Market Prediction with Tree-based Method

HAOTIAN ZHANG

December 24, 2020

1 Background and Motivation

1.1 Background

Machine learning is used to figure out the laws and patterns that exist in the dataset. After learning those laws and patterns, it gains the capacity to predict(for continuous labels) or classify(for discrete labels) when given a new dataset and such capacity is generated by what it has already learned from the old dataset. Primary goal of this paper is predicting either the movement direction(upward movement or downward movement) of asset returns. Moreover, the application of machine learning approaches in portfolio management will also be mentioned briefly.

The effectiveness of model rely on the variables and models. Consequently, it is necessary for me to clarify the basic principles in the data section which determines the features in my model. This paper we will use simplified variables and focus on tree-based models and tuning parameters.

Besides, Tree-based models are a popular choice for classification problem. One reason is that trees require less feature engineering and they generally have higher accuracy when it comes to predictions compared to regression models. Another advantage of using tree-based models is that they are good at picking up relationships. Linear models view each variable as independent and do not recognize relationships like tree-based models.

Two of the most popular tree-based models are Random Forest and Gradient Boosted Tree (GBDT is a generalization of boosting to arbitrary differentiable loss functions). Both are ensemble models which typically provide greater accuracy and broader coverage than single decision trees. This paper focuses on boosting model and use the decision tree classifier and random forests as comparison.

1.2 Motivation

Why this topic is important? Quantitative stock selection and prediction is a popular academic research area and in the industry. Fama and French (1993), Lakonishok (1994), and Song (1994) established a linear model of stock excess returns, and proposed that the excess returns can be well explained by current stock prices, book value of equity, and earnings per share. Compared with the classic linear multi-factor models, the machine learning model pays more attention to the prediction ability of the model. It can capture more detailed market signals. This paper will focus on the first phase of the process, to compose multi-factors into a single factor by machine learning techniques to predict the future stock return. This method can be proposed by Li and Zhang (2018) to use it in further dynamic weighting strategy rather than equal weighting strategy and IC weighting strategy that's traditionally used in the industry. Besides, Pavan (2018) proposed the prediction can be extensively used into the Black-Litterman model.

2 Data

2.1 Dataset in Paper

The study is based on a US stock index, SP 500. This index measures the stock performance of 500 large companies listed on stock exchanges in the United States. It is one of the most commonly followed equity indices. The dataset for this study is public and download from Yahoo Financial, the time series are divided sequentially into three subgroups, a training set(2012-2019), a validation set(2020.01-2020.06), a test set(2020.07-2020.11).

2.2 Generating Data Features

Moving average price is selected as a main part of variables. However, market environment must also be considered, where moving volatility of stock prices is treated as a proxy of market environment. It is true that some of other indicators such as GDP, LIBOR, momentum indicators are also qualified for the role of the proxy. Nevertheless, this text merely take moving volatility into account. Besides, we take volume into account because looking at volume patterns over time can help get a sense of the strength or conviction behind advances and declines in specific stocks and entire markets.

After I acquire the moving average price, the moving volatility and volume, features are ready to be created. I include both the moving average price and the moving volatility into the feature set because comparison in empirical analysis tells me that the combination of moving average price and moving volatility performs a little bit better than adjusted moving average price.

Variables contained in the feature set for tree-based classifiers are shown in Table 1.

Index	Variable	Range
1	Close Price for Today	[1277, 3702]
2	Moving Average Close Price with Time Window=2 days	[2262, 3697]
3	Moving Average Close Price with Time Window=3 days	[2262, 3697]
4	Moving Average Close Price with Time Window=4 days	[2262, 3697]
5	Moving Average Close Price with Time Window=5 days	[2261, 3697]
6	Moving Average Close Price with Time Window=6 days	[2261, 3691]
7	Moving Average Close Price with Time Window=7 days	[2260, 3683]
8	Square Root of Naive Sample Variance of Price (days=2)	[174, 26388]
9	Square Root of Naive Sample Variance of Price with Time Window=2 days	[174, 19828]
10	Square Root of Naive Sample Variance of Price with Time Window=3 days	[174, 18884]
11	Square Root of Naive Sample Variance of Price with Time Window=4 days	[174, 15442]
12	Square Root of Naive Sample Variance of Price with Time Window=5 days	[174, 13345]
13	Square Root of Naive Sample Variance of Price with Time Window=6 days	[174, 13160]
14	Volume	[1.24b, 9.04b]

Table 1: Variable Names

The features are not independent, like the close prices are correlated with moving average ones. Thus, we choose the tree-based method to detect the nonlinear pattern. Besides, Pavan (2018) shows the tree-based method has higher prediction power in market compared to logistic regression, SVM, Naive Bayes in term of AUC.

3 Model

3.1 the Structure of Prediction in the Project

This section focus on the prediction of movement directions of asset return. This is a binary variables problem and thus only classifiers are needed.

For this paper, we focus more on tree-based models, specifically, decision tree, random forest and boosting tree and compare the effectiveness of the tree-based method based on test set confusion matrix. Assume every variable in the tree is forced to interact with every variable further up the tree.

For each method, the data is divided into train(2012-2019), validation(2020.01-2020.06) and test(2020.07-2020.11). We use validation set to tune the parameters and select the model with smallest validation set. Then we train the best estimated model and use it on the test sets. The process is shown in Figure 1. I don't use K-Fold validation because my features of different day are not independent and K-Fold may lead to overfitting issues.

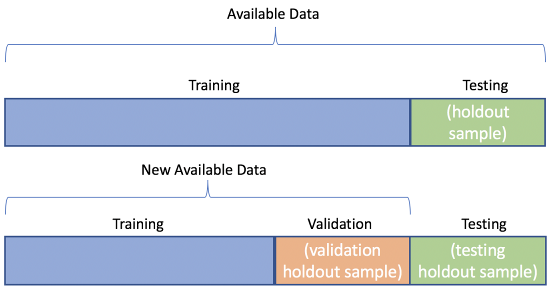


Figure 1: Model Estimation Framework

The main model Boosting Tree is based on weak learners (high bias, low variance). In terms of decision trees, weak learners are shallow trees, sometimes even as small as decision stumps (trees with two leaves). Boosting reduces error mainly by reducing bias. For comparison, Random Forest uses as fully grown decision trees (low bias, high variance). It tackles the error reduction task in the opposite way: by reducing variance. The trees are made uncorrelated to maximize the decrease in variance, but the algorithm cannot reduce bias (which is slightly higher than the bias of an individual tree in the forest). Hence the need for large, unpruned trees, so that the bias is initially as low as possible.

3.2 Extension: The Application of Return Prediction in Portfolio Management

When practitioners want to construct a portfolio, there are two important questions: what kind of assets and what are the weights. By Black-Litterman model, I composes the prediction of each stock into the calculation of weights.
The Black-Litterman (BL) model uses a Bayesian approach to combine the subjective views of an investor regarding the expected returns of one or more assets with the market equilibrium vector of expected returns to form a new, mixed estimate of expected returns as below.

$$E[R] = [(\tau\Sigma)^{-1} + P^T\Omega^{-1}P]^{-1}[\tau\Sigma^{-1}\Pi + P^T\Omega^{-1}Q]$$

where τ is a small constant, Σ is the covariance matrix, and Π is the market equilibrium. Moreover, P, Q and Ω could be determined by investors views. BL model considers two sources of information about future excess returns from market equilibrium and investors' views. Actually we can use machine learning to predict on asset returns and treat the results as our views.

4 Results and Interpretation

4.1 Tree-Based Model Prediction

Tuning Parameters In order to keep away from overfitting in tree, several parameters should be carefully identified. Table 2 shows the parameters, the names of the parameters are consistent with the names in sklearn.tree.GradientBoostingClassifier().

Name of Parameter	n_estimators (#trees)	max_depth
Meaning	The number of boosting stages	The maximum depth of individual estimator

Table 2: Parameters to tune

At first, it is difficult for us to change the two parameters at the same time and see the impacts of changes. Thus, I first hold all but one variable constant and tune only one parameter and then use cross validation. Repeat this process and only add one parameter at one time to simplify the problem. But we should notice that those parameters that have already been considered before should be set to their best values to maximize the accuracy score. Besides, based on empirical results, tuning many parameters may cause overfitting in the validation set and thus decrease prediction power on test set. Thus, I choose two of the most important parameters. Finally, I get the combination of best values of the two parameters: n_estimators and max_depth. Figure 2-(left), Figure 2-(right) shows the relationship between validation accuracy score and n_estimate, max_depth respectively.

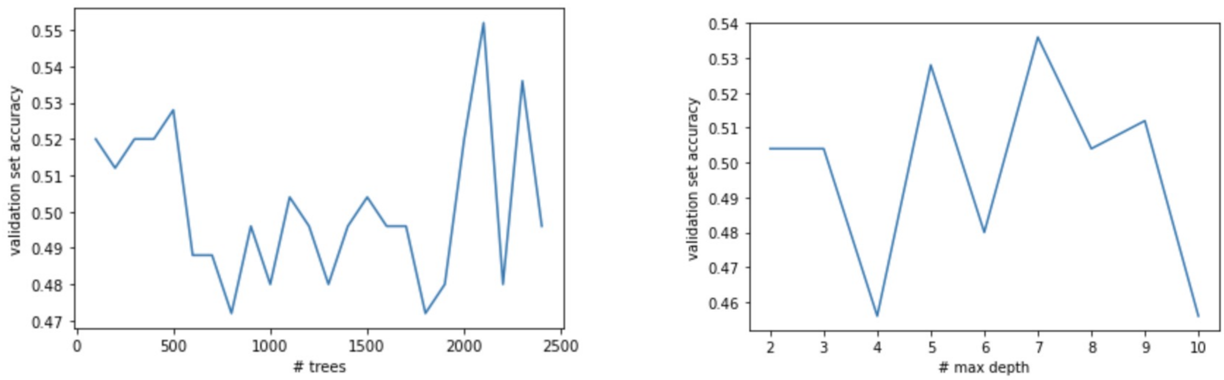


Figure 2: validation accuracy score: consider n_estimate(left) consider MD given n_estimate(right)

n_estimators	max_depth
2100	7

Table 3: Tuned Parameters for GBDT

Table 3 shows the optimal values of n_estimate, max_depth.

