for i in range(4,num_cols+4):
    for j in range(num_rows):
        # Apply TruncatedSVD
        svd = TruncatedSVD(n_components=i)  # specify the number of components,␣
 ↪can be adjusted based on your needs
        X_svd = svd.fit_transform(X)
        # Split the data into training and test sets
        X_train, X_test, y_train, y_test = train_test_split(X_svd, y,␣
 ↪test_size=0.2, random_state=42)

        # Train an XGBoost regressor
        xgb = XGBRegressor(objective='reg:squarederror', random_state=42)
        xgb.fit(X_train, y_train)

        # Evaluate the model
        train_score_df.iat[j,i-4] = xgb.score(X_train, y_train)
        test_score_df.iat[j,i-4] = xgb.score(X_test, y_test)
```

[12]:
```
test_score_df.head(10)
```

[12]:
```
          0         1         2         3         4         5         6  \
0  0.399332  0.707178  0.731261  0.683504  0.759852  0.656395  0.637740
1  0.401846  0.712542  0.755513  0.680714  0.757530  0.645657  0.588085
2  0.401846  0.707178  0.760670  0.671762  0.758432  0.666944  0.607315
3  0.399332  0.708568  0.757364  0.675555  0.759756  0.666395  0.634048
4  0.401846  0.709355  0.757363  0.663761  0.759608  0.635076  0.659257
5  0.401846  0.707178  0.756860  0.675865  0.756873  0.669821  0.521219
6  0.399332  0.707178  0.757364  0.687093  0.757052  0.651431  0.625772
7  0.401846  0.707178  0.755915  0.675381  0.757320  0.654603  0.588539
8  0.401846  0.707178  0.758674  0.683531  0.753802  0.668210  0.623493
9  0.401846  0.711161  0.755506  0.678314  0.757818  0.626702  0.640143

          7         8         9        10        11        12        13  \
0  0.678626  0.703874  0.669170  0.641379  0.491338  0.467539  0.476261
1  0.676284  0.706438  0.645756  0.667462  0.478638  0.467924  0.489849
```

```
2  0.707286  0.667478  0.742182  0.636922  0.474465  0.502289  0.460614
3  0.686651  0.659073  0.684626  0.655692  0.498198  0.462394  0.469451
4  0.651343  0.749167  0.646477  0.639052  0.478371  0.458417  0.469359
5  0.678963  0.638780  0.671207  0.639060  0.481431  0.460126  0.491408
6  0.687651  0.660770  0.649684  0.715158  0.460258  0.487854  0.493682
7  0.634687  0.663398  0.644066  0.668449  0.481339  0.454811  0.486459
8  0.730357  0.737300  0.681620  0.735028  0.494197  0.496407  0.470187
9  0.683182  0.731398  0.657197  0.653669  0.490023  0.453435  0.528558

         14        15
0  0.489494  0.680423
1  0.477648  0.676704
2  0.467889  0.653162
3  0.553708  0.675854
4  0.447270  0.684364
5  0.482344  0.628213
6  0.474138  0.683142
7  0.546508  0.555334
8  0.466828  0.541357
9  0.480619  0.693483
```

[13]:
```
import matplotlib.pyplot as plt
```

[14]:
```
col_names = {}
for column in test_score_df.columns:
    print(type(column))
    col_names[column] = column+4
test_score_df = test_score_df.rename(col_names, axis=1)
```

```
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
```

[15]:
```
print(col_names)
```

