Short Term XgBoost Model

Summary: In this code we shall build and test a short term XgBoost Model using Technical Indicators

In [1]:

```
# Import required libraries
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import KFold, GridSearchCV
from itertools import cycle
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.metrics import balanced_accuracy_score, make_scorer, classification_re
from sklearn import metrics
import os
import ta
import pickle
from scipy import interp
np.random.seed(0)
```

In [2]:

```
# User defined names
index = "Gold"
filename = index+"_hurst_segment_dependent.csv"
date_col = "Date"

# Declare the hyper-parameters for grid search
max_depth = [4, 6]
min_child_weight = [10, 20]
gamma = [0, 0.1]
subsample = [0.8]
colsample_bytree = [0.8]
scale_pos_weight = [1]
learning_rate = [0.05, 0.1]
n_estimators = [200]
reg_alpha = [1e-5, 0.1, 1, 100]
reg_lambda = [0, 0.001, 0.01, 0.1]
```

In [3]:

```
# Get current working directory
mycwd = os.getcwd()
print(mycwd)
```

```
C:\Users\sidhu\Downloads\Course 10 Capstone Project\Trading Strategy Development\Dev
\Gold\Codes
```

In [4]:

```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Data")
```

In [5]:

```
# Read the data
df = pd.read_csv(filename, index_col=date_col)
df.index = pd.to_datetime(df.index)
df.head()
```

Out[5]:

	High	Low	Open	Close	Volume	Adj Close	hurst_100	h
Date								
2010-12-31	138.770004	137.720001	137.779999	138.720001	9219800	138.720001	NaN	
2011-01-03	139.000000	137.880005	138.669998	138.000000	11510200	138.000000	NaN	
2011-01-04	136.279999	134.160004	136.240005	134.750000	26154300	134.750000	NaN	
2011-01-05	134.679993	133.100006	133.500000	134.369995	16700900	134.369995	NaN	
2011-01-06	134.380005	133.139999	134.050003	133.830002	15965300	133.830002	NaN	

5 rows × 29 columns

Functions

In [6]:

```
def Split_data_XY(df, dv):
    """
    Given a dataset returns two dataframes, X-Dataframe and y-dataframe
    """
    X_df = df.drop([dv], axis=1)
    y_df = df[dv]
    y_labelizer = label_binarize(y_df, classes=[-1, 0, 1])
    return X_df, y_df, y_labelizer
```

In [7]:

```
def Get_Max_Discretevar(df, var, window=10):
    """
    Get maximum value on rolling basis for the variable
    """
    df[var+"_max"+str(window)] = df[var].rolling(window=window).max()
    return df
```

```
In [8]: def Get_SMA_Continousvar(df, var, window=10):
            """
            Get SMA for continous variable
            """
            df[var+"_sma"+str(window)] = df[var].rolling(window=window).mean()
            return df
```

```
In [9]: def Get_Ratio_Continousvar(df, var, window=10):
            """
            Get Ratio for continous variable Min/Max
            """
            df[var+"_ratio_minmax"+str(window)] = np.where(np.abs(df[var].rolling(window=wi
                                                           df[var].rolling(window=window).m
                                                           df[var].rolling(window=window).m
            return df
```

```
In [10]: def Get_std_Continousvar(df, var, window=30):
             """
             Get Ratio for continous variable Min/Max
             """
             df[var+"_std"+str(window)] = df[var].rolling(window=window).std()
             return df
```

```
In [11]: def Generate_Predicted_df(X_train, y_train, X_test, y_test, clf):
             """
             Generates Pandas dataframe with predicted values and other columns for P&L anal
             """
             # Train Sample
             df_train = pd.DataFrame(y_train)
             df_train['Predicted'] = clf.predict(X_train)
             df_train['Adj Close'] = X_train['Adj Close']
             df_train['Open'] = X_train['Open']
             df_train['DVT STD'] = X_train['DVT STD']
             df_train["Sample"] = "Train"
             # Test Sample
             df_test = pd.DataFrame(y_test)
             df_test['Predicted'] = clf.predict(X_test)
             df_test['Adj Close'] = X_test['Adj Close']
             df_test['Open'] = X_test['Open']
             df_test['DVT STD'] = X_test['DVT STD']
             df_test['Sample'] = "Test"
             df = df_train.append(df_test)
             return df
```