Model Comparison Based on the validation accuracy in table 4, decision appears to achieve the highest result. But one key observation is that the decision tree has very low prediction power because it classify most of samples as 1 in the validation set. Boosted Tree Classifier and Random Forest Classifier both achieve reasonable results with the same validation accuracy. But there is little difference that boosting tree tend to classify more samples as 1. Based on validation set, we can either pick Random Forest or GBDT. GBDT has more TN but fewer TP than Random Forest. Since the features have poor prediction power, I don't use AUC but suggest to further compare AUC, recall rate.

Algorithm	TP	FP	FN	TN	valid accuracy
Decision Tree	68	54	1	2	56.0%
Random Forests	54	46	15	10	51.2%
Gradient Boosted Tree	43	33	26	23	51.2%

Table 4: validation set performance of all the tree-based classifiers

Result & **Interpretation** Gradient boosted classifier presents preferences for feature set computed by volume and drift-independent volatility based on Figure 3. Moreover, we should notice that the decision tree has very low prediction power because it classifies most of samples as 1. Boosted Tree Classifier and Random Forest Classifier both achieve reasonable results but both suffer from high FNs in Table 5. Since the features have poor prediction power, I would suggest to use. On the other hand, test results for newly generated test set also imply such preferences. In conclusion, I regard GBDT as a good classifier because of reasonable prediction results, feature selection based on feature importance. After tuning parameters, the prediction is still consistent than in short time prediction, thus can decrease overfitting. But the limitation is also related to the prediction, GBDT has lower interpretability.

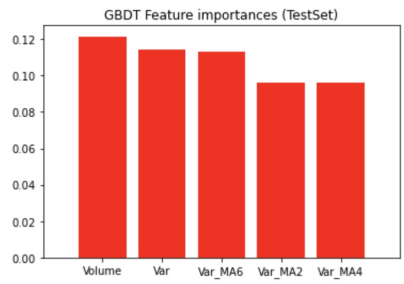


Figure 3: Feature Importance

Algorithm	TP	FP	FN	TN	valid Accuracy
Decision Tree	63	48	4	0	54.8%
Random Forests	21	11	46	37	50.4%
Gradient Boosted Tree	24	12	43	36	52.2%

Table 5: test set performance of all the tree-based classifiers

5 Conclusion

Gradient Boosted Tree is a good classifier within tree methods because of reasonable out of sample prediction results, feature selection based on feature importance. After tuning parameters, the prediction is still consistent than Random Forest, Decision Tree in short time prediction, thus can decrease overfitting. The accuracy is overall low because the stock market data has low signal to noise ratio and it's time changing data generating process so overall overfitting is a main issue. But the problem can be addressed by further data mining highly related predicting features. A limitation of this study is that it considered only tree-based ensemble models. Hence, in our future work, we will incorporate machine learning models that involve the Gaussian process, a regularization technique, and kernel-based techniques. Also, a good extension of this is to predict on absolute returns and further study on BL models for portfolio construction.

Appendix

A. the visualization of tuned decision tree

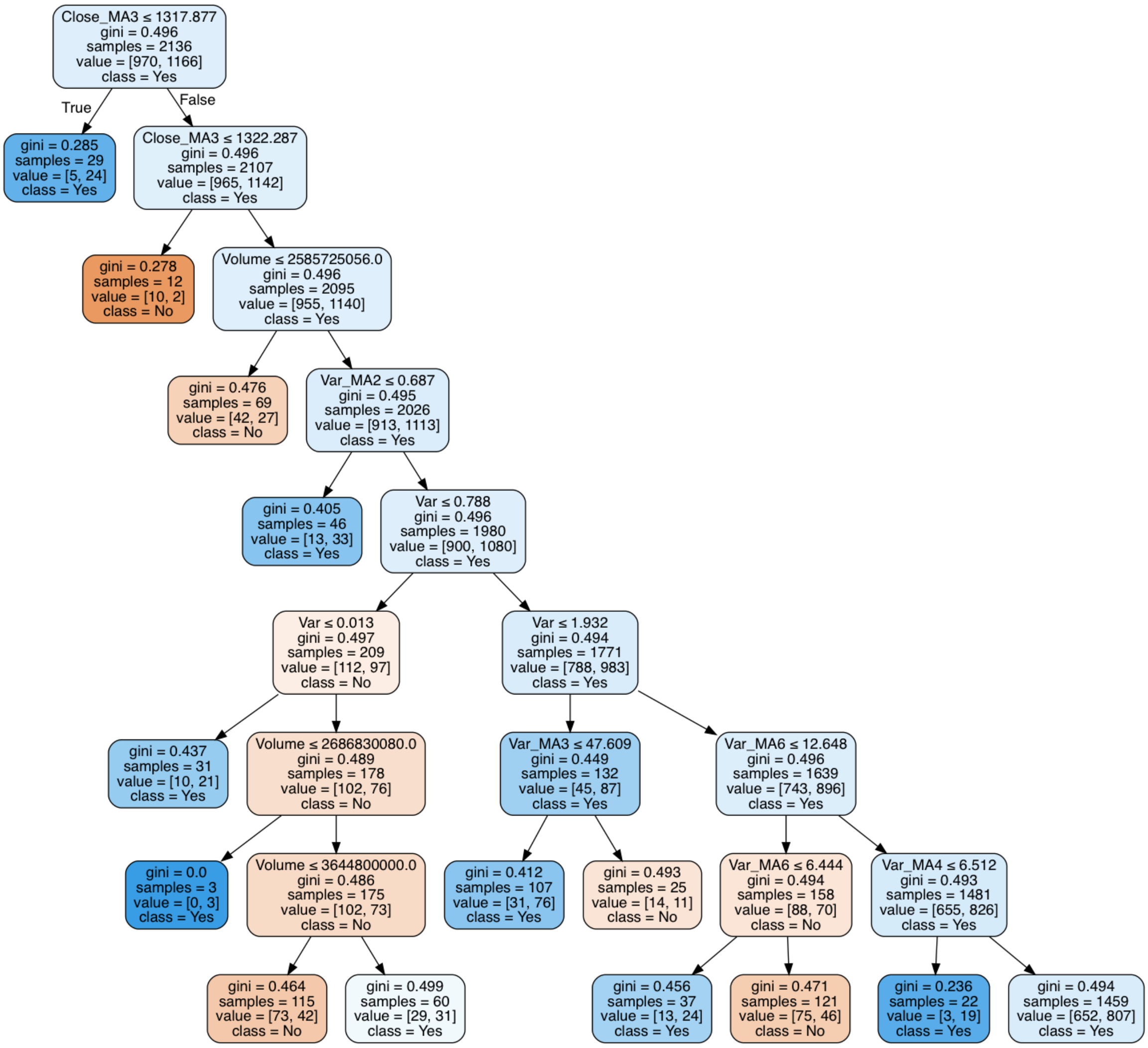


Figure 4: Trained Decision Tree

B. python code in the Jupyter Notebook

Appendix Market Prediction with Tree-based Method (Python Code)

December 23, 2020

data cleaning

[124]:
```python
import os # accessing directory structure
import sys
import warnings
import time

# plotting
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np # linear algebra
import pandas as pd


# metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.metrics import accuracy_score, r2_score, plot_confusion_matrix

#Trees
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

warnings.simplefilter(action='ignore', category=FutureWarning)
```

[51]:
```python
nRowsRead = None # specify 'None' if want to read whole file
# SP500_test.csv may have more rows in reality, but we are only loading/
 ↪previewing the first 1000 rows
raw_data = pd.read_csv('^GSPC.csv', delimiter=',', nrows = nRowsRead)
raw_data.drop(["Adj Close"],axis=1,inplace=True)
```

```
raw_data["Date"] = raw_data["Date"].apply(lambda x: time.strptime(x,␣
↪"%Y-%m-%d")[0]*10000+
                                    time.strptime(x, "%Y-%m-%d")[1]*100+
                                    time.strptime(x, "%Y-%m-%d")[2])
# df1.dataframeName = 'SP500_test.csv'
nRow, nCol = raw_data.shape
print(f'There are {nRow} rows and {nCol} columns')

# data.drop(["Date","Adj Close"],axis=1,inplace=True)
```

```
There are 5271 rows and 6 columns
```

[52]:
```
raw_data["y"] = raw_data["Close"].diff(1).shift(-1).apply(lambda x: 1 if x > 0␣
↪else 0)
```

[53]:
```
raw_data.tail(5)
```

[53]:

	Date	Open	High	Low	Close
5266	20201207	3694.729980	3697.409912	3678.879883	3691.959961
5267	20201208	3683.050049	3708.449951	3678.830078	3702.250000
5268	20201209	3705.979980	3712.389893	3660.540039	3672.820068
5269	20201210	3659.129883	3678.489990	3645.179932	3668.100098
5270	20201211	3656.080078	3665.909912	3633.399902	3663.459961

\

	Volume	y
5266	4788560000	1
5267	4549670000	0
5268	5209940000	0
5269	4618240000	0
5270	4367150000	0

[54]:
```
raw_data.corr()
```

[54]:

	Date	Open	High	Low	Close	Volume	y
Date	1.000000	0.834010	0.834031	0.834087	0.834243	0.594475	0.036526
Open	0.834010	1.000000	0.999868	0.999797	0.999667	0.256370	0.015954
High	0.834031	0.999868	1.000000	0.999725	0.999821	0.259663	0.015322
Low	0.834087	0.999797	0.999725	1.000000	0.999839	0.251457	0.015358
Close	0.834243	0.999667	0.999821	0.999839	1.000000	0.255507	0.014487
Volume	0.594475	0.256370	0.259663	0.251457	0.255507	1.000000	0.022911
y	0.036526	0.015954	0.015322	0.015358	0.014487	0.022911	1.000000

[55]:
```
raw_data.describe()
```

[55]:

	Date	Open	High	Low	Close
count	5.271000e+03	5271.000000	5271.000000	5271.000000	5271.000000
mean	2.010048e+07	1648.424131	1658.038902	1637.971133	1648.603365
std	6.041693e+04	667.361682	669.474109	664.966367	667.452044

