Data Science And Machine Learning

Xiaodong YANG
yangxiaodong993993@gmail.com

A Report for Exam Three
Certificate in Quantitative Finance
June 2022 Program

November 2022

Contents

1 Maths **1**
1.1 The Trade-off of Bias and Variance . . . 1
1.2 Gaussian RBF kernel . . . 1
1.3 Loss Functions for Decision Trees . . . 1

2 Models **2**
2.1 Produce a Model to Predict Positive Moves using SVM . . . 2
2.2 Tune Hyperparameters for the Estimator and the Best Model . . . 4
2.3 Evaluating the Prediction Quality . . . 7

Appendix **10**
Python Code . . . 10

1 Maths

1.1 The Trade-off of Bias and Variance

$$E[(\hat{\beta} - \beta)^2] = Var[\hat{\beta}]^2 + (E[\hat{\beta}] - \beta)^2 \tag{1}$$

(a). Yes. We can get the smallest MSE when the error is distributed normally.

(b). The MSE measures the model error for predictions. The irreducible error is the minimum lower bound for the test MSE, which can be written as $Var(\varepsilon)$ ($\varepsilon \sim N[0, 1]$).

1.2 Gaussian RBF kernel

$$k(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{2\sigma}\right) \tag{2}$$

Answer: (a)

For z_1 and x, we have

$$k(z_1, x) = \exp\left(-\frac{||z_1 - x||^2}{2\sigma}\right)$$

Assuming $||z_1 - x|| \sim 0$

$$k(z_1, x) \approx e^0 = 1$$

For z_2 and x, we have

$$k(z_2, x) = \exp\left(-\frac{||z_2 - x||^2}{2\sigma}\right)$$

Assuming $||z_2 - x|| \sim \infty$

$$k(z_2, x) \approx e^{-\infty} = 0$$

1.3 Loss Functions for Decision Trees

(a) Gini Impurity Index

Gini Impurity Index measures the diversity of a dataset. It can be used to check if the tree is bad or good. Specifically speaking, the good trees are defined each leaf contains a single class with low Gini Impurity Index. Assuming the dataset has n classes, and the proportions are presented as $p_1, p_2, \cdots, p_n$

$$\text{Gini Impurity Index} = 1 - \sum_{i=1}^{n} p_i^2 \tag{3}$$

(b) Entropy

The entropy of a decision tree is the average level of information inherent to the variable's possible outcomes. The lower entropy means purer leaf in decision tree.

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{4}$$

(c) Gradient Descent

Gradient Descent is an efficient method to find the optimal solution through the differentiable and convex loss function. Stochastic Gradient Descent is another branch using a random point to start the Gradient Descent and compute the negative gradient. Repeating the steps until we found the optimal value.

2 Models

2.1 Produce a Model to Predict Positive Moves using SVM

Krystal Biotech (KRYS) operates as a biopharmaceutical company, which is the leader in redosable gene therapy. Its market cap is about 1.973B, making it a small-cap instrument. The overview of Krystal Biotech is shown by Figure 1.

Figure 1: Krystal Biotech Summary

Previous Close	**77.50**	Market Cap	**1.973B**
Open	**76.90**	Beta (5Y Monthly)	**0.87**
Bid	**76.64 x 900**	PE Ratio (TTM)	**N/A**
Ask	**119.93 x 900**	EPS (TTM)	**-4.48**
Day's Range	**74.56 - 77.75**	Earnings Date	**Nov 07, 2022**
52 Week Range	**38.86 - 102.99**	Forward Dividend & Yield	**N/A (N/A)**
Volume	**178,184**	Ex-Dividend Date	**N/A**
Avg. Volume	**165,546**	1y Target Est	**104.75**

Source: Yahoo Finance

The historical data from November 1st 2016 to October 31st 2022 is downloaded by YahooFinance package. Figure 2 shows the adjusted close price of Krystal Biotech. And its continuous daily returns is shown in Figure 3. We label the returns using 0 and 1. In particular, the threshold are set as 0.20%, meaning that the returns below the threshold are labeled as negative. The sign of return is labeled as 1 if the return is positive, otherwise 0. The label 1 accounts for 50.7% and the rest is label 0 about 49.3%. In addition, extremely small near-zero returns are dropped from the training sample. We calculate the volatility of the returns and drop the returns ($r_i \leq |\gamma|$), where $\gamma = 0.1 \times \sigma_{\text{Returns}}$. We get the 0.1 from several empirical results since the model with 0.1 performs better than other models. In practice, there are some factors to results in extremely small near-zero returns, such as transactions. These returns are considered as the noise in modelling.

Figure 2: Adjusted Close Price of Krystal Biotech

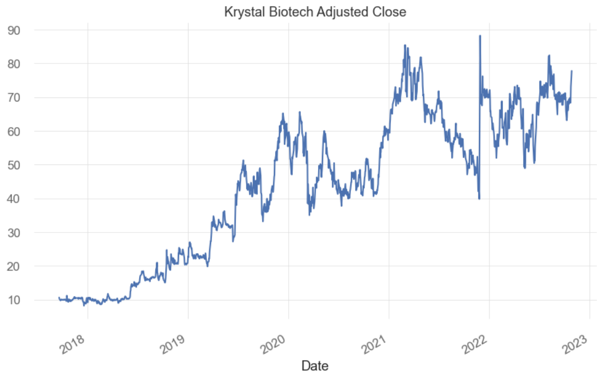


Figure 3: Daily Returns of Krystal Biotech

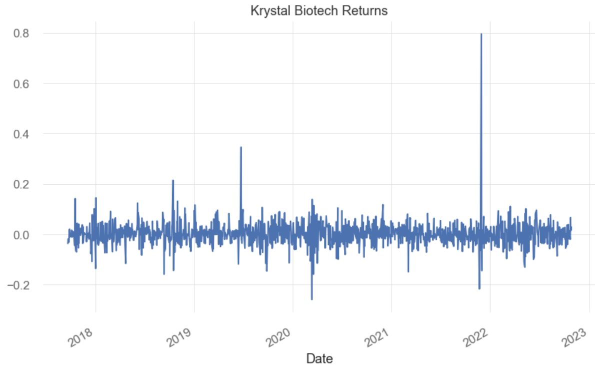


Table 1: Features List

Features	Formula
Open/Close Price	Open-Close
High/Low Price	High-Low
Sign of Return	$\text{sign}[\ln \frac{P_t}{P_{t-1}}]$
Past Returns	$r_{t-1}, r_{t-2}, \cdots$
Momentum	$P_t - P_{t-k}$
Sample Moving Average	$SMA_i = \frac{1}{n}\sum_{i=1}^{n-1} P_{t-i}$
Exponential Moving Average	$EMA_t = EMA_{t-1} + \alpha[P_t - EMA_{t-1}]$

Next, the features given by Table 1 are computed. All features are scaled by StandardScaler in scikitlearn package. The StandardScaler is utilised since it perform better than other scalers in classification or cluster algorithm. The summary of all features is shown by Figure 4 in which there are 16 features and 1,257 samples. We set the training size is 30% and the test size is 70%.

Figure 4: Features Summary

	count	mean	std	min	25%	50%	75%	max
pastRtn_1	1257.0	1.258606e-17	1.000398	-4.810285	-0.377374	-0.021748	0.375228	20.004916
pastRtn_2	1257.0	3.444606e-18	1.000398	-5.687859	-0.412591	-0.022506	0.398053	13.926027
pastRtn_5	1257.0	-6.712566e-18	1.000398	-4.684071	-0.447136	-0.032702	0.436844	8.414714
pastRtn_10	1257.0	-6.005980e-18	1.000398	-3.844993	-0.482820	-0.059360	0.472941	6.846952
mtm_1	1257.0	1.258606e-17	1.000398	-4.810285	-0.377374	-0.021748	0.375228	20.004916
mtm_10	1257.0	-6.005980e-18	1.000398	-3.844993	-0.482820	-0.059360	0.472941	6.846952
mtm_15	1257.0	-4.027539e-17	1.000398	-3.438295	-0.489632	-0.054655	0.480813	5.200970
sma_5	1257.0	5.672118e-16	1.000398	-1.677893	-1.012629	0.126141	0.844310	1.797443
sma_15	1257.0	3.870324e-16	1.000398	-1.656966	-1.007593	0.108798	0.809521	1.669271
sma_30	1257.0	4.545114e-16	1.000398	-1.630469	-0.995381	0.102296	0.818032	1.581627
ema_5	1257.0	-9.185616e-18	1.000398	-1.667928	-1.004796	0.106958	0.817408	1.691809
ema_10	1257.0	9.122023e-16	1.000398	-1.654986	-1.005578	0.092886	0.822356	1.621519
ema_20	1257.0	-5.204005e-16	1.000398	-1.632406	-0.991891	0.129542	0.860873	1.562467
ema_50	1257.0	1.009358e-15	1.000398	-1.574159	-1.017928	0.215055	0.931274	1.374784
openClose	1257.0	3.488768e-17	1.000398	-3.089395	-0.483076	-0.000826	0.476007	6.019177
highLow	1257.0	-1.999196e-16	1.000398	-1.538077	-0.699369	-0.116631	0.536907	8.934868