Feature Engineering

In [12]:

```
# Add all technical features
df = ta.add_all_ta_features(df, open="Open", high="High", low="Low", close="Adj Clo
```

In [13]:

```
# Max variable list
max_vars = ['volatility_bbhi', 'volatility_bbli', 'volatility_kchi', 'volatility_kc
            'trend_psar_down_indicator']
for i in range(0, len(max_vars)):
    df = Get_Max_Discretevar(df, max_vars[i], 10)
```

In [14]:

```
# SMA variable list
sma_vars = ['volume_adi', 'volume_obv', 'volume_cmf', 'volume_fi', 'volume_mfi', 'v
            'volume_vpt', 'volume_nvi', 'volume_vwap', 'volatility_atr', 'volatilit
            'volatility_bbl', 'volatility_bbw', 'volatility_bbp', 'volatility_kcc',
            'volatility_kcw', 'volatility_kcp', 'volatility_dcl', 'volatility_dch',
            'volatility_dcp', 'volatility_ui', 'trend_macd', 'trend_macd_signal', '
            'trend_sma_slow', 'trend_ema_fast', 'trend_ema_slow', 'trend_adx', 'tre
            'trend_vortex_ind_pos', 'trend_vortex_ind_neg', 'trend_vortex_ind_diff'
            'trend_cci', 'trend_dpo', 'trend_kst', 'trend_kst_sig', 'trend_kst_diff
            'trend_ichimoku_base', 'trend_ichimoku_a', 'trend_ichimoku_b', 'trend_v
            'trend_visual_ichimoku_b', 'trend_aroon_up', 'trend_aroon_down', 'trend
            'momentum_rsi', 'momentum_stoch_rsi', 'momentum_stoch_rsi_k', 'momentum
            'momentum_uo', 'momentum_stoch', 'momentum_stoch_signal', 'momentum_wr'
            'momentum_roc', 'momentum_ppo', 'momentum_ppo_signal', 'momentum_ppo_hi
            'others_cr']
for i in range(0, len(sma_vars)):
    df = Get_SMA_Continousvar(df, sma_vars[i], window=10)
```

In [15]:

```
# Ratio of Min Max variables
for i in range(0, len(sma_vars)):
    df = Get_Ratio_Continousvar(df, sma_vars[i], window=10)
```

In [16]:

```
# Ratio of std variables
for i in range(0, len(sma_vars)):
    df = Get_std_Continousvar(df, sma_vars[i], window=30)
```

In [17]:

```
# Drop two features
df = df.drop(['trend_psar_down', 'trend_psar_up'], axis=1)
df = df[df['hurst_150'] > 0]
df.shape
```

Out[17]:

```
(2369, 341)
```

In [18]:

```
# Drop rows with null values
df.dropna(inplace=True)
df.shape
```

Out[18]:

```
(2119, 341)
```

Divide the data in Segments

In [19]:

```
df['Segment'].value_counts()
```

Out[19]:

```
Mean Reverting    1278
Trending           841
Name: Segment, dtype: int64
```

In [20]:

```
# Break dataset into three segments
df_MeanReverting = df[df['Segment'] == "Mean Reverting"]
df_Trending = df[df['Segment'] == "Trending"]
```

In [21]:

```
# Drop Segment variable from all datasets
df.drop("Segment", axis=1, inplace=True)
df_MeanReverting.drop("Segment", axis=1, inplace=True)
df_Trending.drop("Segment", axis=1, inplace=True)
```

Mean Reverting Dataset

In [22]:

```
# Divide dataset into Train and Test Sample. (5 Fold CV will be used for validation
df_MeanReverting_Train = df_MeanReverting[df_MeanReverting.index.year <= 2018]
df_MeanReverting_Test = df_MeanReverting[df_MeanReverting.index.year > 2018]
print("Train Sample: ", df_MeanReverting_Train.shape)
print("Test Sample: ", df_MeanReverting_Test.shape)
```

```
Train Sample:  (1197, 340)
Test Sample:  (81, 340)
```

Trending Dataset

In [23]:

```
df_Trending_Train = df_Trending[df_Trending.index.year <= 2018]
df_Trending_Test = df_Trending[df_Trending.index.year > 2018]
print("Train Sample: ", df_Trending_Train.shape)
print("Test Sample: ", df_Trending_Test.shape)
```

```
Train Sample:  (417, 340)
Test Sample:  (424, 340)
```