\

```
min    2.000010e+07   679.280029   695.270020   666.789978   676.530029
25%    2.005040e+07  1162.679993  1172.174988  1153.825012  1162.559998
50%    2.010062e+07  1385.939941  1394.900024  1373.930054  1385.589966
75%    2.015092e+07  2067.405029  2077.340088  2057.035034  2067.764893
max    2.020121e+07  3705.979980  3712.389893  3678.879883  3702.250000

             Volume            y
count  5.271000e+03  5271.000000
mean   3.177191e+09     0.536900
std    1.519290e+09     0.498684
min    3.560700e+08     0.000000
25%    1.744745e+09     0.000000
50%    3.283490e+09     1.000000
75%    4.007750e+09     1.000000
max    1.145623e+10     1.000000
```

feature engineering

[57]:

```
data = raw_data[["Close", "Volume","Date","y"]].copy()
data["Close_MA2"] = data.Close.rolling(window=2).mean()
data["Close_MA3"] = data.Close.rolling(window=3).mean()
data["Close_MA4"] = data.Close.rolling(window=4).mean()
data["Close_MA5"] = data.Close.rolling(window=5).mean()
data["Close_MA6"] = data.Close.rolling(window=6).mean()

data["Var"] = (data.Close.rolling(window=2).std()**2/2)
data["Var_MA2"] = data.Var.rolling(window=2).mean()
data["Var_MA3"] = data.Var.rolling(window=3).mean()
data["Var_MA4"] = data.Var.rolling(window=4).mean()
data["Var_MA5"] = data.Var.rolling(window=5).mean()
data["Var_MA6"] = data.Var.rolling(window=6).mean()
```

[58]:

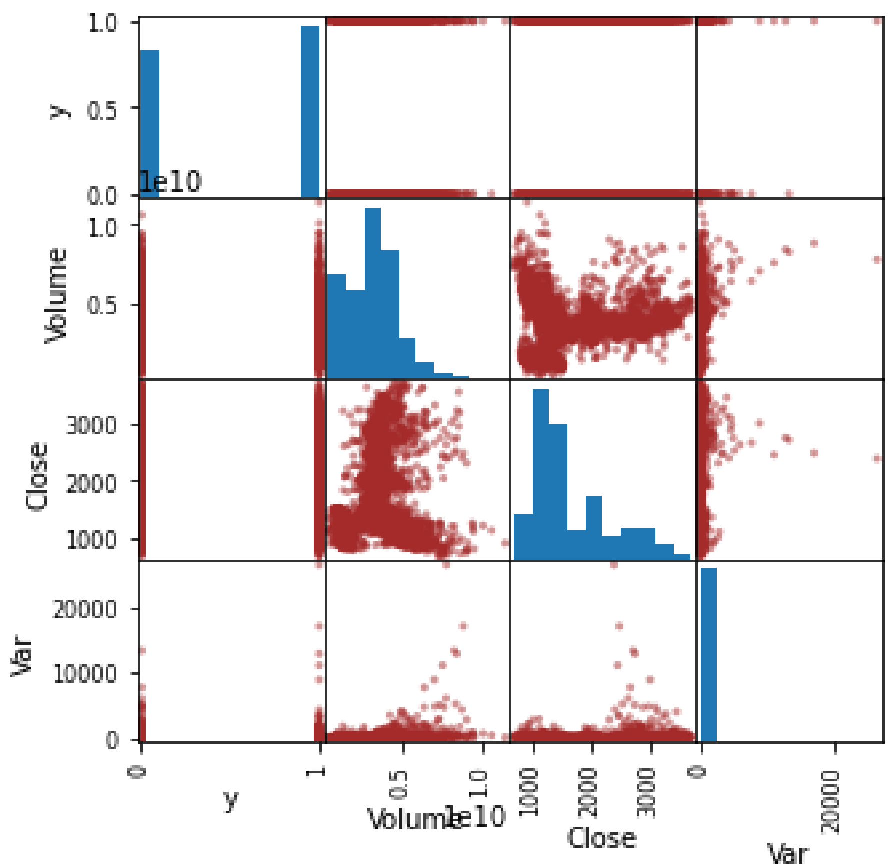
```
data.tail()
```

[58]:

```
            Close      Volume      Date  y    Close_MA2    Close_MA3  \
5266  3691.959961  4788560000  20201207  1  3695.540039  3685.933350
5267  3702.250000  4549670000  20201208  0  3697.104981  3697.776693
5268  3672.820068  5209940000  20201209  0  3687.535034  3689.010010
5269  3668.100098  4618240000  20201210  0  3670.460083  3681.056722
5270  3663.459961  4367150000  20201211  0  3665.780030  3668.126709

        Close_MA4    Close_MA5    Close_MA6         Var     Var_MA2  \
5266  3681.702515  3677.852002  3668.481649   12.816958  137.629662
5267  3690.012512  3685.812012  3681.918335   26.471226   19.644092
5268  3691.537536  3686.574023  3683.646688  216.530224  121.500725
5269  3683.782532  3686.850049  3683.495036    5.569529  111.049877
5270  3676.657532  3679.718018  3682.951701    5.382718    5.476124
```

	Var_MA3	Var_MA4	Var_MA5	Var_MA6
5266	92.190131	71.832247	140.779695	128.964979
5267	100.576850	75.760405	62.760042	121.728283
5268	85.272803	129.565193	103.914369	88.388406
5269	82.856993	65.346984	104.766061	87.523562
5270	75.827490	63.488424	53.354131	88.202170

[65]:
```
# Matrix of pairwise scatter plots
axes = pd.plotting.scatter_matrix(data.loc[:,["y","Volume", "Close", "Var"]],␣
↪figsize = [5,5], color="brown")
```

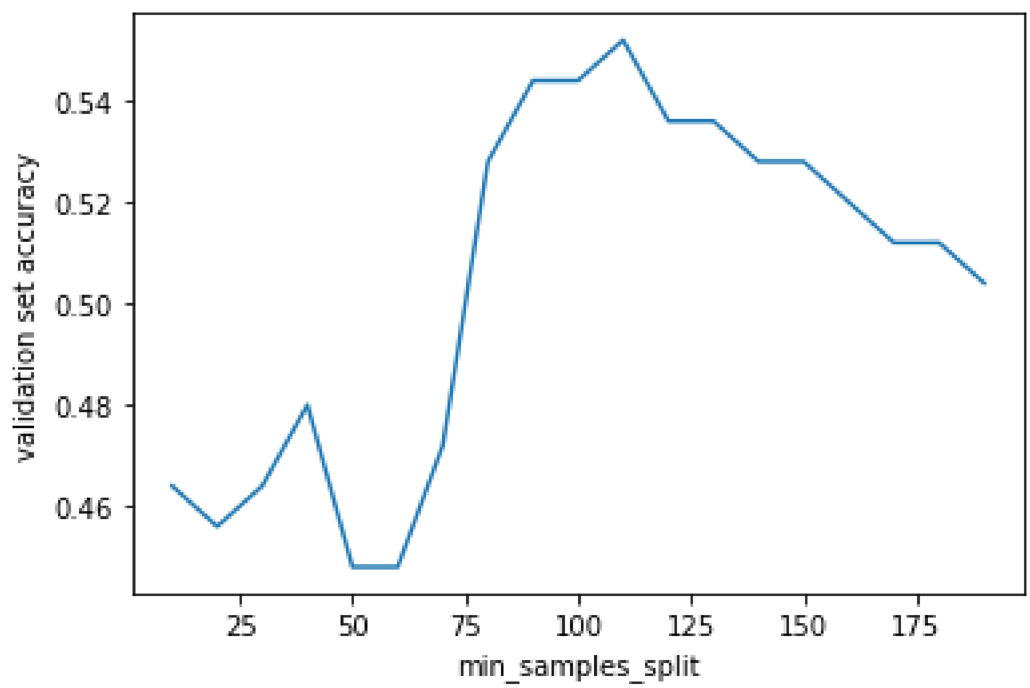


[97]:
```
train = data.loc[(data["Date"] > 20120000) & (data["Date"] < 20191231)]
valid = data.loc[(data["Date"] > 20200000) & (data["Date"] < 20200631)]
test = data.loc[(data["Date"] > 20200631)]
train2 = pd.concat( [train, valid], axis=0)
```

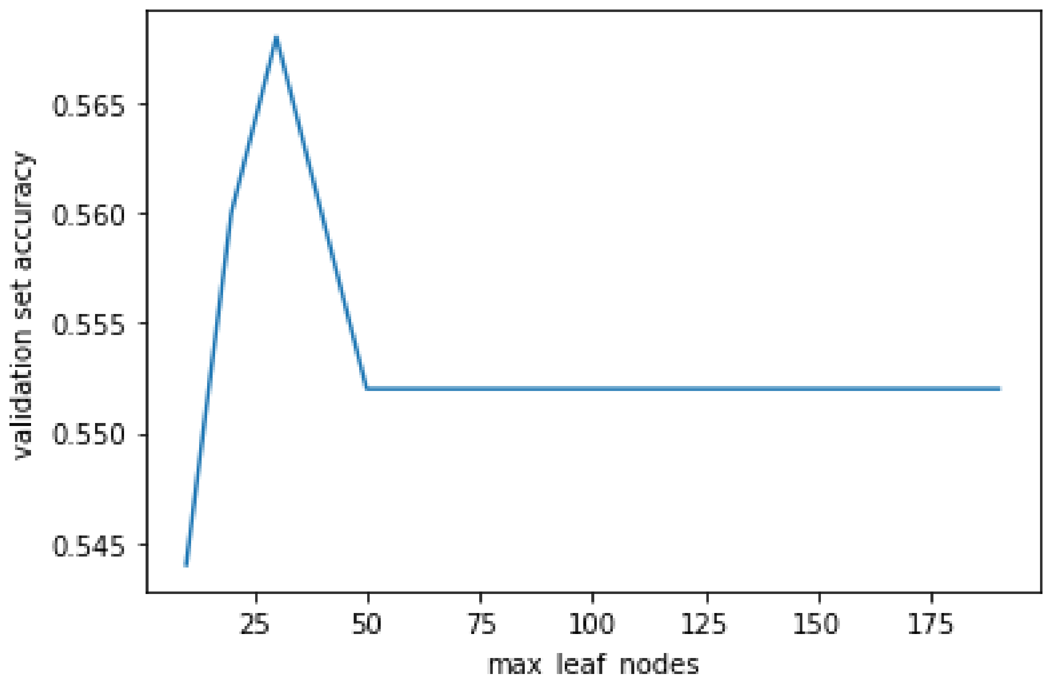
data modeling

[98]:
```
# 1. train, valid, test split
y_train = train.loc[:,"y"]
X_train = train.drop(["Date","y"],axis=1)

y_valid = valid.loc[:,"y"]
X_valid = valid.drop(["Date","y"],axis=1)

y_test = test.loc[:,"y"]
X_test = test.drop(["Date","y"],axis=1)

# 2. train, test split
y_train2 = train2.loc[:,"y"]
X_train2 = train2.drop(["Date","y"],axis=1)
```