2.2 Tune Hyperparameters for the Estimator and the Best Model

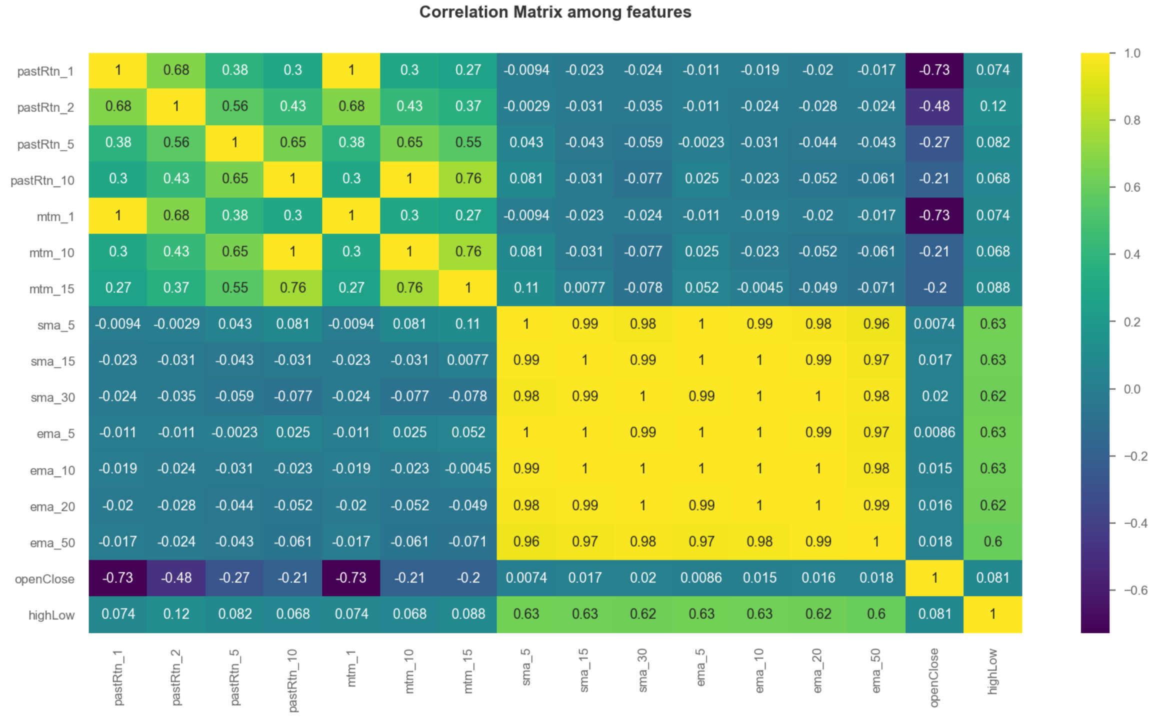
(a) Correlation Analysis

The first step is to analyse the correlation since colinear features, also called correlation bias affect the predictions in Machine Learning. The interplay between two variables with high correlation can reduce the importance of one of them. It will cause that we omit the important variables. We calculated the correlation of all features. From the Figure 5, we found there are high correlation among several variables shown by Table 2. And the correlation of these features is more than ± 0.8. We will not select the features with high correlation.

Table 2: Features with High Correlations

	Variables
1	pastRtn_1, mtm_1
2	pastRtn_10, mtm_10
3	sma_5, sma_15, sma_30, ema_5, ema_10, ema_20, ema_50

Figure 5: Correlation Matrix of All Features



The complete sample includes 16 features. We can get the model with high accuracy using all features to train the model. However, the too many features will slow the computing speed and cause overfitting for the model in sample. Hence, it is necessary to select the more important features to fit the model. Ridge regression and XGBoost are utilised in this study.

(b) Ridge Regularization

Ridge regression is usually used to avoid overfitting by shrink coefficients of variable. From empirical results, we found that "sma_5", "sma_15", "sma_30", and "ema_5" are significant in the ridge regression. This result will be utilised in further steps.

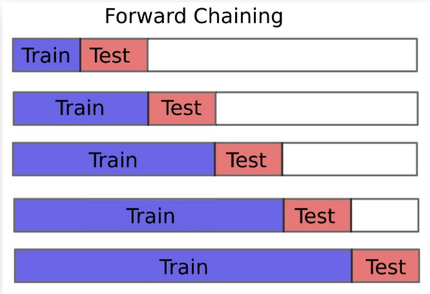
(c) XGBoost

The strength of the XGBoost is to provide the estimates of the most important features from all training models. In the XGBoost, we use the non-traditional cross validation. Specifically speaking, Forward Chaining method (shown by Figure 6) is employed in which the model is initially trained and tested with the same windows size. Besides, the training window increases in size for each subsequent fold, encompassing both the previous training and test data. The new window once again follows the training window but stays the same length. In addition, we set several parameters for Gridsearch, including learning rare, max-depth, min-child-weight, gamma, and the colsample-bytree. Then, we use the training data to fit the model and get the best parameters and the best score.

```
# The best parameters:
{'min_child_weight': 1, 'max_depth': 12, 'learning_rate': 0.15,
'gamma': 0.2, 'colsample_bytree': 0.3}
```

```
# The best score:
0.5649520461665961
```

Figure 6: Forward Chaining Method



Source: CQF Python Lab

Python code for this part is as follows

```
# Cross-validation
tscv = TimeSeriesSplit(n_splits=5, gap=1)
# Hyper parameter optimization
param_grid = {'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
              'max_depth': [3, 4, 5, 6, 8, 10, 12, 15],
              'min_child_weight': [1, 3, 5, 7],
              'gamma': [0.0, 0.1, 0.2 , 0.3, 0.4],
              'colsample_bytree': [0.3, 0.4, 0.5 , 0.7]}
```

Then, we use the training data to fit the model with the best parameters and get the scores of all features. The Figure 7 shows the order by importance of features in which the "pastRtn_1" and "pastRtn_5" are the most important features, while the "ema_5" and "ema_20" have the least importance. Figure 8 shows the first three important features are "pastRtn_1", "pastRtn_2", and "pastRtn_10".

Figure 7: Features Importance

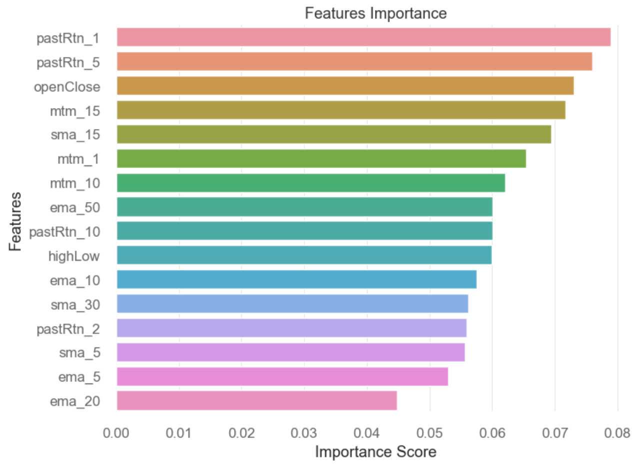
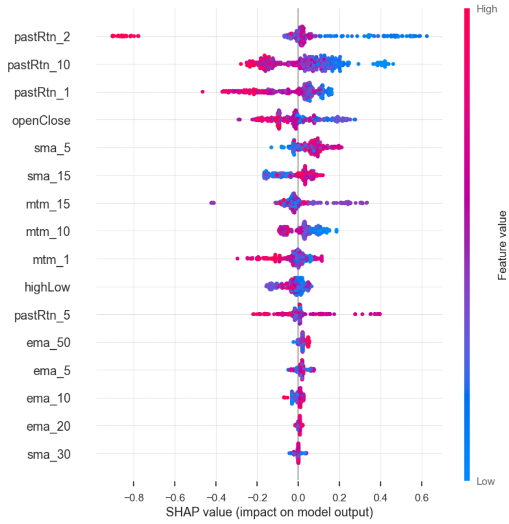


Figure 8: SHAP Values



It is widely known that Machine Learning models are often black boxes that makes their interpretation difficult. In contrast, the SHAP values provide the concise explanation and make it easier to comprehend. SHAP method, used to explain how each feature affects the model, and allows local and global analysis for the dataset and problem at hand. From the Figure 8, the "pastRtn_2" is the most important features. In addition, most red dots cluster the left side, meaning that higher "pastRtn_2" values have lower SHAP values.

In this study, we constructed two models. The first model has 3 features, which are "pastRtn_1", "pastRtn_10", and "sma_15". Another model are created by adding "pastRtn_2" and "sma_5" to the first model. They are shown in Table 3.

Table 3: The 2 Models

Model	Features
1	pastRtn_1, pastRtn_10, sma_15
2	pastRtn_1, pastRtn_10, sma_15, pastRtn_2, sma_5

2.3 Evaluating the Prediction Quality

Figure 9 exhibits the true positive, true negative, false positive, and false negative. And the accuracy for the test set is 53.58%. The ROC of the best-estimator is 0.54 shown in Figure 10. Moving on to the classification report, the precision indicates the proportion of positive identifications (model predicted class 1) which were actually correct. A model which produces no false positives has a precision of 1.0. Recall indicates the proportion of actual positives which were correctly classified. A model which produces no false negatives has a recall of 1.0. The F1 score is the combination of precision and recall. A perfect model achieves an F1 score of 1.0. Support is the number of samples for each class. From this classification report, the accuracy is 0.54, which is just a little better than random. The values of the macro avg are the same as the weighted avg, meaning that our results are not damaged by the imbalanced class of the sample.

```
# The best parameters:
{'C': 1.0, 'gamma': 'scale', 'kernel': 'linear'}
# The accuracy for training and test sets:
The Accuracy for Training Set is 52.21238938053098
The Accuracy for Test Set is 53.58090185676393
# The classification report:
              precision    recall  f1-score   support
           0       0.59      0.48      0.53       205
           1       0.49      0.60      0.54       172
    accuracy                           0.54       377
   macro avg       0.54      0.54      0.54       377
weighted avg       0.55      0.54      0.54       377
```

Figure 9: Confusion Matrix (3 Features)

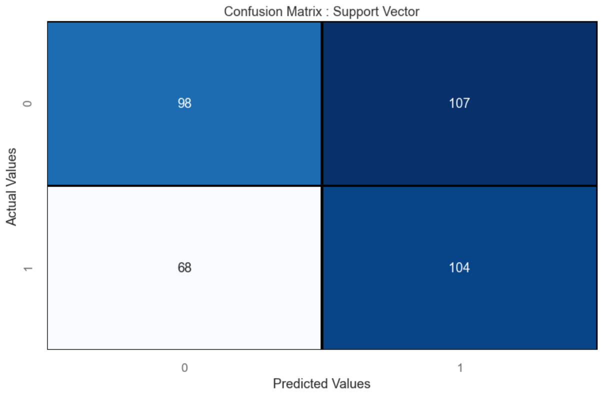
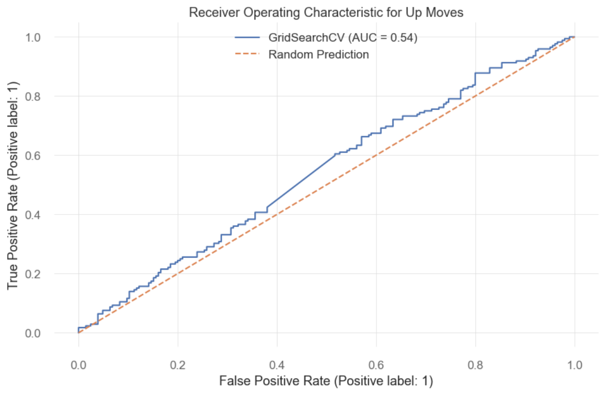