Whole Dataset

In [24]:

```
df_Train = df[df.index.year <= 2018]
df_Test = df[df.index.year > 2018]
print("Train Sample: ", df_Train.shape)
print("Test Sample: ", df_Test.shape)
```

```
Train Sample:  (1614, 340)
Test Sample:  (505, 340)
```

XgBoost Model Grid Search

In [25]:

```
# Grid
grid = {'max_depth': max_depth,
        'min_child_weight': min_child_weight,
        'gamma': gamma,
        'subsample': subsample,
        'colsample_bytree': colsample_bytree,
        'scale_pos_weight': scale_pos_weight,
        'learning_rate': learning_rate,
        'n_estimators':n_estimators,
        'reg_alpha':reg_alpha,
        'reg_lambda':reg_lambda}
```

In [26]:

```
# XgBoost Model
scoring = {'Accuracy':make_scorer(balanced_accuracy_score)}
kfold = KFold(n_splits=3)
clf = XGBClassifier( objective= 'multi:softprob', num_classes=3, nthread=4, scale_p
                    eval_metric='mlogloss')
# Define grid search
grid = GridSearchCV(estimator = clf, param_grid=grid, cv=kfold, scoring=scoring, re
```

Whole Dataset

In [27]:

```
# Get X, Y variables
X_train, y_train, y_train_label = Split_data_XY(df_Train, 'Target')
X_test, y_test, y_test_label = Split_data_XY(df_Test, 'Target')
```

In [28]:

```
# Fit the grid search model
model = grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 91.3min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 342.3min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 360.2min
[Parallel(n_jobs=-1)]: Done 768 out of 768 | elapsed: 384.7min finished
```

```
[04:07:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

In [29]:

```
# Get the best xgboost model based on Grid Search
best_xgboost = model.best_estimator_
best_xgboost
```

Out[29]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0.1, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.05, max_delta_step=0,
              max_depth=4, min_child_weight=20, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1e-05, reg_lambda=0.1,
              scale_pos_weight=1, seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [30]:

```
# XgBoost model selected using Grid search
clf = best_xgboost
clf.fit(X_train, y_train)
```

```
[04:07:13] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

Out[30]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0.1, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.05, max_delta_step=0,
              max_depth=4, min_child_weight=20, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1e-05, reg_lambda=0.1,
              scale_pos_weight=1, seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [31]:

```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Models")
```

In [32]:

```python
# Save the model
with open('whole_dataset'+str(index)+'_xgboost_model.pkl', 'wb') as f:
    pickle.dump(clf, f)

# load it
with open('whole_dataset'+str(index)+'_xgboost_model.pkl', 'rb') as f:
    clf = pickle.load(f)
```

In [33]:

```python
y_train_out = clf.predict(X_train)
print(classification_report(y_train, y_train_out))
```

```
              precision    recall  f1-score   support

          -1       0.96      0.88      0.92       309
           0       0.93      0.97      0.95       817
           1       0.96      0.94      0.95       488

    accuracy                           0.95      1614
   macro avg       0.95      0.93      0.94      1614
weighted avg       0.95      0.95      0.95      1614
```

In [34]:

```python
# Confusion Matrix Train Sample
print("Train Sample Confusion Matrix")
pd.crosstab(y_train, y_train_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Train Sample Confusion Matrix
```

Out[34]:

Predicted	**-1**	**0**	**1**
Actual			
-1	273	34	2
0	5	796	16
1	5	25	458

In [35]:

```python
y_test_out = clf.predict(X_test)
print(classification_report(y_test, y_test_out))
```

```
              precision    recall  f1-score   support

          -1       0.21      0.41      0.28        76
           0       0.56      0.57      0.57       235
           1       0.40      0.24      0.30       194

    accuracy                           0.42       505
   macro avg       0.39      0.41      0.38       505
weighted avg       0.45      0.42      0.42       505
```

In [36]:

```python
# Confusion Matrix Train Sample
print("Test Sample Confusion Matrix")
pd.crosstab(y_test, y_test_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Test Sample Confusion Matrix
```

Out[36]:

Predicted	-1	0	1
Actual			
-1	31	28	17
0	46	135	54
1	69	78	47

In [37]:

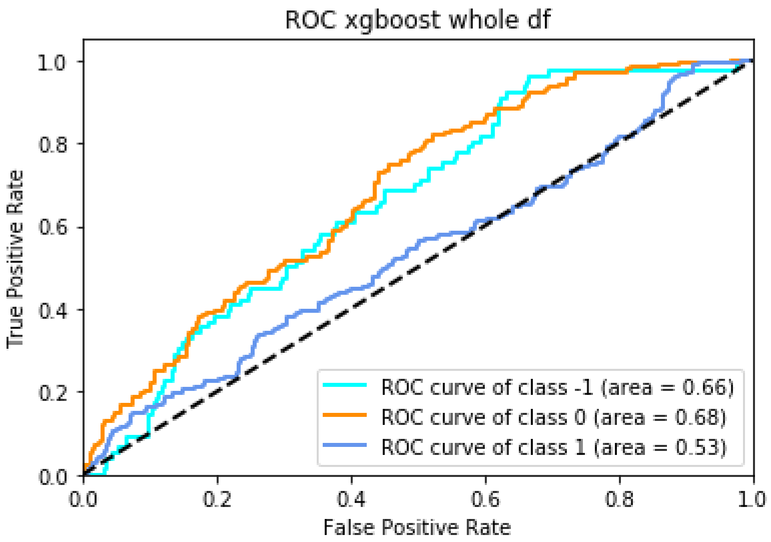
```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Images")
```

In [38]:

```
y_score = clf.predict_proba(X_test)
n_classes = 3
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_label[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [39]:

```
lw = 2

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

classes = [-1,0,1]
plt.figure()
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(classes[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC xgboost whole df')
plt.legend(loc="lower right")
plt.savefig("xgboost Whole df test" + str(index)+ " ROC curve"+'.png')
plt.show()
```



```
In [40]: # Change to data directory
         os.chdir("..")
         os.chdir(str(os.getcwd()) + "\\Data")
```

```
In [41]: df_out = Generate_Predicted_df(X_train, y_train, X_test, y_test, clf)
         df_out.to_csv('whole_dataset'+str(index)+'_xgboost_model.csv', index=True)
```

Trending Dataset

```
In [42]: # Get X, Y variables
         X_train, y_train, y_train_label = Split_data_XY(df_Trending_Train, 'Target')
         X_test, y_test, y_test_label = Split_data_XY(df_Trending_Test, 'Target')
```

```
In [43]: # Fit the grid search model
         model = grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:   25.4s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 768 out of 768 | elapsed:  7.3min finished
[04:14:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

In [44]:

```
# Get the best xgboost model based on Grid Search
best_xgboost = model.best_estimator_
best_xgboost
```

Out[44]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=4, min_child_weight=10, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1, reg_lambda=0.001,
              scale_pos_weight=1, seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [45]:

```
# XgBoost model selected using Grid search
clf = best_xgboost
clf.fit(X_train, y_train)
```

```
[04:14:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

Out[45]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=4, min_child_weight=10, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1, reg_lambda=0.001,
              scale_pos_weight=1, seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [46]:

```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Models")
```

In [47]:

```
# Save the model
with open('Trending_dataset'+str(index)+'_xgboost_model.pkl', 'wb') as f:
    pickle.dump(clf, f)

# load it
with open('Trending_dataset'+str(index)+'_xgboost_model.pkl', 'rb') as f:
    clf = pickle.load(f)
```

In [48]:

```
y_train_out = clf.predict(X_train)
print(classification_report(y_train, y_train_out))
```

```
              precision    recall  f1-score   support

          -1       1.00      0.99      0.99        73
           0       1.00      1.00      1.00       221
           1       1.00      1.00      1.00       123

    accuracy                           1.00       417
   macro avg       1.00      1.00      1.00       417
weighted avg       1.00      1.00      1.00       417
```

In [49]:

```
# Confusion Matrix Train Sample
print("Train Sample Confusion Matrix")
pd.crosstab(y_train, y_train_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Train Sample Confusion Matrix
```

Out[49]:

Predicted	**-1**	**0**	**1**
Actual			
-1	72	1	0
0	0	221	0
1	0	0	123

In [50]:

```
y_test_out = clf.predict(X_test)
print(classification_report(y_test, y_test_out))
```