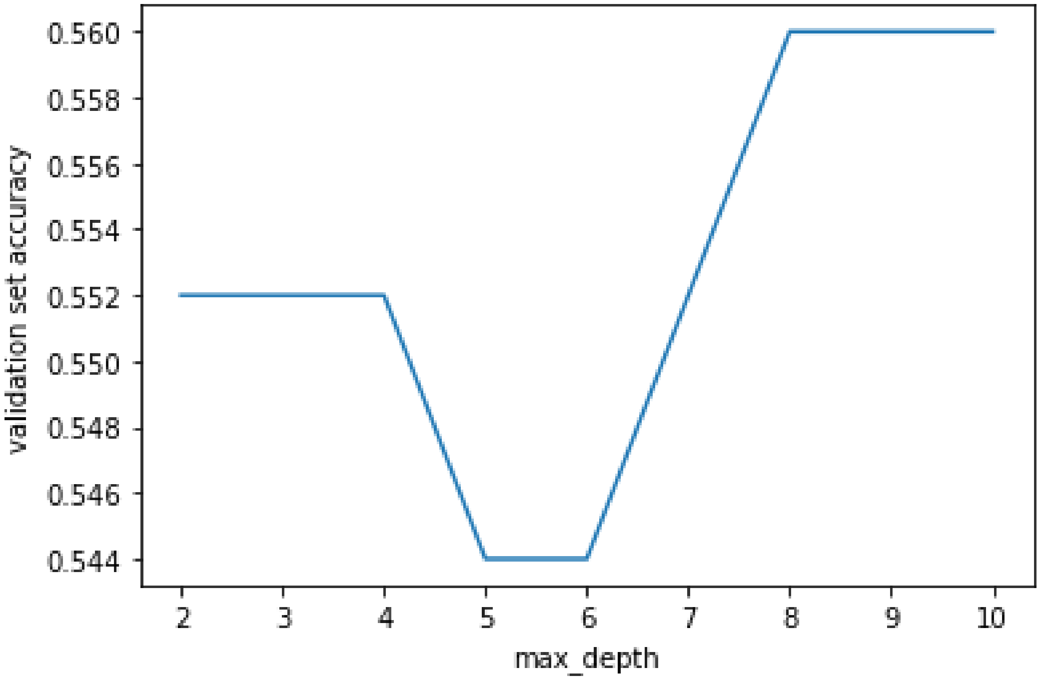
[84]:
```
# 2. Decision Tree
MSS = range(10, 200, 10)
accuracy = []
for min_samples_split in MSS:
    clf = tree.DecisionTreeClassifier(min_samples_split=min_samples_split)
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(MSS, accuracy);
plt.xlabel("min_samples_split");
plt.ylabel("validation set accuracy");
```



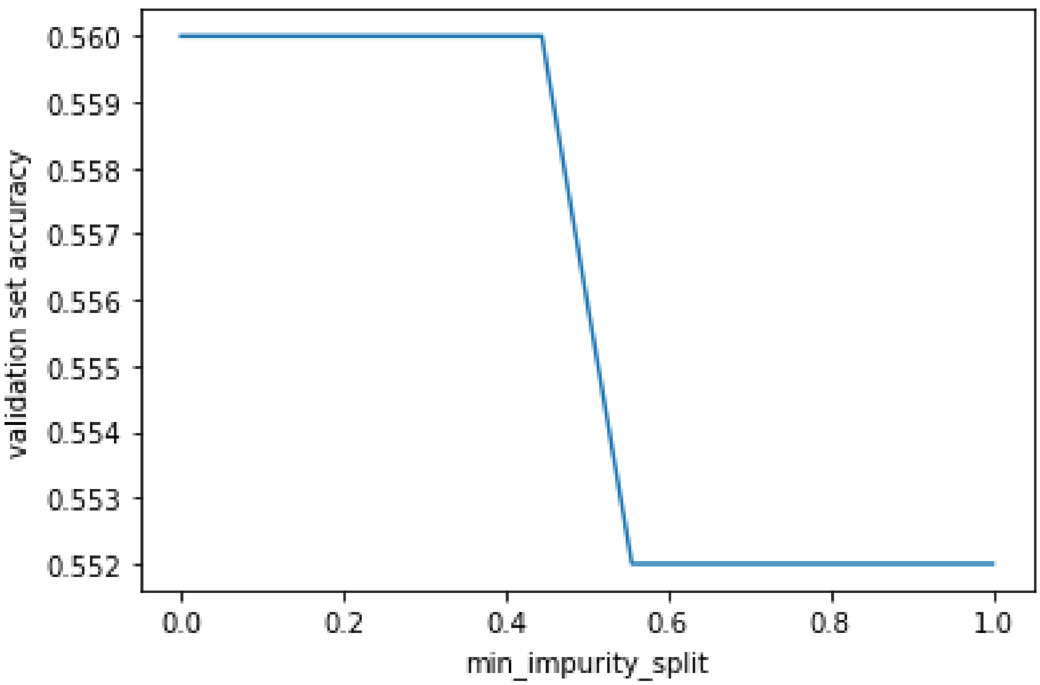
[91]:
```
MLN = range(10, 200, 10)
accuracy = []
for max_leaf_nodes in MLN:
    clf = tree.DecisionTreeClassifier(min_samples_split=110,␣
 ↪max_leaf_nodes=max_leaf_nodes)
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(MLN, accuracy);
plt.xlabel("max_leaf_nodes");
plt.ylabel("validation set accuracy");
```



[92]:
```
MD = range(2, 11, 1)
accuracy = []
for max_depth in MD:
    clf = tree.DecisionTreeClassifier(min_samples_split= 110,␣
 ↪max_leaf_nodes=30, max_depth=max_depth)
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(MD, accuracy);
plt.xlabel("max_depth");
plt.ylabel("validation set accuracy");
```



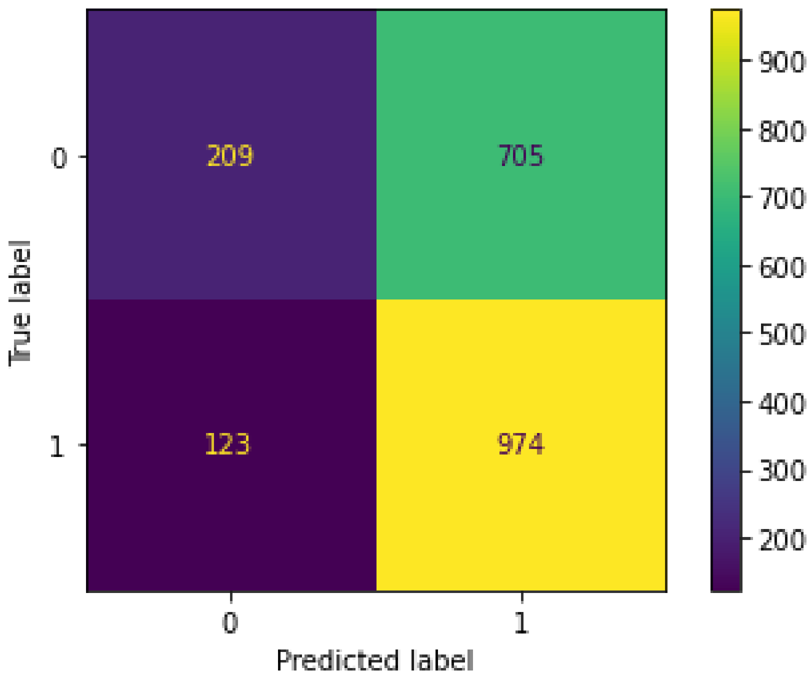
[93]:
```
MIS = np.linspace(0, 1, num=10)
accuracy = []
for min_impurity_split in MIS:
    clf = tree.DecisionTreeClassifier(min_samples_split= 110,␣
 ↪max_leaf_nodes=30, max_depth=8,
                                      min_impurity_split=min_impurity_split)
    clf.fit(X_train, y_train)
    ypred = clf.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(MIS, accuracy);
plt.xlabel("min_impurity_split");
plt.ylabel("validation set accuracy");
```



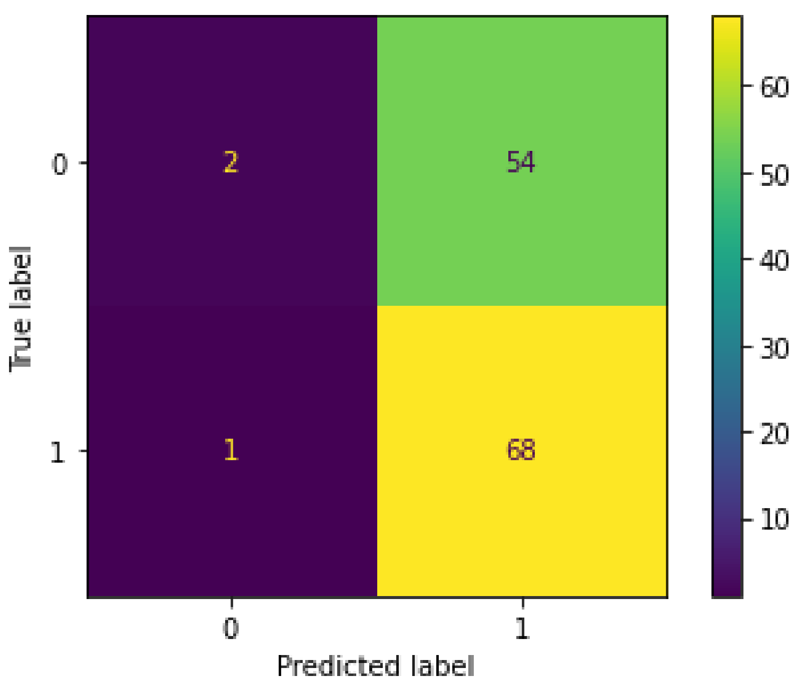
[136]:
```
#Fit Model
clf_gini = tree.DecisionTreeClassifier(min_samples_split=110,␣
 ↪max_leaf_nodes=30, max_depth=8,
                                       min_impurity_split=0.4,␣
 ↪criterion='gini') #If depth is not set, it constructs a very deep tree
clf_gini = clf_gini.fit(X_train, y_train)

print('The Decision Tree Classifer accuracy rate is',  accuracy_score(clf_gini.
 ↪predict(X_valid), y_valid))
```

```
The Decision Tree Classifer accuracy rate is 0.56
```