Figure 10: ROC for Up Moves (3 Features)



In this part, we evaluated the second model with 5 features. From Figure 11, the second model performs better in classification. In Figure 12, the precision, both class "0" and "1", improved a lot compared to the first model, 0.63 and 0.57 respectively. The f1 score, the combination between precision and recall, changed to 0.64 and 0.56 from 0.53 and 0.54. We concluded that the model with 5 features is more accurate than that with 3 features in accuracy for test set, true positive, true negative, and ROC. However, SVM estimation with more than 2-3 features becomes very slow. Hence, we need to trade off the accuracy and the speed.

```
# The best parameters:
{'C': 100, 'gamma': 'scale', 'kernel': 'linear'}
# The accuracy for training and test sets:
The Accuracy for Training Set is 59.41845764854614
The Accuracy for Test Set is 60.47745358090185
```

```
# The classification report:
              precision    recall  f1-score   support
           0       0.63      0.66      0.64       205
           1       0.57      0.54      0.56       172
    accuracy                           0.60       377
   macro avg       0.60      0.60      0.60       377
weighted avg       0.60      0.60      0.60       377
```

Figure 11: Confusion Matrix (5 Features)

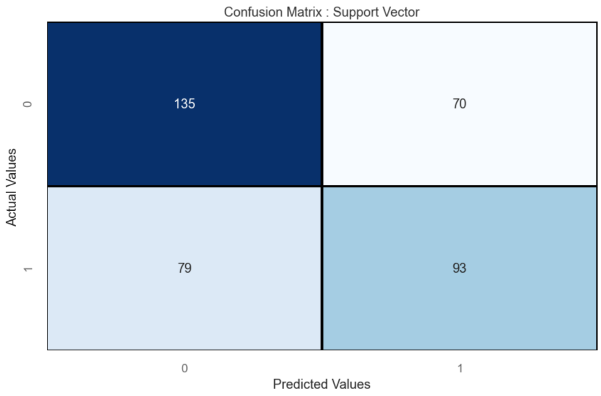
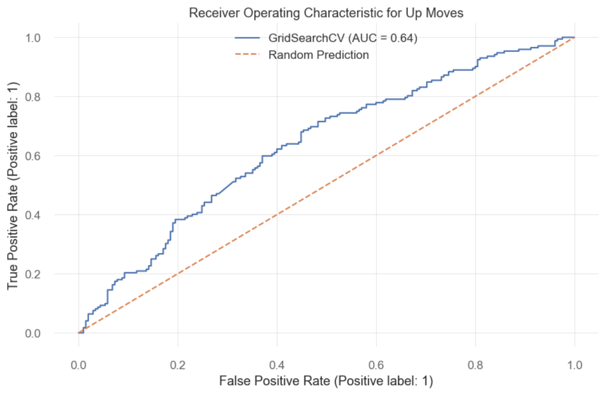


Figure 12: ROC for Up Moves (5 Features)



Market Direction Prediction using SVM

Table of Contents

1 Import Libraries
2 Defining the Functions
2.1 FeaturesComputation
2.2 Processing Data
2.3 XGBoost
2.4 SVM Function
3 Loading the Trading Data
4 Preprocessing the Dataset
4.1 Computing the features
4.2 Labeling
4.3 Processing Data
5 SVM
5.1 SVM Model 1
5.2 Ridge Regression
5.3 LASSO Regression
5.4 Correlation Analysis
5.5 XGBoosting
5.6 Model with 3 Features
5.7 Model with 5 Features
5.8 Trading Strategy
5.9 Trading Strategy

1 Import Libraries

In [2]:

```
# Base Libraries
import numpy as np
import pandas as pd

import yfinance as yf
import cufflinks as cf
cf.set_config_file(offline=True, dimensions=((1000,600)))

import pandas_ta as tap

# Visualization
import seaborn as sns
from pylab import plt
plt.style.use('seaborn')
%matplotlib inline

# Preprocessing
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler, Normalizer
from sklearn.model_selection import train_test_split, TimeSeriesSplit, GridSearchCV

# Preprocessing & Cross validation
from sklearn.model_selection import RandomizedSearchCV, cross_val_score

# SVM
from sklearn import svm
from sklearn.svm import SVR
from sklearn.svm import SVC

# Metrics
from sklearn.metrics import  r2_score, mean_squared_error, mean_absolute_error, accuracy_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix, auc, roc_curve, plot_roc_curve

# XGBoost Classifier
from xgboost import XGBClassifier, plot_importance, to_graphviz

import shap

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

executed in 55ms, finished 12:39:02 2022-11-10

2 Defining the Functions

In [3]:

```python
#==============================================================================
# Feature 1: Open - Close
def openClose(o, c, lag=1):
    return (o.shift(lag)-c.shift(lag)).values

# Feature 2: High - Low
def highLow(h, l, lag=1):
    return (h.shift(lag)-l.shift(lag)).values

#================================================================================
# Feature 3:  Past Returns
def pastReturns(c, lag_list=[1, 2, 5, 10]):
    return [(c.shift(1) - c.shift(lagVal+1)).values for lagVal in lag_list]

#================================================================================
# Feature 4:  Momemtum
def momemtum(c, lag_list=[1, 2, 5, 10, 15]):
    return [(c.shift(1) - c.shift(lagVal+1)).values for lagVal in lag_list]

#================================================================================
# Feature 5: SMA
def SMA(c, lag_list=[5, 15, 30]):
    return [c.rolling(lagVal).mean().values for lagVal in lag_list]

#================================================================================
# Feature 6:  EMA
def EMA(c, lag_list=[5, 10, 20, 50]):
    return [c.ewm(lagVal, adjust=False).mean().values for lagVal in lag_list]
```

executed in 8ms, finished 12:39:05 2022-11-10

2.1 FeaturesComputation

In [4]:

```python
class FeaturesComputation(object):

    def __init__(self, dataset, k=[1, 10, 15], n=[5, 15, 30], m=[1, 2, 5, 10], o=[5, 10, 20, 50]):
        self.dataset = dataset
        self.k = k
        self.n = n
        self.m = m
        self.o = o

    def compute_features(self, base_lag = 1):

        pastRtnDict = \
        dict(
            zip(
                ["pastRtn_" + lag for lag in list(map(str, self.m))],
                pastReturns(self.dataset.Close, self.m)
            )
        )

        momemtumDict = \
        dict(
            zip(
                ["mtm_" + lag for lag in list(map(str, self.k))],
                momemtum(self.dataset.Close, self.k)
            )
        )

        smaDict = \
        dict(
            zip(
                ["sma_" + lag for lag in list(map(str, self.n))],
                SMA(self.dataset.Close, self.n)
            )
        )

        emaDict = \
        dict(
            zip(
                ['ema_' + lag for lag in list(map(str, self.o))],
                EMA(self.dataset.Close, self.o)
            )
        )

        dictFeatures = {
            "openClose": \
            openClose(self.dataset.Open, self.dataset.Close, base_lag),
            "highLow": \
            highLow(self.dataset.High, self.dataset.Low, base_lag),
        }

        return {
            **pastRtnDict,
            **momemtumDict,
            **smaDict,
            **emaDict,
            **dictFeatures
        }
```

executed in 8ms, finished 12:39:08 2022-11-10

2.2 Processing Data

In [5]:
```python
class DataProcess(object):

    def __init__(self, data, X, y, scale_method, testsize, gamma_control):
        self.data = data
        self.X = X
        self.y = y
        self.scale_method = scale_method
        self.testsize = testsize
        self.gamma_control = gamma_control

        self.X_scaled = self.Scaling_Func()
        [self.X_train, self.X_test, self.y_train, self.y_test] = self.train_test_splitFunc()
        [self.X_train_f, self.y_train_f, self.gammaValue] = self.gammaValueFunc()

    def Scaling_Func(self):

        if self.scale_method == 'StandardScaler':
            scale_func = StandardScaler()
            X_scaled = scale_func.fit_transform(self.X)

        if self.scale_method == 'Normalizer':
            scale_func = Normalizer(feature_range=(0, 1))
            X_scaled = scale_func.fit_transform(self.X)

        if self.scale_method == 'MinMaxScaler':
            scale_func = MinMaxScaler()
            X_scaled = scale_func.fit_transform(self.X)

        if self.scale_method == 'RobustScaler':
            scale_func = RobustScaler(norm='max')
            X_scaled = scale_func.fit_transform(self.X)