```
              precision    recall  f1-score   support

          -1       0.24      0.48      0.32        71
           0       0.55      0.61      0.58       198
           1       0.45      0.18      0.26       155

    accuracy                           0.43       424
   macro avg       0.41      0.42      0.39       424
weighted avg       0.46      0.43      0.42       424
```

In [51]:

```
# Confusion Matrix Train Sample
print("Test Sample Confusion Matrix")
pd.crosstab(y_test, y_test_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Test Sample Confusion Matrix
```

Out[51]:

Predicted	**-1**	**0**	**1**
Actual			
-1	34	25	12
0	55	121	22
1	54	73	28

In [52]:

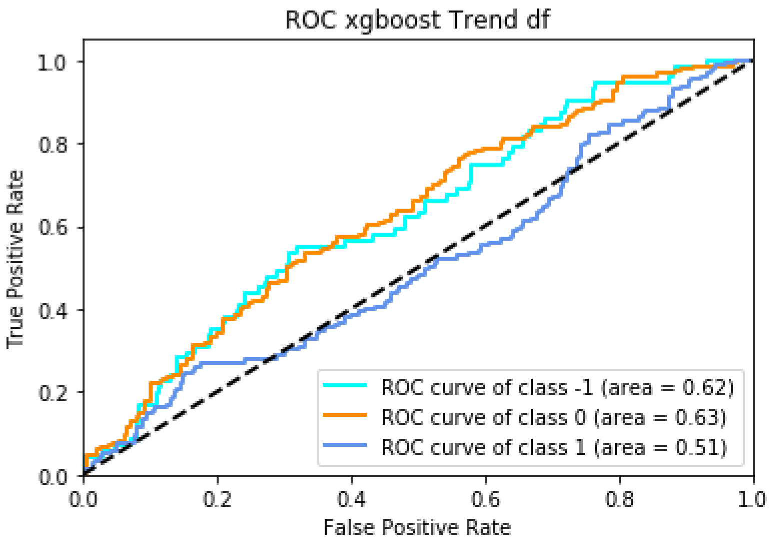
```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Images")
```

In [53]:

```
y_score = clf.predict_proba(X_test)
n_classes = 3
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_label[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [54]:

```
lw = 2

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

classes = [-1,0,1]
plt.figure()
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(classes[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC xgboost Trend df')
plt.legend(loc="lower right")
plt.savefig("xgboost Trending_dataset test" + str(index)+ " ROC curve"+'.png')
plt.show()
```



In [55]:
```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Data")
```

In [56]:
```
df_out = Generate_Predicted_df(X_train, y_train, X_test, y_test, clf)
df_out.to_csv('Trending_dataset'+str(index)+'_xgboost_model.csv', index=True)
```

Mean Reverting Dataset

In [57]:
```
# Get X, Y variables
X_train, y_train, y_train_label = Split_data_XY(df_MeanReverting_Train, 'Target')
X_test, y_test, y_test_label = Split_data_XY(df_MeanReverting_Test, 'Target')
```

In [58]:
```
# Fit the grid search model
model = grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 256 candidates, totalling 768 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed: 21.5min
[Parallel(n_jobs=-1)]: Done 768 out of 768 | elapsed: 37.4min finished
```

```
[04:52:09] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

In [59]:

```
# Get the best xgboost model based on Grid Search
best_xgboost = model.best_estimator_
best_xgboost
```

Out[59]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0.1, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.05, max_delta_step=0,
              max_depth=6, min_child_weight=10, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1, reg_lambda=0.1, scale_pos_weight=1,
              seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [60]:

```
# XgBoost model selected using Grid search
clf = best_xgboost
clf.fit(X_train, y_train)
```

```
[04:52:18] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/sr
c/learner.cc:541:
Parameters: { num_classes, scale_pos_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 thi
s
  verification. Please open an issue if you find above cases.
```

Out[60]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, eval_metric='mlogloss',
              gamma=0.1, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.05, max_delta_step=0,
              max_depth=6, min_child_weight=10, missing=nan,
              monotone_constraints='()', n_estimators=200, n_jobs=4, nthread=4,
              num_classes=3, num_parallel_tree=1, objective='multi:softprob',
              random_state=27, reg_alpha=1, reg_lambda=0.1, scale_pos_weight=1,
              seed=27, subsample=0.8, tree_method='exact',
              use_label_encoder=True, ...)
```