[137]:
```
plot_confusion_matrix(clf_gini, X_train, y_train)
plt.show()
```

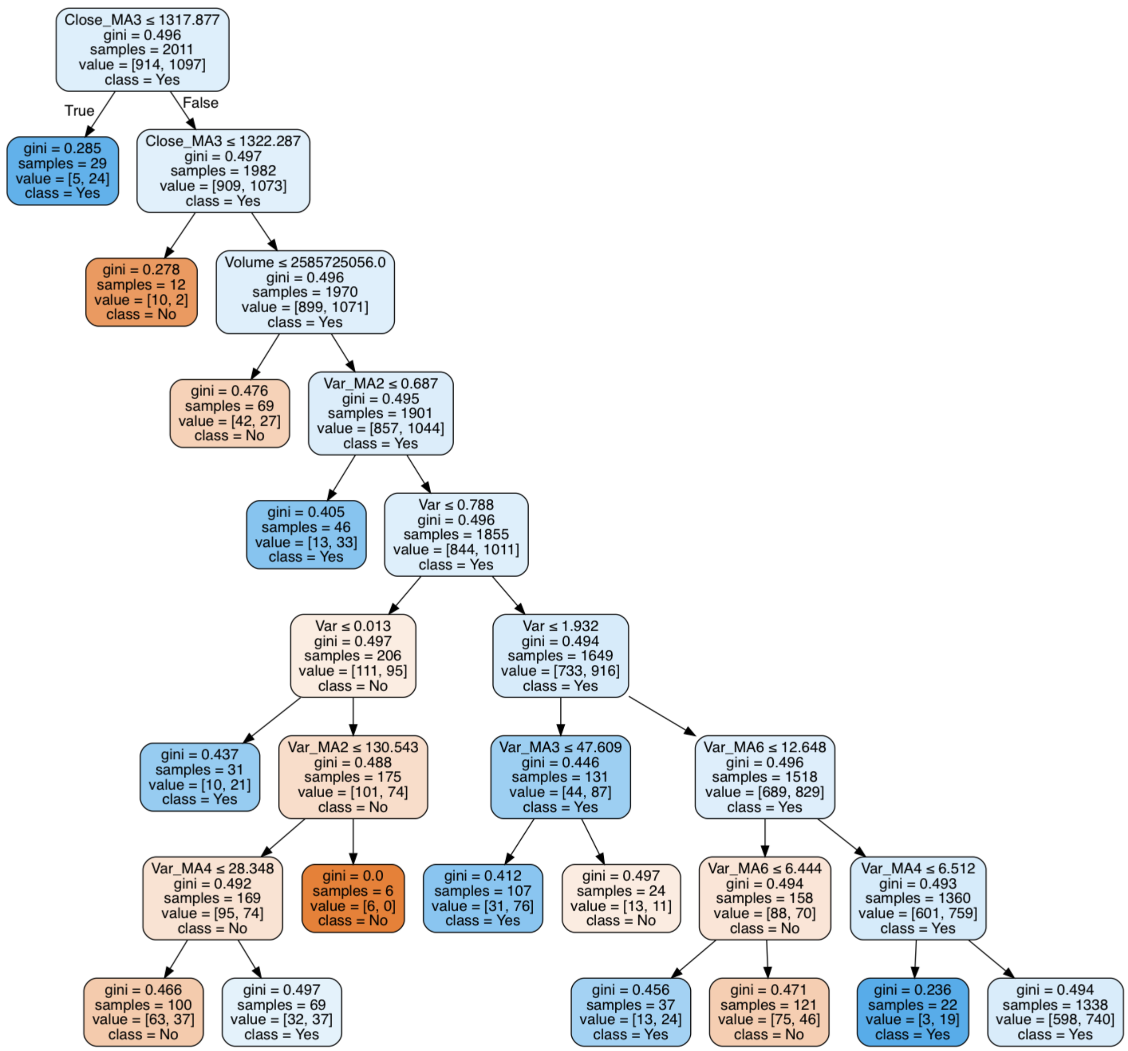


[138]:
```
plot_confusion_matrix(clf_gini, X_valid, y_valid)
plt.show()
```



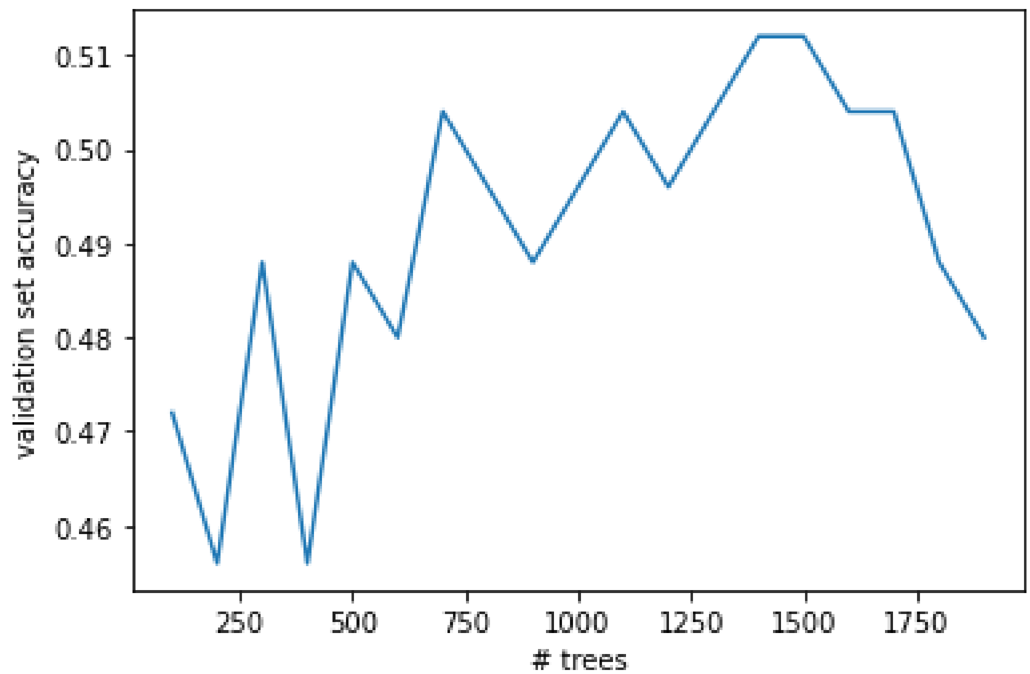
[102]:
```
import pydotplus
from IPython.display import Image
dot_data = tree.export_graphviz(clf_gini, out_file=None,
                               feature_names=list(X_train.columns), #names of the␣
 ↪features being used
                               class_names=['No','Yes'], #for categorical variables␣
 ↪only
                               filled=True, rounded=True,
                               special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)

Image(graph.create_png())
```

[102]:



[111]:
```
# 3. Fit a random forest by setting the max_features paramater to 'sqrt'
num_trees = range(100, 2000, 100)
```

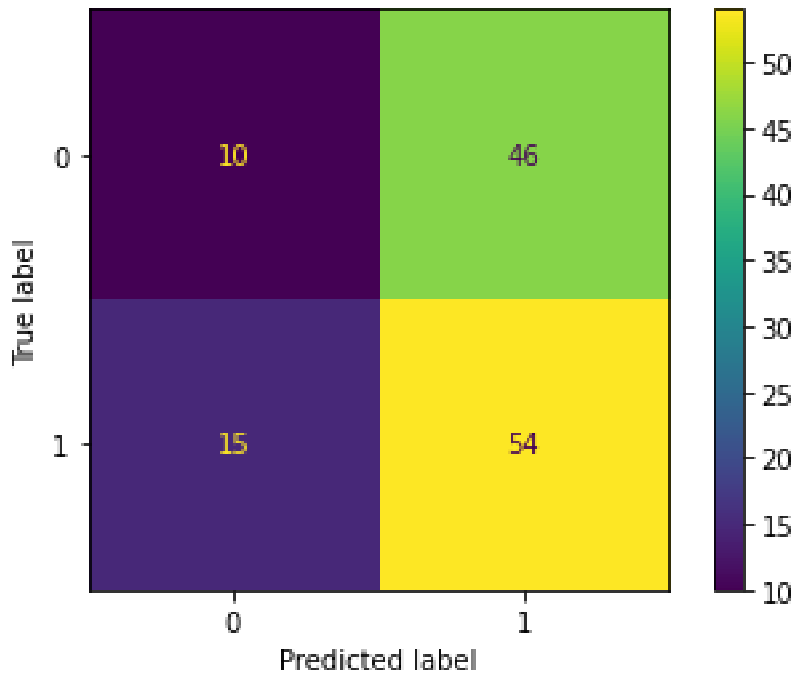
```
accuracy = []
for num_tree in num_trees:
    reg = RandomForestClassifier(max_features='sqrt', n_estimators=num_tree,␣
 ↪random_state=1)
    reg.fit(X_train, y_train)
    ypred = reg.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(num_trees, accuracy);
plt.xlabel("# trees");
plt.ylabel("validation set accuracy");
```



[140]:
```
# Fit a random forest by setting the max_features paramater to 'sqrt'
rgc = RandomForestClassifier(max_features='sqrt',random_state=1,␣
 ↪n_estimators=1500)
rgc.fit(X_train, y_train)
print('The Random Forest Classifer accuracy rate is',  accuracy_score(rgc.
 ↪predict(X_valid), y_valid))
```

```
The Random Forest Classifer accuracy rate is 0.512
```