        X_scaled = pd.DataFrame(X_scaled)
        X_scaled.index = X.index
        col_list = X.columns
        X_scaled.columns = col_list
        return X_scaled

    def train_test_splitFunc(self):
        X_train, X_test, y_train, y_test = train_test_split(
            self.X_scaled, self.y, test_size=self.testsize, random_state=0, shuffle=False)
        return X_train, X_test, y_train, y_test

    def gammaValueFunc(self):
        y_train_mean = np.mean(self.data.loc[self.y_train.index].Returns)
        gammaValue = round(
            np.std(self.data.loc[self.y_train.index].Returns - y_train_mean)/self.gamma_control, 5)

        y_train_f = self.data.loc[self.y_train.index].query(
            "~(-@gammaValue<Returns<@gammaValue)").iloc[:, -1:]
        X_train_f = self.X_train.loc[y_train_f.index]
        return X_train_f, y_train_f, gammaValue

    def print_Func(self):
        print(":> Considering γ := ", round(self.gammaValue*100, 4), "%")
        print(":> New training set observations:",
              self.X_train_f.shape[0], f"({round(self.X_train_f.shape[0]/self.X.shape[0],4)*100}%)")

        print(
            f"::>> Sign '0' obs.: {round(self.y_train_f.query('Sign == 0').shape[0] / self.y_train_f.shape[0], 6)
        print(
            f"::>> Sign '1' obs.: {round(self.y_train_f.query('Sign == 1').shape[0] / self.y_train_f.shape[0], 6)

```

executed in 11ms, finished 12:39:10 2022-11-10

2.3 XGBoost

In [ ]:
```python
def XGBoosting_Filter(X_train_f, y_train_f, X_test, y_test):

    xgbcls =  XGBClassifier(verbosity = 0, silent=True, random_state=101)
    xgbcls.fit(X_train_f, y_train_f)

    tscv = TimeSeriesSplit(n_splits=5, gap=1)

    param_grid = {'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                  'max_depth': [3, 4, 5, 6, 8, 10, 12, 15],
                  'min_child_weight': [1, 3, 5, 7],
                  'gamma': [0.0, 0.1, 0.2 , 0.3, 0.4],
                  'colsample_bytree': [0.3, 0.4, 0.5 , 0.7]}

    svm_rdm = RandomizedSearchCV(xgbcls, param_grid, n_iter=100, scoring='f1', cv=tscv, verbose=0)
    svm_rdm.fit(X_train_f, y_train_f, verbose=0)

    print("----------------------------------------------------------")
    print(svm_rdm.best_params_)
    print("----------------------------------------------------------")
    print(svm_rdm.best_score_)
    print("----------------------------------------------------------")

    cls_best = XGBClassifier(**svm_rdm.best_params_)

    cls_best.fit(X_train_f, y_train_f,
            eval_set=[(X_train_f, y_train_f), (X_test, y_test)],
            eval_metric='logloss',
            verbose=True)

    fig, ax = plt.subplots(figsize=(8, 6))
    feature_imp = pd.DataFrame({'Importance Score': cls_best.feature_importances_,
                               'Features': X.columns}).sort_values(by='Importance Score', ascending=False)

    sns.barplot(x=feature_imp['Importance Score'], y=feature_imp['Features'])
    ax.set_title('Features Importance')
    plt.show()

    plot_importance(cls_best, importance_type='gain', show_values=False)
    plt.show()

    explainer = shap.TreeExplainer(cls_best)
    shap_values = explainer.shap_values(X_test)

    shap.summary_plot(shap_values, X_test, plot_type="bar")

    shap.summary_plot(shap_values, X_test)
```

2.4 SVM Function

In [6]:
```
def SVM_Classifier(X_train_f, y_train_f, X_test, y_test):
    classifierSVM = svm.SVC(probability=True)
    param_grid_svm = {'C': [0.001, 0.01, 0.05, 0.1, 0.5, 1.0, 10.0, 100],
                      'gamma':['scale', 'auto'],
                      'kernel': ['linear','rbf','sigmoid']}
    tscv = TimeSeriesSplit(n_splits=5, gap=1)
    gridSVM = GridSearchCV(estimator=classifierSVM,
                 param_grid = param_grid_svm,
                 scoring='accuracy',
                 cv=tscv,
                 refit=True,
                 n_jobs=-1)

    gridSVM.fit(X_train_f, y_train_f)

    print("----------------------------------------------------------")
    print(gridSVM.best_params_)

    y_pred = gridSVM.predict(X_test)

    train_acc = accuracy_score(y_train_f, gridSVM.predict(X_train_f))
    test_acc = accuracy_score(y_test, y_pred)
    print("----------------------------------------------------------")
    print("The Accuracy for Training Set is {}".format(train_acc*100))
    print("The Accuracy for Test Set is {}".format(test_acc*100))

    print("----------------------------------------------------------")
    print(classification_report(y_test, y_pred))
    print("----------------------------------------------------------")

    cm=confusion_matrix(y_test,y_pred)
    plt.title("Confusion Matrix : Support Vector")
    sns.heatmap(cm, annot=True,fmt='d', cmap='Blues',linewidths=2,linecolor='black',cbar=False)
    plt.ylabel("Actual Values")
    plt.xlabel("Predicted Values")
    plt.savefig('confusion_matrix.png')
    plt.show()

    r_prob = [0 for _ in range(len(y_test))]
    r_fpr, r_tpr, _ = roc_curve(y_test, r_prob, pos_label=1)

    plot_roc_curve(gridSVM, X_test, y_test)
    plt.plot(r_fpr, r_tpr, linestyle='dashed', label='Random Prediction')
    plt.title('Receiver Operating Characteristic for Up Moves')
    plt.legend(loc=9)
    plt.show()

    return gridSVM
```
executed in 8ms, finished 12:39:14 2022-11-10

3 Loading the Trading Data

[https://stockmarketmba.com/stocksintherussell2000.php (https://stockmarketmba.com/stocksintherussell2000.php)](https://stockmarketmba.com/stocksintherussell2000.php)

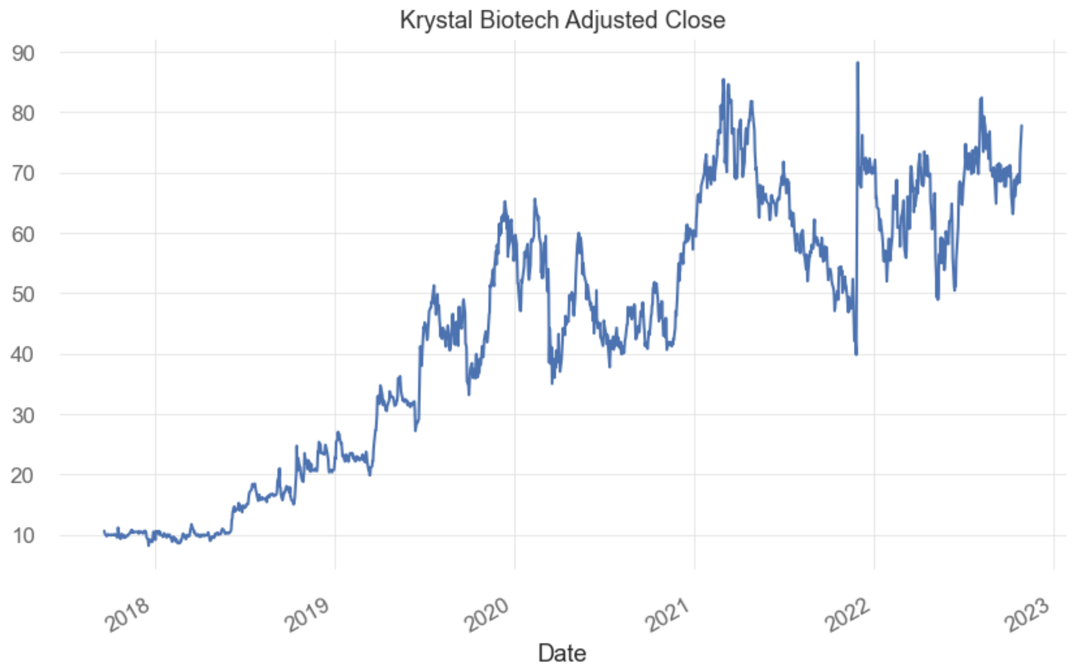
[https://etfdb.com/etf/VBK/#etf-ticker-profile (https://etfdb.com/etf/VBK/#etf-ticker-profile)](https://etfdb.com/etf/VBK/#etf-ticker-profile)

In [454]:
```
stock_data = yf.download('KRYS', start='2016-11-1',end='2022-10-31')
```
executed in 261ms, finished 15:56:36 2022-11-13

```
[*********************100%***********************]  1 of 1 completed
```

In [455]:
```
stock_data['Adj Close'].plot(title='Krystal Biotech Adjusted Close')
```
executed in 232ms, finished 15:56:38 2022-11-13

Out[455]:
```
<AxesSubplot:title={'center':'Krystal Biotech Adjusted Close'}, xlabel='Date'>
```



In [462]:
```
# Setting historical laged returns as fwd returns to compute the labels
# stock_data['Returns']=np.log(stock_data['Adj Close']).diff()
stock_data['Returns']=np.log(stock_data['Adj Close']/stock_data['Adj Close'].shift(1))
stock_data['Sign'] = stock_data['Returns'].apply(lambda x: 0 if x<=0.002 else 1)
stock_data = stock_data.dropna(axis=0)
```
executed in 8ms, finished 15:57:15 2022-11-13

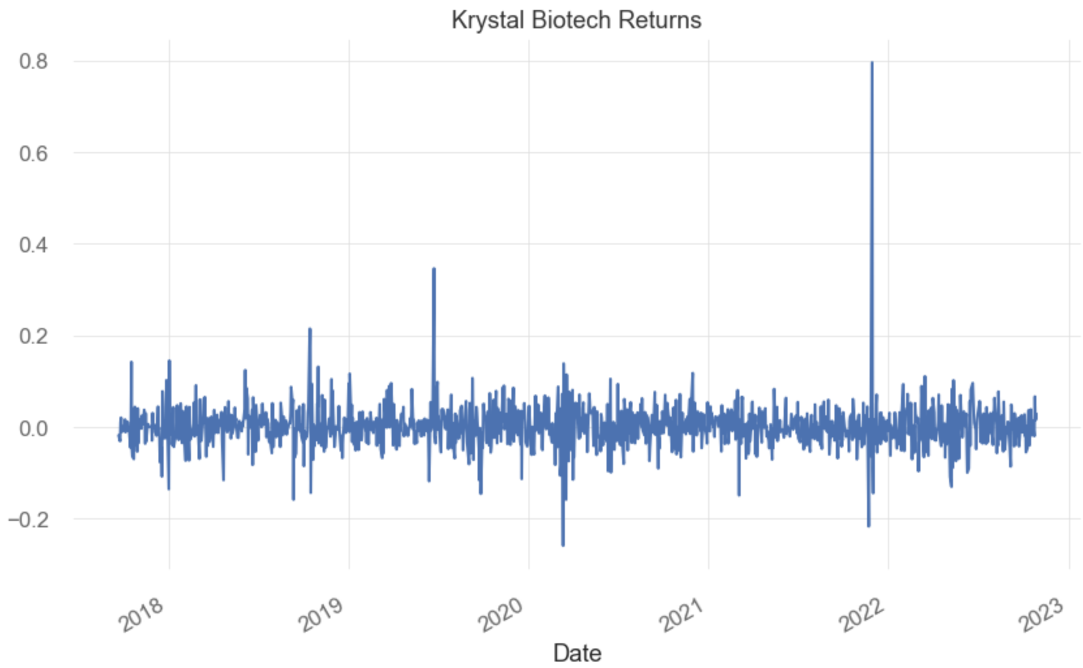
In [464]:

```python
stock_data['Returns'].plot(title='Krystal Biotech Returns')
```

executed in 216ms, finished 15:57:19 2022-11-13

Out[464]:

```
<AxesSubplot:title={'center':'Krystal Biotech Returns'}, xlabel='Date'>
```



4 Preprocessing the Dataset

4.1 Computing the features

In [465]:

```python
featuresDf = pd.DataFrame(
    FeaturesComputation(stock_data, k=[1, 10, 15], n=[5, 15, 30],
                        m=[1, 2, 5, 10], o=[5, 10, 20, 50]).compute_features())

featuresDf.index = stock_data.index
featuresDf = featuresDf.dropna()
```

executed in 12ms, finished 15:57:23 2022-11-13

4.2 Labeling

In [466]:

```python
print(f":::>>> Counting labels for stock_data dataset (Obs.:{stock_data.shape[0]}):")
print("- Sign '0':",round(stock_data.query("Sign == 0").shape[0]/stock_data.shape[0],4)*100,"%")
print("- Sign '1':",round(stock_data.query("Sign == 1").shape[0]/stock_data.shape[0],4)*100,"%")
```

executed in 9ms, finished 15:57:24 2022-11-13

```
:::>>> Counting labels for stock_data dataset (Obs.:1285):
- Sign '0': 52.449999999999996 %
- Sign '1': 47.55 %
```

In [467]:

```python
X = featuresDf
y = stock_data.iloc[:,-1:].loc[X.index]
```

executed in 3ms, finished 15:57:26 2022-11-13

4.3 Processing Data

In [468]:

```
X_scaled = DataProcess(stock_data, X, y, scale_method='StandardScaler',
                       testsize=0.3, gamma_control=10).Scaling_Func()
```

executed in 22ms, finished 15:57:28 2022-11-13

In [469]:

```
X_train, X_test, y_train, y_test = DataProcess(stock_data, X, y,
                                               scale_method='StandardScaler',
                                               testsize=0.3, gamma_control=10
                                              ).train_test_splitFunc()
```

executed in 12ms, finished 15:57:30 2022-11-13

In [470]:

```
X_train_f, y_train_f, gammaValue = DataProcess(stock_data, X, y,
                                               scale_method='StandardScaler',
                                               testsize=0.3, gamma_control=10
                                              ).gammaValueFunc()
```

executed in 14ms, finished 15:57:32 2022-11-13

In [471]:

```
DataProcess(stock_data, X, y,
            scale_method='StandardScaler',
            testsize=0.3, gamma_control=10
            ).print_Func()
```

executed in 15ms, finished 15:57:33 2022-11-13

```
:> Considering γ :=  0.445 %
:> New training set observations: 791 (62.980000000000004%)
::>> Sign '0' obs.: 48.546099999999996%
::>> Sign '1' obs.: 51.4539%
```

In [472]:

```
X_scaled.describe().T
```

executed in 40ms, finished 15:57:35 2022-11-13

Out[472]:

	count	mean	std	min	25%	50%	75%	max
pastRtn_1	1256.0	-1.135857e-17	1.000398	-4.808297	-0.378149	-0.021629	0.375193	19.997223
pastRtn_2	1256.0	2.651807e-18	1.000398	-5.685668	-0.413248	-0.022557	0.398598	13.920458
pastRtn_5	1256.0	-1.555727e-17	1.000398	-4.682211	-0.446962	-0.032692	0.437769	8.411363
pastRtn_10	1256.0	-6.965412e-17	1.000398	-3.843506	-0.482668	-0.059376	0.472715	6.844196
mtm_1	1256.0	-1.135857e-17	1.000398	-4.808297	-0.378149	-0.021629	0.375193	19.997223
mtm_10	1256.0	-6.965412e-17	1.000398	-3.843506	-0.482668	-0.059376	0.472715	6.844196
mtm_15	1256.0	-2.209839e-18	1.000398	-3.437046	-0.490974	-0.054014	0.481602	5.198834
sma_5	1256.0	1.235742e-16	1.000398	-1.680304	-1.013258	0.125444	0.843805	1.797335
sma_15	1256.0	-1.045696e-15	1.000398	-1.659324	-1.009449	0.108320	0.809093	1.669059
sma_30	1256.0	5.692545e-17	1.000398	-1.632777	-0.996155	0.101460	0.817525	1.581345
ema_5	1256.0	2.966488e-16	1.000398	-1.670326	-1.004611	0.107478	0.817182	1.691627
ema_10	1256.0	5.056111e-16	1.000398	-1.657339	-1.006285	0.092206	0.821935	1.621275
ema_20	1256.0	-1.743121e-16	1.000398	-1.634642	-0.993537	0.129376	0.860586	1.562120
ema_50	1256.0	-1.209224e-16	1.000398	-1.576715	-1.017744	0.215187	0.930614	1.374090
openClose	1256.0	4.729055e-18	1.000398	-3.088465	-0.484463	-0.001037	0.475619	6.016741
highLow	1256.0	1.865104e-16	1.000398	-1.539159	-0.700354	-0.117549	0.536201	8.934997

5 SVM

5.1 SVM Model 1

[...]

5.2 Ridge Regression

In [473]:

```python
# Ridge
rid = Pipeline([('scaler', StandardScaler()), ('regressor', Ridge(alpha=1))])

rid.fit(X, y)

print(f'R² Train: {rid.score(X_train, y_train):0.4}')
print(f'R² Test: {rid.score(X_test, y_test):0.4}')
print(rid['regressor'].coef_)

rid_coef_df = pd.DataFrame(rid['regressor'].coef_)

alpha_range = 10**np.linspace(6,-2,100)*0.5
rid_coef = []

for i in alpha_range:
    rid = Pipeline([('scaler', StandardScaler()), ('regressor', Ridge(alpha=i))])
    rid.fit(X, y)
    rid_coef.append(rid['regressor'].coef_)

# ridge.plot_coeff(alpha_range, rid_coef, 'Ridge')
```

executed in 440ms, finished 15:57:40 2022-11-13

```
R² Train: 0.005072
R² Test: 0.001159
[[ 1.20030287e-03  2.91891949e-02 -7.10772083e-02 -8.29076953e-02
   1.20030287e-03 -8.29076953e-02 -5.58531794e-02  1.13453183e+00
  -1.81030823e+00 -4.50021142e-01  1.03719984e+00 -1.86484114e-01
   2.15026195e-01  5.15271188e-02 -1.82442463e-02 -2.25078493e-03]]
```

In [474]:

```python
rid_coef_df.columns = X.T.index
rid_coef_df = rid_coef_df.T
```

executed in 4ms, finished 15:57:43 2022-11-13

In [475]:

```python
rid_coef_df
```

executed in 6ms, finished 15:57:45 2022-11-13

Out[475]:

	0
pastRtn_1	0.001200
pastRtn_2	0.029189
pastRtn_5	-0.071077
pastRtn_10	-0.082908
mtm_1	0.001200
mtm_10	-0.082908
mtm_15	-0.055853
sma_5	1.134532
sma_15	-1.810308
sma_30	-0.450021
ema_5	1.037200
ema_10	-0.186484
ema_20	0.215026
ema_50	0.051527
openClose	-0.018244
highLow	-0.002251

['sma_5', 'sma_15', 'sma_30', 'ema_5']

5.3 LASSO Regression

[...]

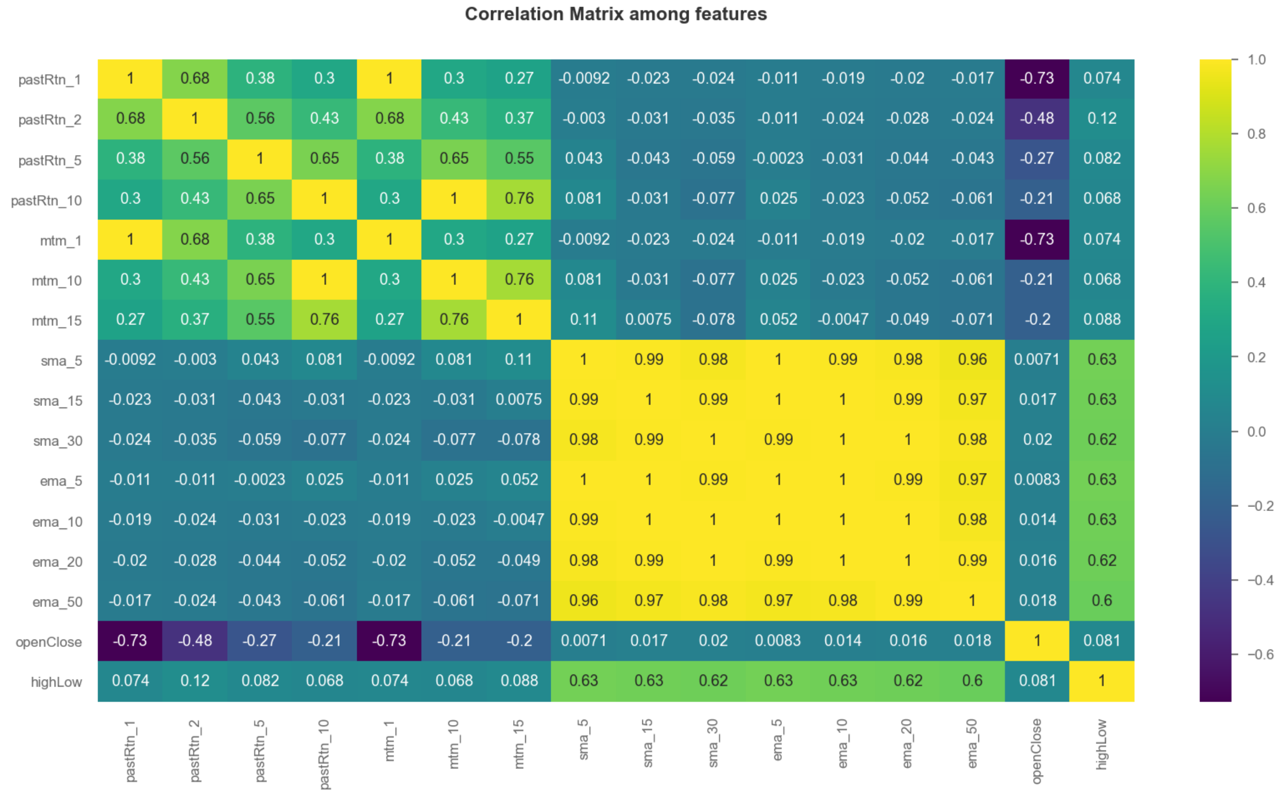
5.4 Correlation Analysis

In [477]:

```python
corrFeatMatrix = X.corr()

plt.figure(figsize=(20,10))
sns.heatmap(corrFeatMatrix, cmap="viridis",annot=True)
plt.title(
    'Correlation Matrix among features',
    fontsize=16, y=1.05, weight='bold'
)
plt.show()
```

executed in 1.38s, finished 15:58:13 2022-11-13



In [478]:

```python
rho_max = 0.80

idxPairwise = np.where(abs(corrFeatMatrix) >= rho_max)
lstPairwise = [
    [corrFeatMatrix.index[x], corrFeatMatrix.columns[y], round(corrFeatMatrix.iloc[x, y], 3)]
    for x, y in zip(*idxPairwise) if x != y and x < y
]
print("::>> Pairwise features with ρ_max= +- 0.80 :")
print("::::>>>> List/list represents [Feat1, Feat2, Corr]:")
lstPairwise
```

executed in 8ms, finished 15:58:15 2022-11-13

```
::>> Pairwise features with ρ_max= +- 0.80 :
::::>>>> List/list represents [Feat1, Feat2, Corr]:
```

Out[478]:

```
[['pastRtn_1', 'mtm_1', 1.0],
 ['pastRtn_10', 'mtm_10', 1.0],
 ['sma_5', 'sma_15', 0.992],
 ['sma_5', 'sma_30', 0.978],
 ['sma_5', 'ema_5', 0.998],
 ['sma_5', 'ema_10', 0.992],
 ['sma_5', 'ema_20', 0.981],
 ['sma_5', 'ema_50', 0.959],
 ['sma_15', 'sma_30', 0.992],
 ['sma_15', 'ema_5', 0.998],
 ['sma_15', 'ema_10', 0.999],
 ['sma_15', 'ema_20', 0.992],
 ['sma_15', 'ema_50', 0.972],
 ['sma_30', 'ema_5', 0.988],
 ['sma_30', 'ema_10', 0.996],
 ['sma_30', 'ema_20', 0.998],
 ['sma_30', 'ema_50', 0.984],
 ['ema_5', 'ema_10', 0.998],
 ['ema_5', 'ema_20', 0.99],
 ['ema_5', 'ema_50', 0.969],
 ['ema_10', 'ema_20', 0.997],
 ['ema_10', 'ema_50', 0.98],
 ['ema_20', 'ema_50', 0.991]]
```

- `['pastRtn_1','mtm_1', 'openClose']`
- `['pastRtn_10', 'mtm_10', 'mtm_15']`
- `['sma_5', 'sma_15', 'sma_30', 'ema_5', 'ema_10', 'ema_20', 'ema_50']`

5.5 XGBoosting

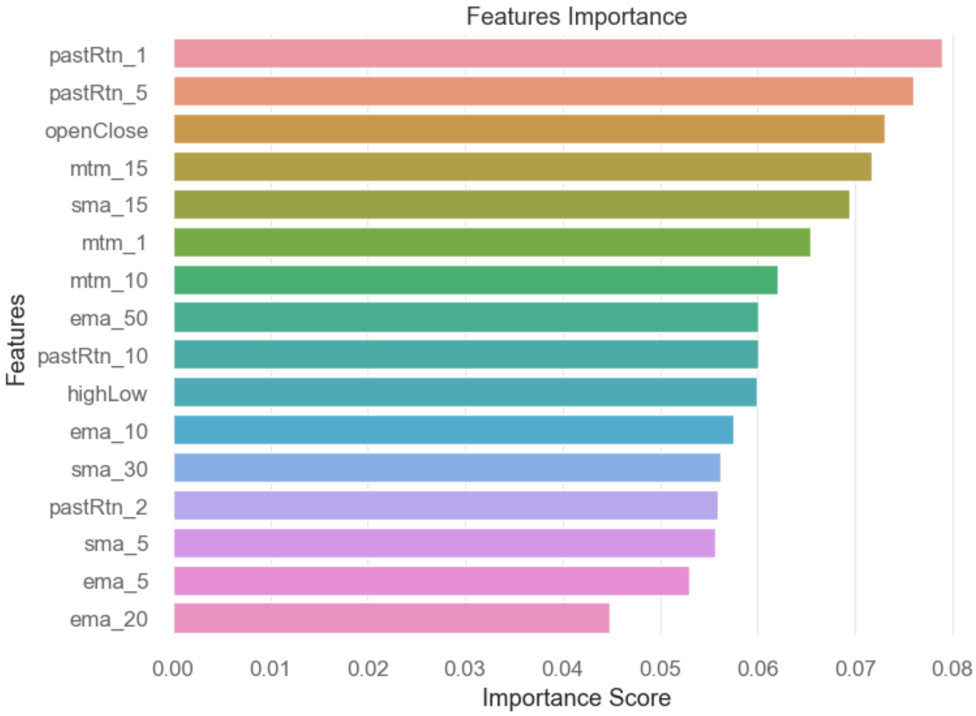
In [479]:

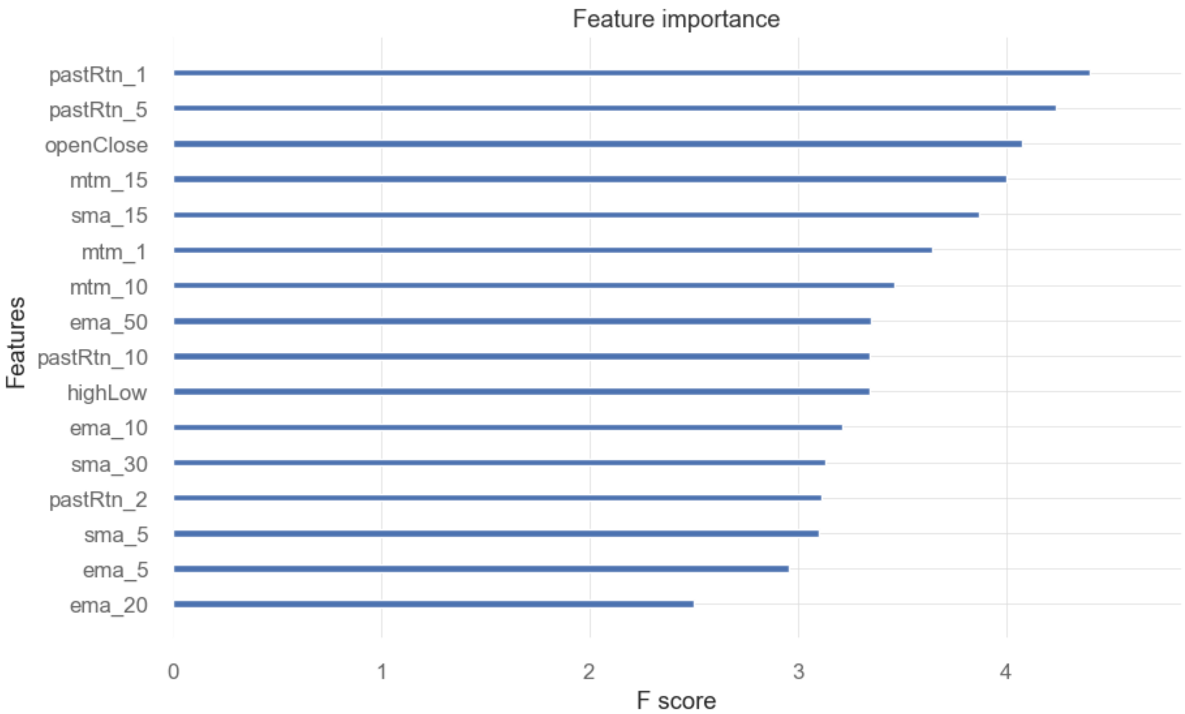
```
1 XGBoosting_Filter(X_train_f, y_train_f, X_test, y_test)
```

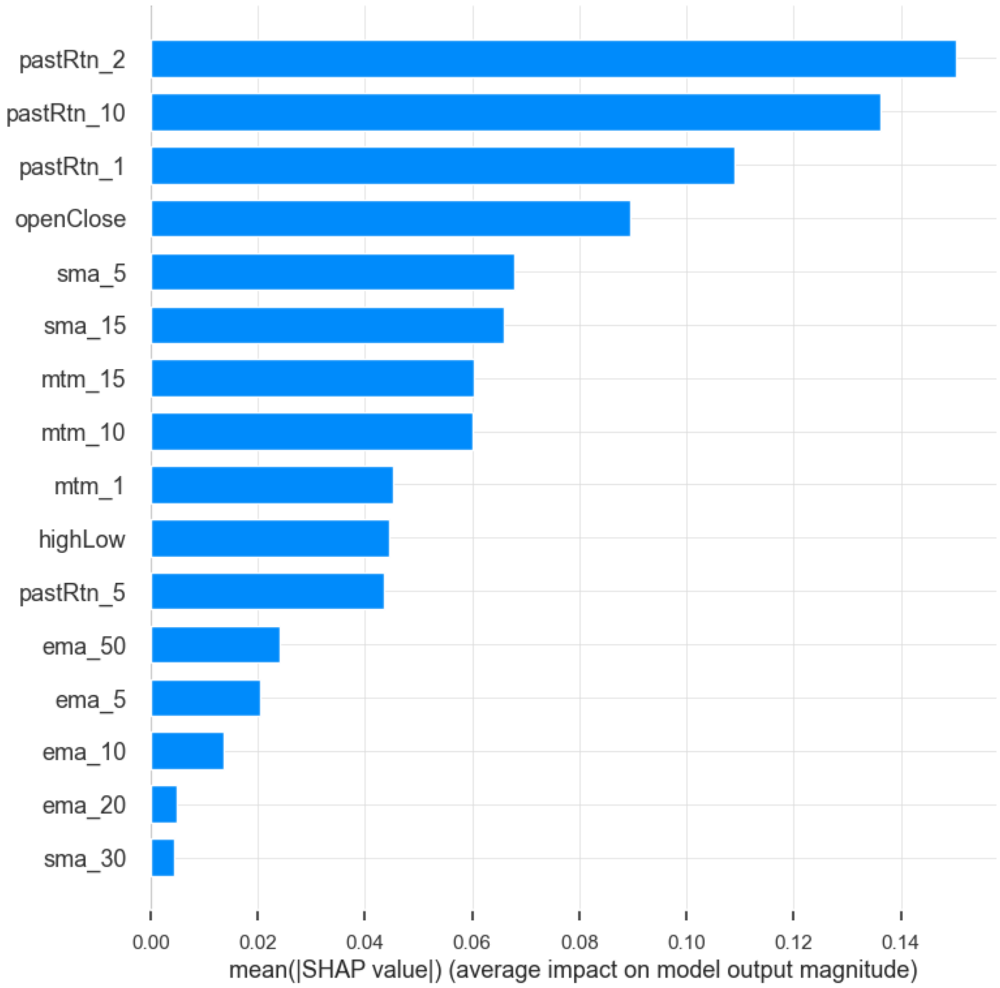
executed in 27.9s, finished 15:58:46 2022-11-13

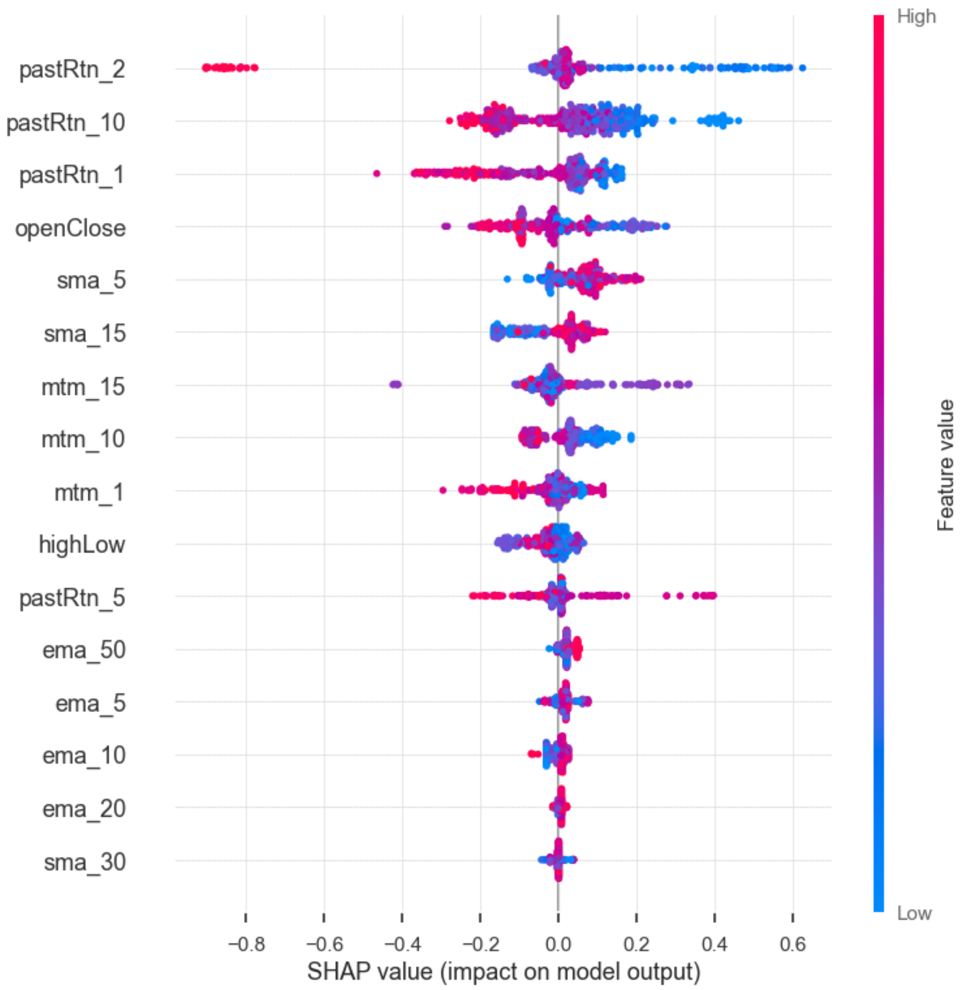
```
------------------------------------------------------------
{'min_child_weight': 3, 'max_depth': 3, 'learning_rate': 0.05, 'gamma': 0.2, 'colsample_bytree': 0.5}
------------------------------------------------------------
0.5543562337937731
------------------------------------------------------------
[0]	validation_0-logloss:0.69049	validation_1-logloss:0.69425
[1]	validation_0-logloss:0.68787	validation_1-logloss:0.69540
[2]	validation_0-logloss:0.68528	validation_1-logloss:0.69587
[3]	validation_0-logloss:0.68248	validation_1-logloss:0.69518
[4]	validation_0-logloss:0.68069	validation_1-logloss:0.69524
[5]	validation_0-logloss:0.67839	validation_1-logloss:0.69466
[6]	validation_0-logloss:0.67657	validation_1-logloss:0.69456
[7]	validation_0-logloss:0.67509	validation_1-logloss:0.69452
[8]	validation_0-logloss:0.67346	validation_1-logloss:0.69499
[9]	validation_0-logloss:0.67129	validation_1-logloss:0.69542
[10]	validation_0-logloss:0.67004	validation_1-logloss:0.69559
[11]	validation_0-logloss:0.66806	validation_1-logloss:0.69508
[12]	validation_0-logloss:0.66670	validation_1-logloss:0.69511
[13]	validation_0-logloss:0.66478	validation_1-logloss:0.69487
[14]	validation_0-logloss:0.66338	validation_1-logloss:0.69558
[15]	validation_0-logloss:0.66183	validation_1-logloss:0.69667
[16]	validation_0-logloss:0.66014	validation_1-logloss:0.69632
[17]	validation_0-logloss:0.65895	validation_1-logloss:0.69659
[18]	validation_0-logloss:0.65719	validation_1-logloss:0.69703
[19]	validation_0-logloss:0.65621	validation_1-logloss:0.69732
[20]	validation_0-logloss:0.65499	validation_1-logloss:0.69800
[21]	validation_0-logloss:0.65407	validation_1-logloss:0.69794
[22]	validation_0-logloss:0.65287	validation_1-logloss:0.69793
[23]	validation_0-logloss:0.65081	validation_1-logloss:0.69759
[24]	validation_0-logloss:0.64860	validation_1-logloss:0.69753