In [61]:

```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Models")
```

In [62]:

```
# Save the model
with open('MeanReverting_dataset'+str(index)+'_xgboost_model.pkl', 'wb') as f:
    pickle.dump(clf, f)

# load it
with open('MeanReverting_dataset'+str(index)+'_xgboost_model.pkl', 'rb') as f:
    clf = pickle.load(f)
```

In [63]:

```
y_train_out = clf.predict(X_train)
print(classification_report(y_train, y_train_out))
```

```
              precision    recall  f1-score   support

          -1       1.00      0.99      0.99       236
           0       0.99      1.00      1.00       596
           1       1.00      0.99      1.00       365

    accuracy                           1.00      1197
   macro avg       1.00      1.00      1.00      1197
weighted avg       1.00      1.00      1.00      1197
```

In [64]:

```
# Confusion Matrix Train Sample
print("Train Sample Confusion Matrix")
pd.crosstab(y_train, y_train_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Train Sample Confusion Matrix
```

Out[64]:

Predicted	**-1**	**0**	**1**
Actual			
-1	234	2	0
0	0	596	0
1	1	1	363

In [65]:

```
y_test_out = clf.predict(X_test)
print(classification_report(y_test, y_test_out))
```

```
              precision    recall  f1-score   support

          -1       0.22      0.80      0.35         5
           0       0.62      0.76      0.68        37
           1       0.72      0.33      0.46        39

    accuracy                           0.56        81
   macro avg       0.52      0.63      0.50        81
weighted avg       0.65      0.56      0.55        81
```

In [66]:

```
# Confusion Matrix Train Sample
print("Test Sample Confusion Matrix")
pd.crosstab(y_test, y_test_out, rownames=['Actual'], colnames=['Predicted'])
```

```
Test Sample Confusion Matrix
```

Out[66]:

Predicted	-1	0	1
Actual			
-1	4	0	1
0	5	28	4
1	9	17	13

In [67]:

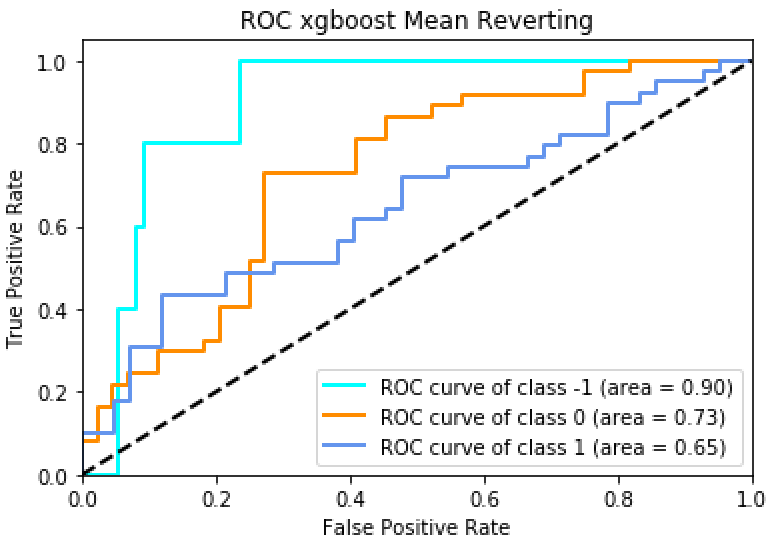
```
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Images")
```

In [68]:

```
y_score = clf.predict_proba(X_test)
n_classes = 3
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_label[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
```

In [69]:

```
lw = 2

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

classes = [-1,0,1]
plt.figure()
colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(classes[i], roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC xgboost Mean Reverting')
plt.legend(loc="lower right")
plt.savefig("xgboost MeanReverting_dataset test" + str(index)+ " ROC curve"+'.png')
plt.show()
```



In [70]:

```python
# Change to data directory
os.chdir("..")
os.chdir(str(os.getcwd()) + "\\Data")
```

In [71]:

```python
df_out = Generate_Predicted_df(X_train, y_train, X_test, y_test, clf)
df_out.to_csv('MeanReverting_dataset'+str(index)+'_xgboost_model.csv', index=True)
```