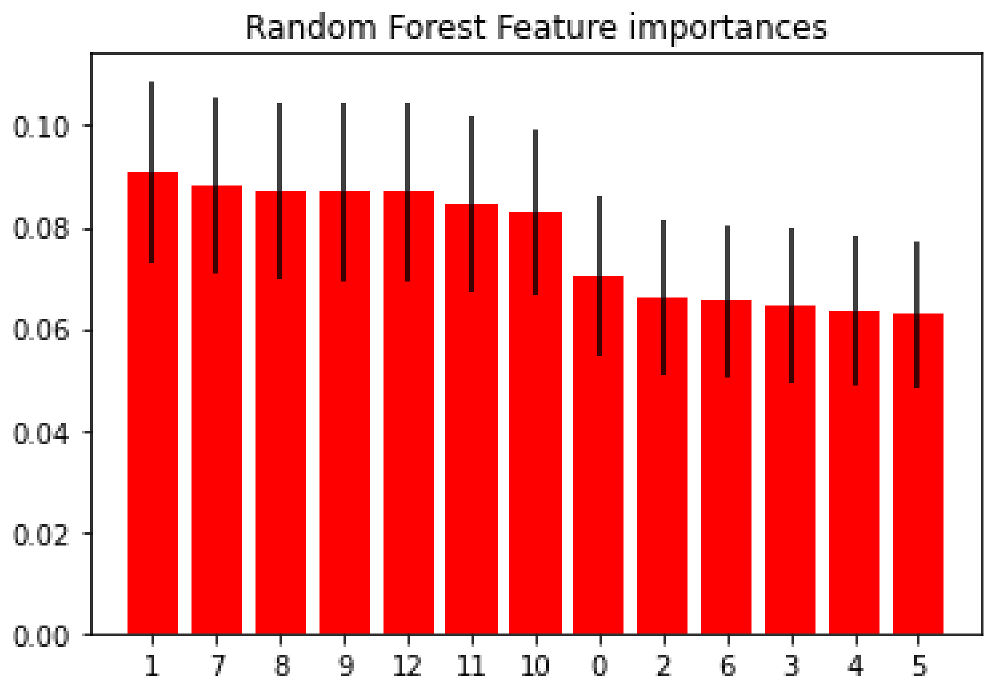
[141]:
```
plot_confusion_matrix(rgc, X_valid, y_valid)
plt.show()
```



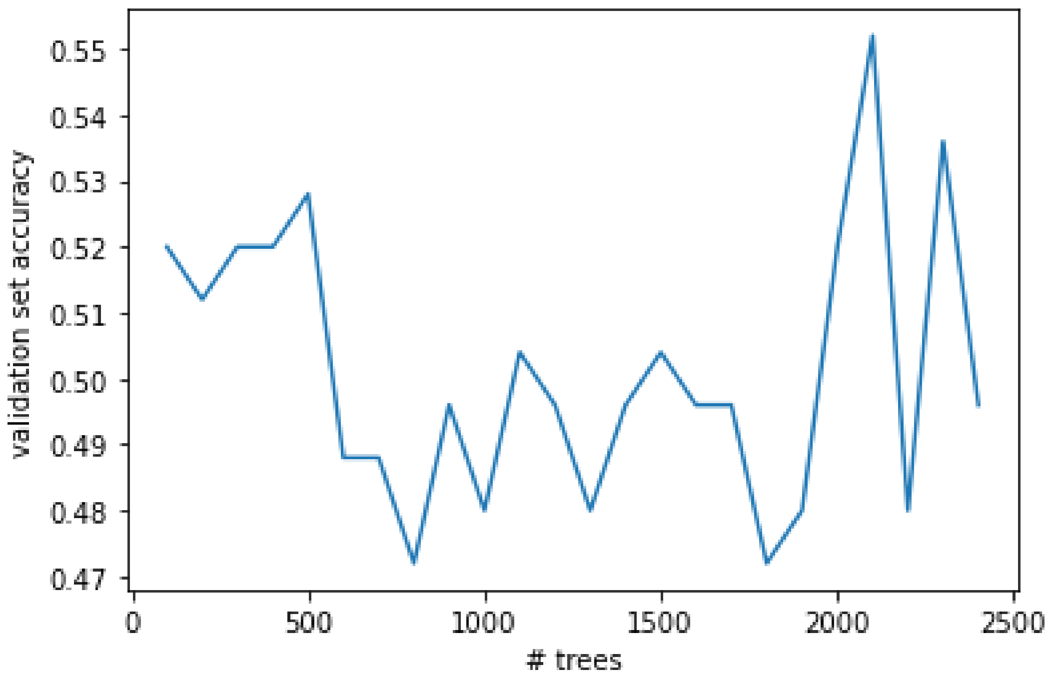
[115]:
```
importances = rgc.feature_importances_
std = np.std([tree.feature_importances_ for tree in rgc.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print('Feature Ranking:')
columns = np.array(X_train.columns)

for f in range(X_train.shape[1]):
    print("%d. feature %d %s (%f)" % (f + 1, indices[f], columns[indices[f]],␣
 ↪importances[indices[f]]))

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Random Forest Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
        color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
Feature Ranking:
1. feature 1 Volume (0.090781)
2. feature 7 Var (0.088271)
```

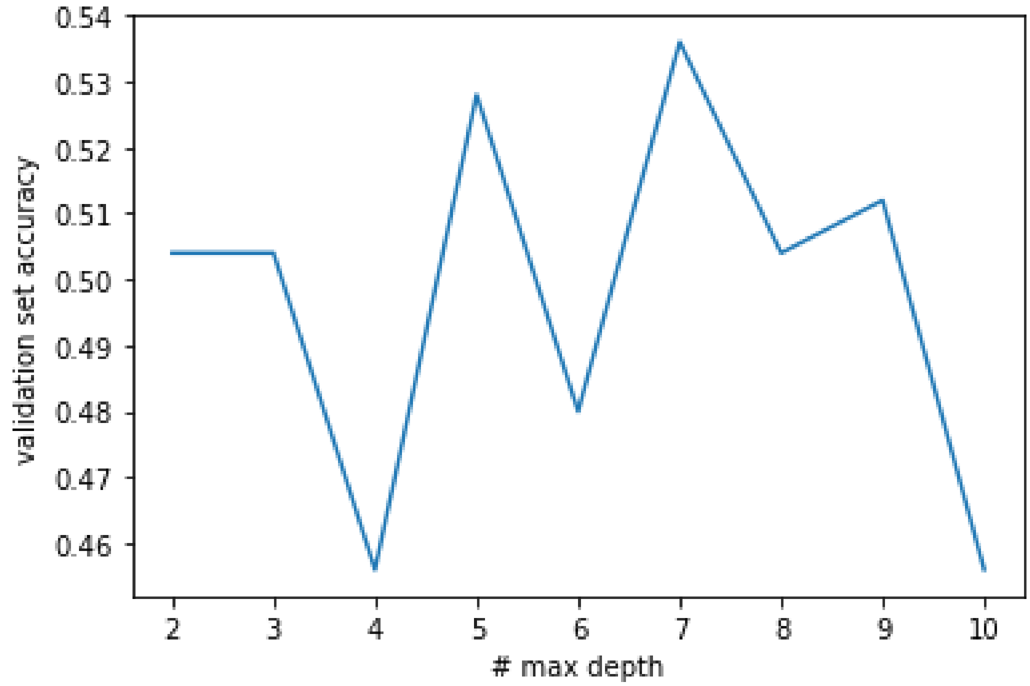
```
3. feature 8 Var_MA2 (0.087202)
4. feature 9 Var_MA3 (0.087035)
5. feature 12 Var_MA6 (0.086960)
6. feature 11 Var_MA5 (0.084376)
7. feature 10 Var_MA4 (0.082743)
8. feature 0 Close (0.070282)
9. feature 2 Close_MA2 (0.065984)
10. feature 6 Close_MA6 (0.065326)
11. feature 3 Close_MA3 (0.064672)
12. feature 4 Close_MA4 (0.063590)
13. feature 5 Close_MA5 (0.062777)
```



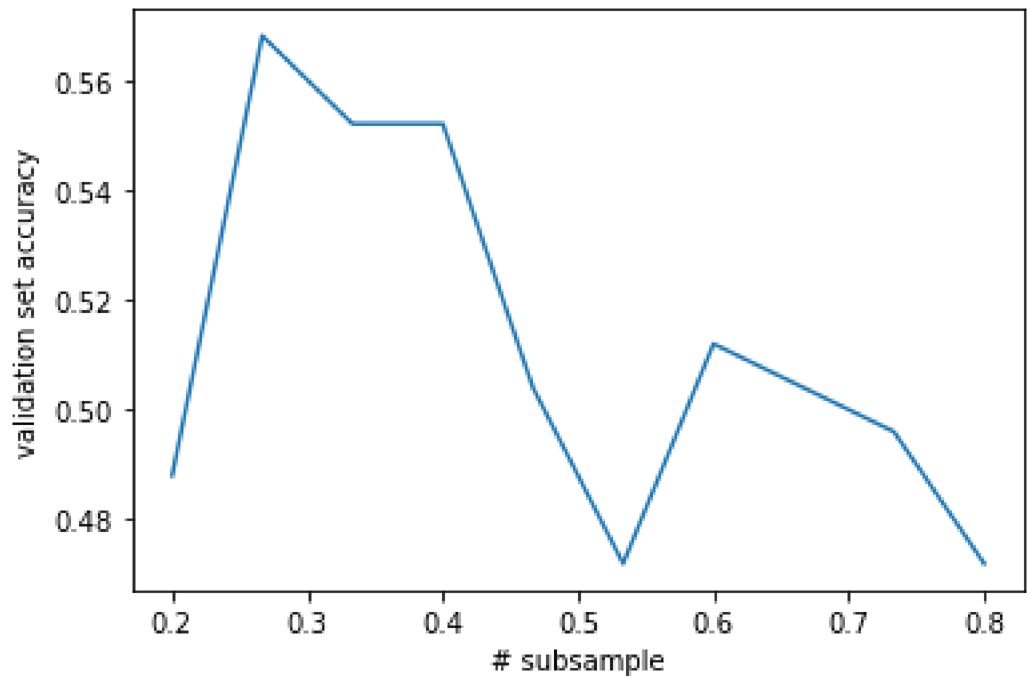
[118]:
```python
# 4. Tune parameters for a boosted tree
num_trees = range(100, 2500, 100)
accuracy = []
for num_tree in num_trees:
    reg = GradientBoostingClassifier(n_estimators=num_tree)
    reg.fit(X_train, y_train)
    ypred = reg.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(num_trees, accuracy);
plt.xlabel("# trees");
plt.ylabel("validation set accuracy");
```