```
{0: 4, 1: 5, 2: 6, 3: 7, 4: 8, 5: 9, 6: 10, 7: 11, 8: 12, 9: 13, 10: 14, 11: 15,
12: 16, 13: 17, 14: 18, 15: 19}
```

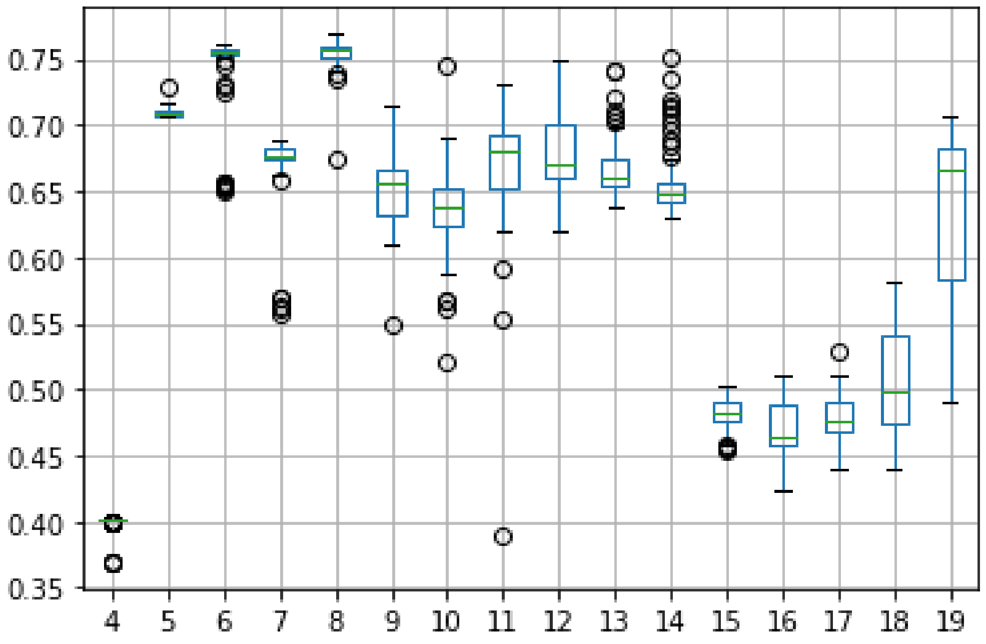
[16]:
```
test_score_df.head()
```

[16]:
```
          4         5         6         7         8         9        10  \
0  0.399332  0.707178  0.731261  0.683504  0.759852  0.656395  0.637740
1  0.401846  0.712542  0.755513  0.680714  0.757530  0.645657  0.588085
2  0.401846  0.707178  0.760670  0.671762  0.758432  0.666944  0.607315
3  0.399332  0.708568  0.757364  0.675555  0.759756  0.666395  0.634048
4  0.401846  0.709355  0.757363  0.663761  0.759608  0.635076  0.659257

         11        12        13        14        15        16        17  \
0  0.678626  0.703874  0.669170  0.641379  0.491338  0.467539  0.476261
1  0.676284  0.706438  0.645756  0.667462  0.478638  0.467924  0.489849
2  0.707286  0.667478  0.742182  0.636922  0.474465  0.502289  0.460614
3  0.686651  0.659073  0.684626  0.655692  0.498198  0.462394  0.469451
4  0.651343  0.749167  0.646477  0.639052  0.478371  0.458417  0.469359

         18        19
0  0.489494  0.680423
1  0.477648  0.676704
2  0.467889  0.653162
3  0.553708  0.675854
4  0.447270  0.684364
```

[17]:
```
test_score_df.boxplot()
plt.show()
```



[18]:
```
def get_column_with_largest_mean(df):
    # Calculate the mean values of each column
    column_means = df.mean()

    # Get the index of the column with the largest mean value
    largest_mean_index = column_means.idxmax()

    # Check for ties by comparing variance values
    column_variances = df.var()
    tiebreaker_indices = []
    largest_variance = df.var()[largest_mean_index]

    # Find indices of columns with the same mean value as the largest mean␣
 ↪column
    for col in df.columns:
        if column_means[col] == column_means[largest_mean_index]:
            if df.var()[col] < largest_variance:
                largest_variance = df.var()[col]
                tiebreaker_indices = [col]
            elif df.var()[col] == largest_variance:
                tiebreaker_indices.append(col)