[25]	validation_0-logloss:0.64767	validation_1-logloss:0.69804
[26]	validation_0-logloss:0.64552	validation_1-logloss:0.69903
[27]	validation_0-logloss:0.64436	validation_1-logloss:0.69927
[28]	validation_0-logloss:0.64300	validation_1-logloss:0.69965
[29]	validation_0-logloss:0.64123	validation_1-logloss:0.70012
[30]	validation_0-logloss:0.63967	validation_1-logloss:0.70005
[31]	validation_0-logloss:0.63842	validation_1-logloss:0.70043
[32]	validation_0-logloss:0.63742	validation_1-logloss:0.70064
[33]	validation_0-logloss:0.63572	validation_1-logloss:0.70060
[34]	validation_0-logloss:0.63513	validation_1-logloss:0.70133
[35]	validation_0-logloss:0.63342	validation_1-logloss:0.70131
[36]	validation_0-logloss:0.63255	validation_1-logloss:0.70163
[37]	validation_0-logloss:0.63117	validation_1-logloss:0.70203
[38]	validation_0-logloss:0.62999	validation_1-logloss:0.70230
[39]	validation_0-logloss:0.62847	validation_1-logloss:0.70256
[40]	validation_0-logloss:0.62728	validation_1-logloss:0.70261
[41]	validation_0-logloss:0.62548	validation_1-logloss:0.70254
[42]	validation_0-logloss:0.62476	validation_1-logloss:0.70511
[43]	validation_0-logloss:0.62330	validation_1-logloss:0.70507
[44]	validation_0-logloss:0.62213	validation_1-logloss:0.70548
[45]	validation_0-logloss:0.62147	validation_1-logloss:0.70809
[46]	validation_0-logloss:0.62019	validation_1-logloss:0.70819
[47]	validation_0-logloss:0.61902	validation_1-logloss:0.70831
[48]	validation_0-logloss:0.61820	validation_1-logloss:0.70856
[49]	validation_0-logloss:0.61688	validation_1-logloss:0.70857
[50]	validation_0-logloss:0.61552	validation_1-logloss:0.70853
[51]	validation_0-logloss:0.61497	validation_1-logloss:0.70860
[52]	validation_0-logloss:0.61345	validation_1-logloss:0.70842
[53]	validation_0-logloss:0.61223	validation_1-logloss:0.70944
[54]	validation_0-logloss:0.61185	validation_1-logloss:0.70963
[55]	validation_0-logloss:0.61109	validation_1-logloss:0.71033
[56]	validation_0-logloss:0.60926	validation_1-logloss:0.71009
[57]	validation_0-logloss:0.60830	validation_1-logloss:0.71025
[58]	validation_0-logloss:0.60714	validation_1-logloss:0.71008
[59]	validation_0-logloss:0.60646	validation_1-logloss:0.71013
[60]	validation_0-logloss:0.60544	validation_1-logloss:0.71048
[61]	validation_0-logloss:0.60486	validation_1-logloss:0.71213
[62]	validation_0-logloss:0.60339	validation_1-logloss:0.71225
[63]	validation_0-logloss:0.60235	validation_1-logloss:0.71256
[64]	validation_0-logloss:0.60070	validation_1-logloss:0.71225
[65]	validation_0-logloss:0.59926	validation_1-logloss:0.71213
[66]	validation_0-logloss:0.59768	validation_1-logloss:0.71277
[67]	validation_0-logloss:0.59604	validation_1-logloss:0.71289
[68]	validation_0-logloss:0.59556	validation_1-logloss:0.71295
[69]	validation_0-logloss:0.59464	validation_1-logloss:0.71347
[70]	validation_0-logloss:0.59342	validation_1-logloss:0.71390
[71]	validation_0-logloss:0.59224	validation_1-logloss:0.71665
[72]	validation_0-logloss:0.59149	validation_1-logloss:0.71747
[73]	validation_0-logloss:0.59108	validation_1-logloss:0.71776
[74]	validation_0-logloss:0.59014	validation_1-logloss:0.71805
[75]	validation_0-logloss:0.58985	validation_1-logloss:0.71814
[76]	validation_0-logloss:0.58869	validation_1-logloss:0.71809
[77]	validation_0-logloss:0.58751	validation_1-logloss:0.72098
[78]	validation_0-logloss:0.58680	validation_1-logloss:0.72113
[79]	validation_0-logloss:0.58627	validation_1-logloss:0.72098
[80]	validation_0-logloss:0.58532	validation_1-logloss:0.72055
[81]	validation_0-logloss:0.58486	validation_1-logloss:0.72065
[82]	validation_0-logloss:0.58367	validation_1-logloss:0.72089
[83]	validation_0-logloss:0.58278	validation_1-logloss:0.72161
[84]	validation_0-logloss:0.58211	validation_1-logloss:0.72159
[85]	validation_0-logloss:0.58180	validation_1-logloss:0.72179
```