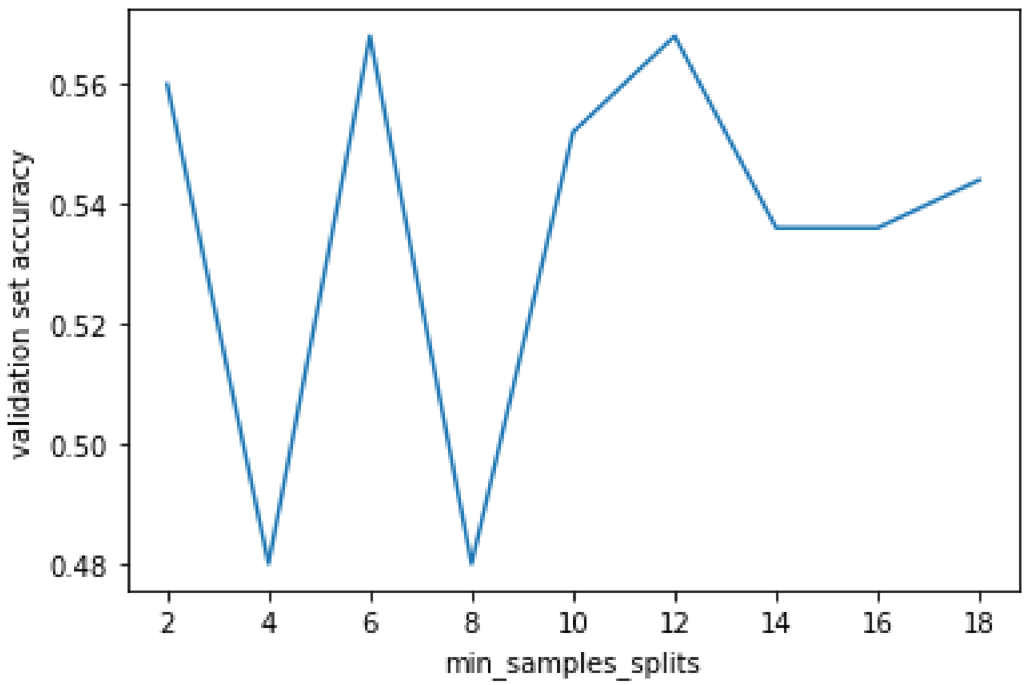
[119]:
```
depths = range(2, 11, 1)
accuracy = []
for depth in depths:
    reg = GradientBoostingClassifier(n_estimators=2100, max_depth=depth)
    reg.fit(X_train, y_train)
    ypred = reg.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(depths, accuracy);
plt.xlabel("# max depth");
plt.ylabel("validation set accuracy");
```



[120]:
```
num_subsamples = np.linspace(0.2, 0.8, num=10) # normally 0.5 - 0.8
accuracy = []
for num_subsample in num_subsamples:
    reg = GradientBoostingClassifier(n_estimators=2100, max_depth=7,
                                     subsample=num_subsample)
    reg.fit(X_train, y_train)
    ypred = reg.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(num_subsamples, accuracy);
plt.xlabel("# subsample");
plt.ylabel("validation set accuracy");
```



[121]:
```
min_samples_splits = range(2, 20, 2)
accuracy = []
for min_samples_split in min_samples_splits:
    reg = GradientBoostingClassifier(n_estimators=2100, max_depth=7,
                                     subsample=0.3,␣
 ↪min_samples_split=min_samples_split)
    reg.fit(X_train, y_train)
    ypred = reg.predict(X_valid)
    accuracy.append(accuracy_score(y_valid, ypred))
plt.plot(min_samples_splits, accuracy);
plt.xlabel("min_samples_splits");
plt.ylabel("validation set accuracy");
```

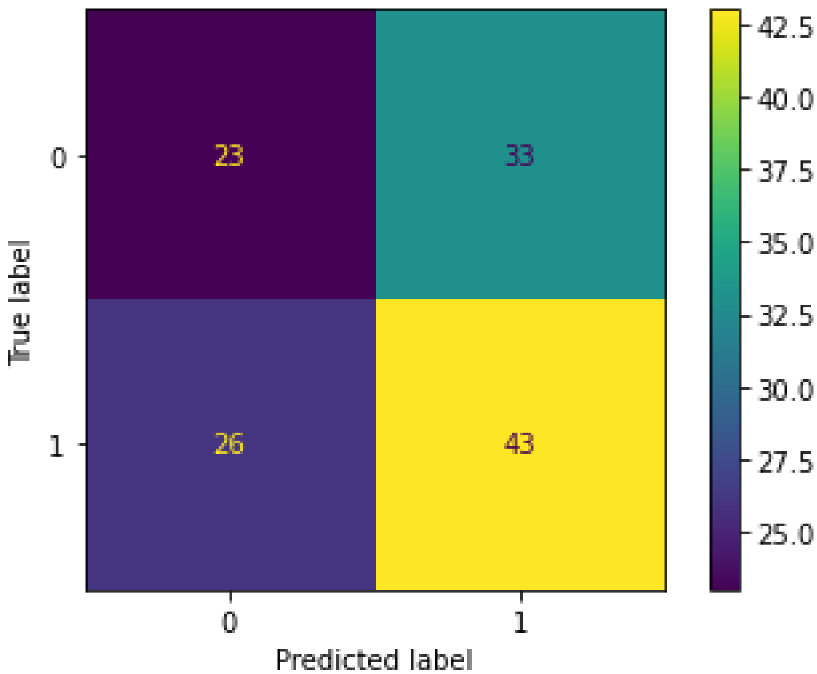


[142]:
```
# 4. Fit a boosted tree
gbc = GradientBoostingClassifier(n_estimators=2100, max_depth=7)
gbc.fit(X_train, y_train.values.ravel())
print('The Boosted Tree Classifer accuracy rate is',  accuracy_score(gbc.
 ↪predict(X_valid), y_valid))
gbc.score(X_train, y_train), gbc.score(X_valid, y_valid)
```

```
The Boosted Tree Classifer accuracy rate is 0.512
```

[142]: (1.0, 0.512)

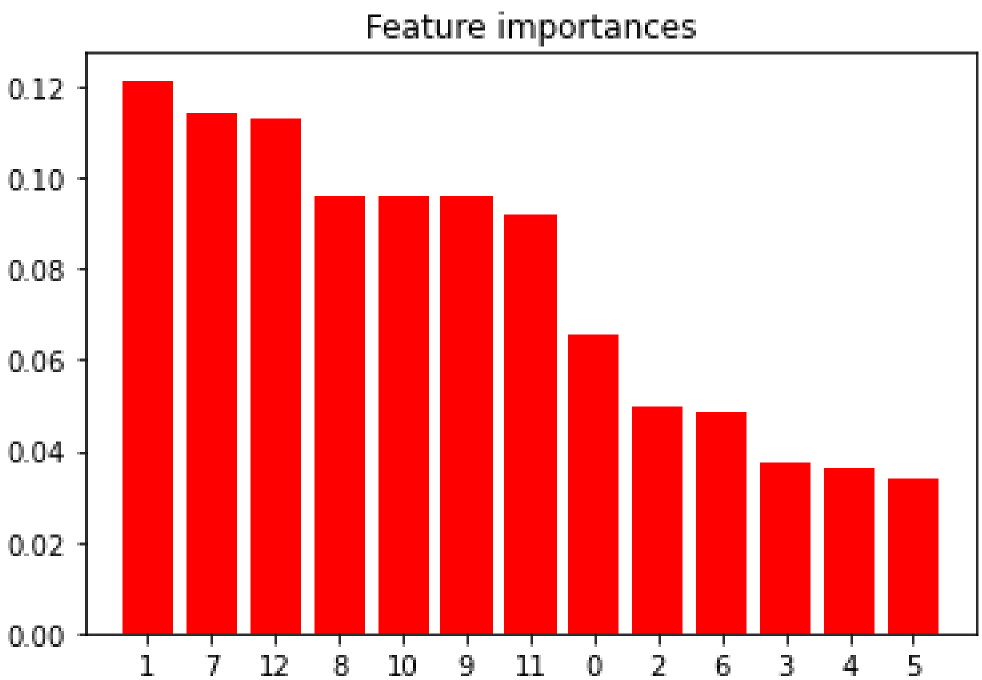
[132]:
```
plot_confusion_matrix(gbc, X_valid, y_valid)
plt.show()
```



[156]:
```
importances = gbc.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print('Feature Ranking:')
columns = np.array(X_train.columns)

for f in range(X_train.shape[1]):
    print("%d. feature %d %s (%f)" % (f + 1, indices[f], columns[indices[f]],␣
↪importances[indices[f]]))

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
        color="r", align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

```
Feature Ranking:
1. feature 1 Volume (0.121392)
2. feature 7 Var (0.114272)
3. feature 12 Var_MA6 (0.113276)
4. feature 8 Var_MA2 (0.096270)
```