    # Check for tiebreaker indices
    if tiebreaker_indices:
        # Return the index of the column with the smallest variance and␣
 ↪smallest index
        return min(tiebreaker_indices)
    else:
        # Return the index of the column with the largest mean value
        return largest_mean_index
```

[19]:
```
n = get_column_with_largest_mean(test_score_df)
print(n)
```

```
8
```

[20]:
```
#Choosing based on rank of average

# Apply TruncatedSVD
svd = TruncatedSVD(n_components=n)  # specify the number of components, can be␣
 ↪adjusted based on your needs
X_svd = svd.fit_transform(X)
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_svd, y, test_size=0.2,␣
 ↪random_state=42)
```

```
# Train an XGBoost regressor
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)
xgb.fit(X_train, y_train)

# Evaluate the model
train_score = xgb.score(X_train, y_train)
test_score = xgb.score(X_test, y_test)

print(f"Train score: {train_score}")
print(f"Test score: {test_score}")
```

```
Train score: 0.9999999785500552
Test score: 0.7570926899283865
```

[21]:
```
shap.initjs()
```

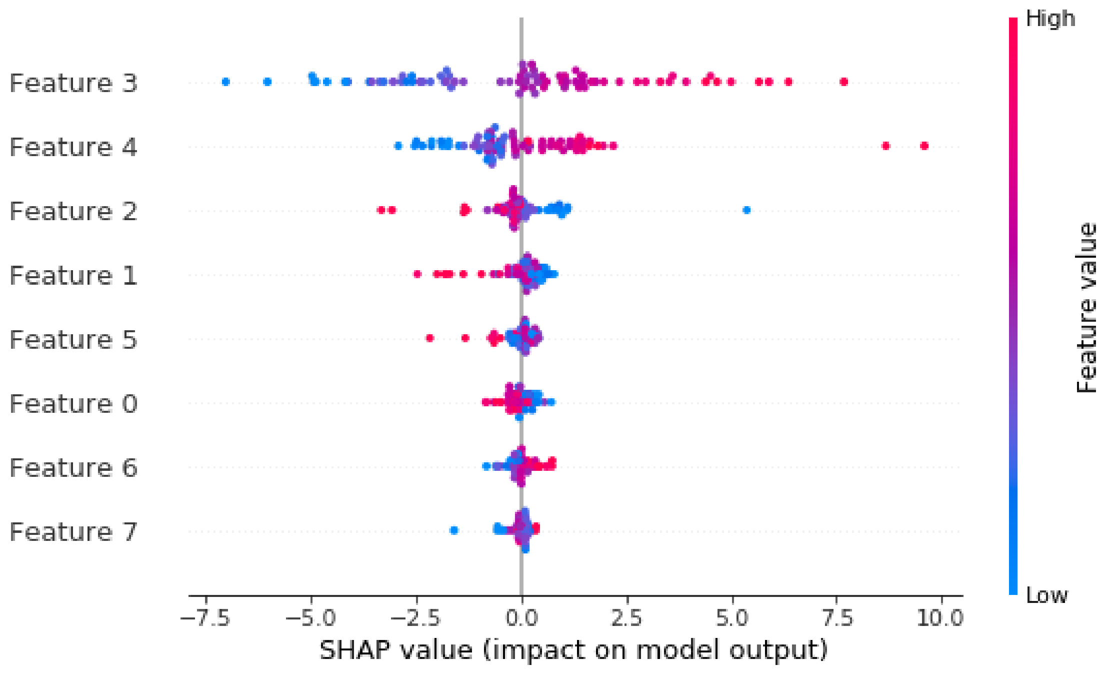
```
<IPython.core.display.HTML object>
```

[22]:
```
explainer = shap.TreeExplainer(xgb)
```

[23]:
```
shap_values = explainer.shap_values(X_train)
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

[24]:
```
shap.summary_plot(shap_values,X_train)
```


Feature 3
Feature 4
Feature 2
Feature 1
Feature 5
Feature 0
Feature 6
Feature 7
SHAP value (impact on model output)
High
Feature value
Low