```
[86]	validation_0-logloss:0.58030	validation_1-logloss:0.72169
[87]	validation_0-logloss:0.57924	validation_1-logloss:0.72218
[88]	validation_0-logloss:0.57833	validation_1-logloss:0.72360
[89]	validation_0-logloss:0.57742	validation_1-logloss:0.72363
[90]	validation_0-logloss:0.57707	validation_1-logloss:0.72348
[91]	validation_0-logloss:0.57599	validation_1-logloss:0.72538
[92]	validation_0-logloss:0.57471	validation_1-logloss:0.72556
[93]	validation_0-logloss:0.57379	validation_1-logloss:0.72543
[94]	validation_0-logloss:0.57289	validation_1-logloss:0.72537
[95]	validation_0-logloss:0.57169	validation_1-logloss:0.72572
[96]	validation_0-logloss:0.57123	validation_1-logloss:0.72636
[97]	validation_0-logloss:0.57005	validation_1-logloss:0.72689
[98]	validation_0-logloss:0.56939	validation_1-logloss:0.72755
[99]	validation_0-logloss:0.56906	validation_1-logloss:0.72757
```









- `['pastRtn_1','mtm_1']`

- ['pastRtn_10', 'mtm_10']
- ['sma_5', 'sma_15', 'sma_30', 'ema_5', 'ema_10', 'ema_20', 'ema_50']

- ['sma_5', 'sma_15', 'ema_5', 'pastRtn_2'] # ROC=0.65
- ['sma_15', 'ema_5', 'pastRtn_2']   # ROC=0.58

- ['mtm_15', 'sma_5', 'highLow']

- ['sma_5', 'pastRtn_5', 'ema_5']

5.6 Model with 3 Features

In [553]:

```
newFeatures = ['pastRtn_1', 'sma_15', 'pastRtn_10'] # 0.54

newX_train_f = X_train_f[newFeatures]
newX_test = X_test[newFeatures]
newX_scaled = X_scaled[newFeatures]

newgridSVM = SVM_Classifier(newX_train_f, y_train_f, newX_test, y_test)
```

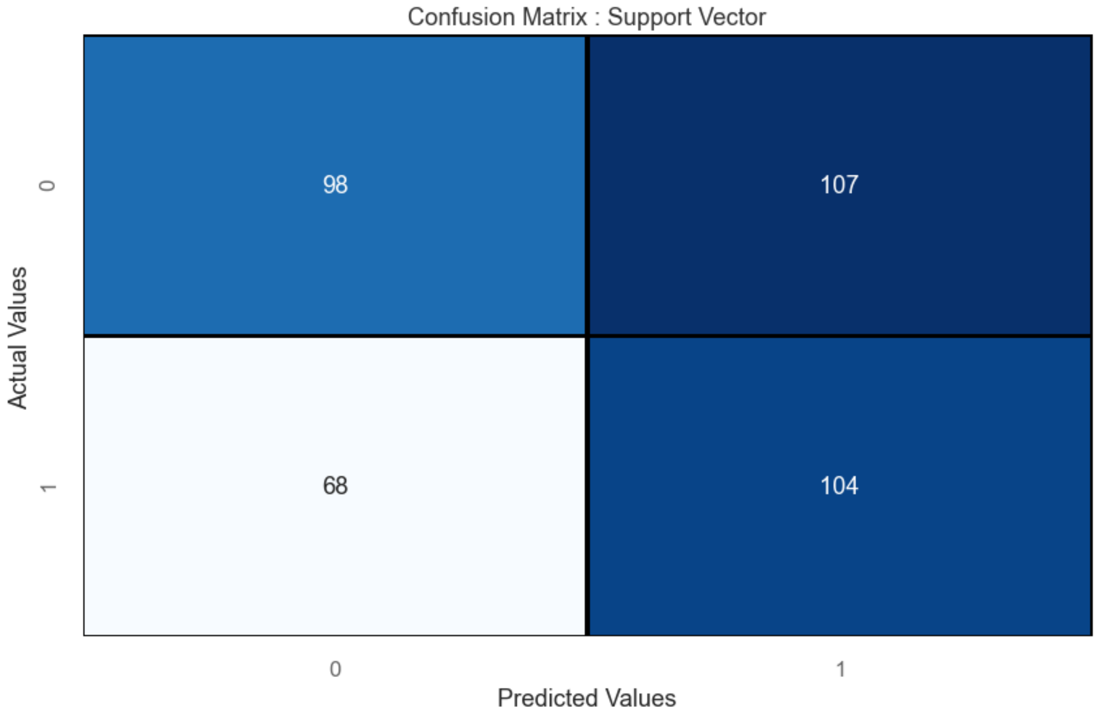
executed in 3.42s, finished 16:22:37 2022-11-14

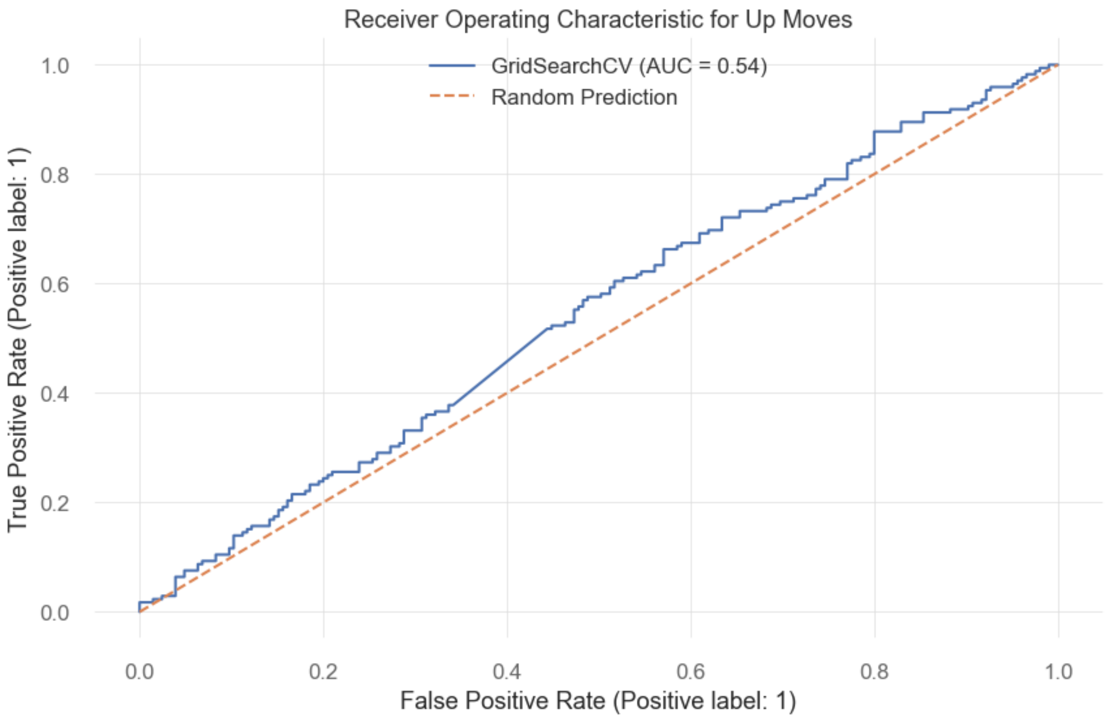
```
---------------------------------------------------------
{'C': 1.0, 'gamma': 'scale', 'kernel': 'linear'}
---------------------------------------------------------
The Accuracy for Training Set is 52.21238938053098
The Accuracy for Test Set is 53.58090185676393
---------------------------------------------------------
              precision    recall  f1-score   support

           0       0.59      0.48      0.53       205
           1       0.49      0.60      0.54       172

    accuracy                           0.54       377
   macro avg       0.54      0.54      0.54       377
weighted avg       0.55      0.54      0.54       377

---------------------------------------------------------
```





5.7 Model with 5 Features

In [549]:

```
newFeatures = ['pastRtn_1', 'pastRtn_2', 'pastRtn_10', 'sma_5', 'sma_15'] # 0.64

newX_train_f = X_train_f[newFeatures]
newX_test = X_test[newFeatures]
newX_scaled = X_scaled[newFeatures]

newgridSVM = SVM_Classifier(newX_train_f, y_train_f, newX_test, y_test)
```

executed in 5.94s, finished 16:18:23 2022-11-14

```
---------------------------------------------------------
{'C': 100, 'gamma': 'scale', 'kernel': 'linear'}
---------------------------------------------------------
The Accuracy for Training Set is 59.41845764854614
The Accuracy for Test Set is 60.47745358090185
---------------------------------------------------------
              precision    recall  f1-score   support

           0       0.63      0.66      0.64       205
           1       0.57      0.54      0.56       172

    accuracy                           0.60       377
   macro avg       0.60      0.60      0.60       377
weighted avg       0.60      0.60      0.60       377

---------------------------------------------------------
```