```
5. feature 10 Var_MA4 (0.095974)
6. feature 9 Var_MA3 (0.095726)
7. feature 11 Var_MA5 (0.091840)
8. feature 0 Close (0.065706)
9. feature 2 Close_MA2 (0.049811)
10. feature 6 Close_MA6 (0.048544)
11. feature 3 Close_MA3 (0.037184)
12. feature 4 Close_MA4 (0.036282)
13. feature 5 Close_MA5 (0.033723)
```

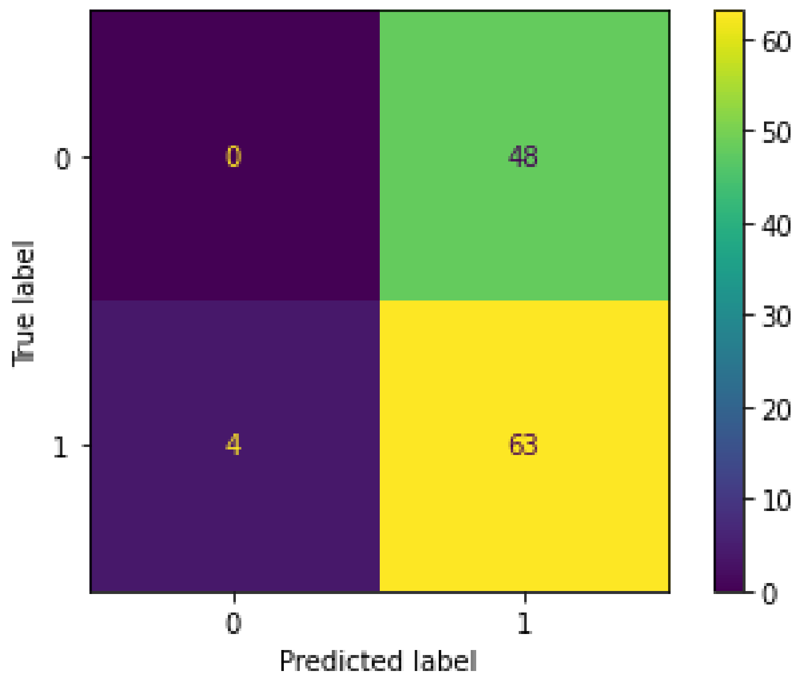


[144]:
```python
# validation accuracy rate
print('The Decision Tree (gini) Classifer accuracy rate is', ␣
 ↪accuracy_score(clf_gini.predict(X_valid), y_valid))
print('The Random Forest Classifer accuracy rate is',  accuracy_score(rgc.
 ↪predict(X_valid), y_valid))
print('The Boosted Tree Classifer accuracy rate is',  accuracy_score(gbc.
 ↪predict(X_valid), y_valid))
```

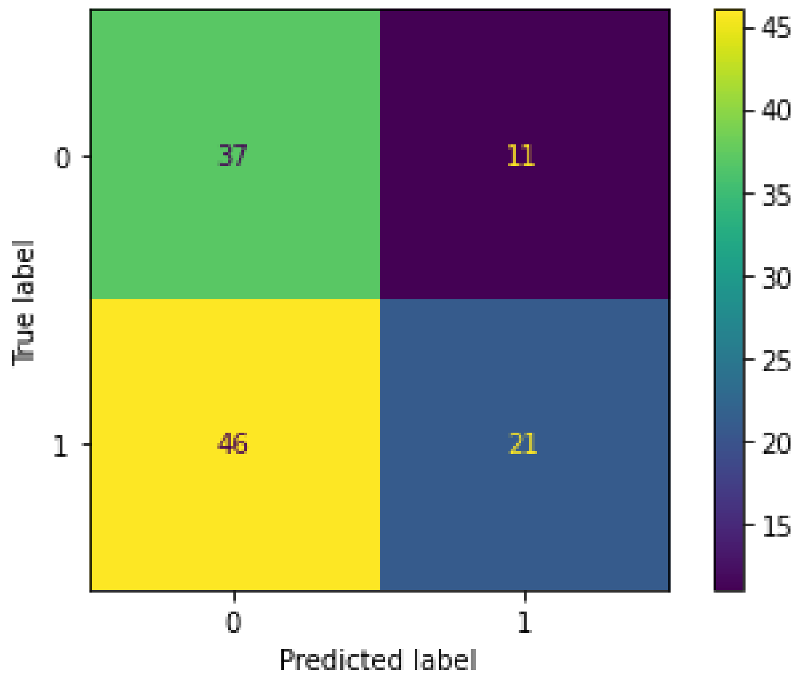
```
The Decision Tree (gini) Classifer accuracy rate is 0.56
The Random Forest Classifer accuracy rate is 0.512
The Boosted Tree Classifer accuracy rate is 0.512
```

after model comparison and tuning parameters, run models on the combined train set and test.

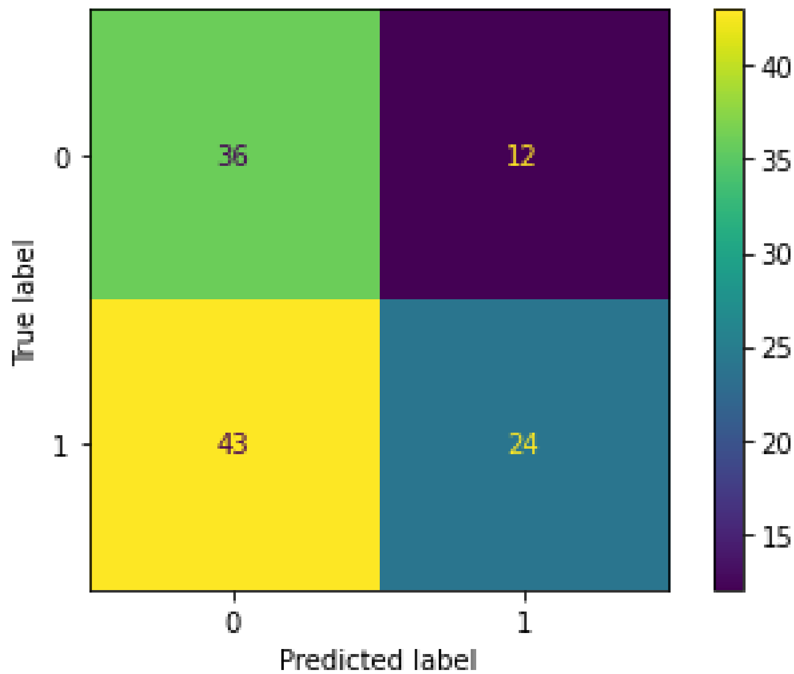
[145]:
```
# Decision Tree
clf_gini = tree.DecisionTreeClassifier(min_samples_split=110,␣
 ↪max_leaf_nodes=30, max_depth=8,
                                       min_impurity_split=0.4,␣
 ↪criterion='gini') #If depth is not set, it constructs a very deep tree
clf_gini.fit(X_train2, y_train2)
plot_confusion_matrix(clf_gini, X_test, y_test)
plt.show()
```



[146]:
```
# Random Forest
rgc = RandomForestClassifier(max_features='sqrt',random_state=1,␣
 ↪n_estimators=1500)
rgc.fit(X_train2, y_train2)
plot_confusion_matrix(rgc, X_test, y_test)
plt.show()
```



[147]:
```
# 4. Fit a boosted tree
gbc = GradientBoostingClassifier(n_estimators=2100, max_depth=7)
gbc.fit(X_train2, y_train2)
plot_confusion_matrix(gbc, X_test, y_test)
plt.show()
```



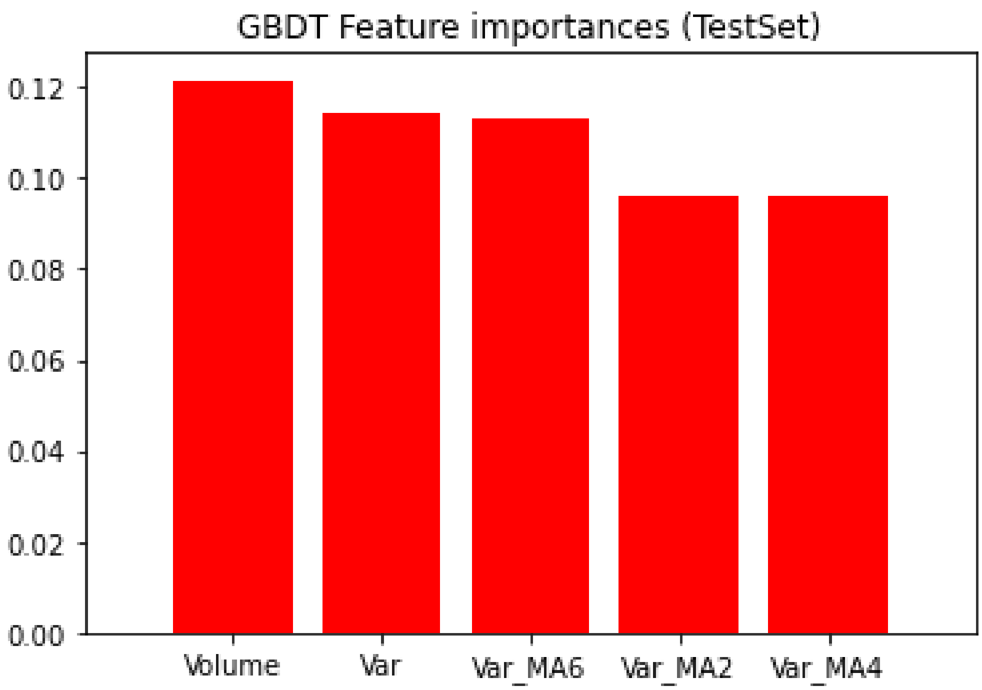
[148]:
```
print('The Decision Tree Classifer accuracy rate is',  accuracy_score(clf_gini.
 ↪predict(X_test), y_test))
print('The Random Forest Classifer accuracy rate is',  accuracy_score(rgc.
 ↪predict(X_test), y_test))
print('The Boosted Tree Classifer accuracy rate is',  accuracy_score(gbc.
 ↪predict(X_test), y_test))
```

```
The Decision Tree Classifer accuracy rate is 0.5478260869565217
The Random Forest Classifer accuracy rate is 0.5043478260869565
The Boosted Tree Classifer accuracy rate is 0.5217391304347826
```

[169]:
```
importances = gbc.feature_importances_
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print('Feature Ranking:')
columns = np.array(X_train.columns)

for f in range(X_train.shape[1]):
    print("%d. feature %d %s (%f)" % (f + 1, indices[f], columns[indices[f]],␣
 ↪importances[indices[f]]))

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("GBDT Feature importances (TestSet)")
```

```
plt.bar(range(5), importances[indices][:5],
        color="r", align="center")
plt.xticks(range(5), columns[indices][:5])
plt.xlim([-1, 5])
plt.show()
```

```
Feature Ranking:
1. feature 1 Volume (0.121392)
2. feature 7 Var (0.114272)
3. feature 12 Var_MA6 (0.113276)
4. feature 8 Var_MA2 (0.096270)
5. feature 10 Var_MA4 (0.095974)
6. feature 9 Var_MA3 (0.095726)
7. feature 11 Var_MA5 (0.091840)
8. feature 0 Close (0.065706)
9. feature 2 Close_MA2 (0.049811)
10. feature 6 Close_MA6 (0.048544)
11. feature 3 Close_MA3 (0.037184)
12. feature 4 Close_MA4 (0.036282)
13. feature 5 Close_MA5 (0.033723)
```



The decision tree has very low prediction power because it classify most of samples as 1. Boosted Tree Classifer and Random Forest Classifer both achieve reasonable results with the same validation accuracies. But there are little difference that boosting tree tend to classify more samples as 1. Since the features have poor prediction power, I don't use AUC but suggest to further compare

AUC, recall rate.

```
[ ]:
```