Options Pricing Project

University of Southern California

DSO 530: Applied Modern Statistical Learning Methods

May 2, 2024

Group 49
Jessica Bratahani - 7041404049
Pin Hsuan Chang - 3949731079
Suhan Ho - 9648292115
Sheena Huang - 7622640822
Yunchi Lee - 1283919193
Contact Person Email: jessica.bratahani.2024@marshall.usc.edu

Executive Summary

In this project, we explore the application of supervised machine learning methods to predict the pricing of European call options on the S&P 500. The ability to predict option prices accurately is important for investors managing risks and seeking profitable opportunities.

Our team applied various predictive models including Linear Regression, Logistic Regression, K-Nearest Neighbors, Decision Trees, and Random Forests to datasets provided. The focus was on two main tasks: regression to predict option values and classification to predict compliance with the Black-Scholes (BS) model, whether the options value is over or underestimated.

Based on our experiments, the Random Forest model is selected for both tasks as it achieved a mean R-squared value of 0.996 in 5-fold cross-validation for regression. For the classification task, the model demonstrated an impressive accuracy rate of 93.66% from 10-fold cross validation. These results show the potential of machine learning models to enhance the traditional option pricing methods like the Black-Scholes formula.

The implications of our findings suggest that machine learning can significantly enhance financial analytics, providing more dynamic and robust tools for price prediction. Future work could explore further enhancements in model accuracy and adaptability under various market conditions.

However, when applying these machine learning results to investments with unknown future characteristics, it is crucial to exercise caution. Our team does not suggest solely relying on ML predictions for decision-making. Instead, we recommend integrating these insights with robust risk management strategies, market dynamics, expert opinion, and quantitative financial Analysis. This balanced approach ensures that while leveraging the predictive power of AI, investors also safeguard against potential uncertainties in financial markets.

In this report, we will introduce the machine learning methods we used, discuss our selection process, and cover key considerations for applying these predictions in business scenarios.

Methodology

I. Regression

Review of Approaches

In order to predict Current Option Value (Value) from the parameters Current Asset Value (S), Strike Price (K), Time to Maturity (tau) and Interest Rate (R), a variety of supervised learning methods for regression were explored. The methods our group explored includes: Linear Regression, Lasso and Ridge Regression, K-Nearest Neighbors (KNN) regression, Decision Tree and Random Forest for regression.

In the case of linear regression, we also explored best subset selection to find the impact of the number of parameters used in the model to model performance. Since there are only 4 parameters in this case, best subset selection is still computationally feasible and was our selected choice compared to forward or backward stepwise selection. However, our findings (as shown in Figure 1 in the appendix) suggest that using 3 vs. 4 predictors did not impact the in-sample R-square significantly as both models resulted in having an R-squared value of 0.925. In the next section, we will review how we selected our final model and discuss further actions that could be taken to improve our predictions.

Selection and Summary of Final Approach

To find the best model, our team used k-fold cross validation, with 5 and 10 folds, with the training data, across the models and compared them based on mean r-squared and mean-squared error (MSE) metrics. K-fold cross validation is a non-parametric evaluation method, thus allowing us to assess each model's generalization capability. For our comparison, the training data parameters was standardized using Scikit-learn's StandardScaler().
Table 1 below illustrates the comparison of the models.

Model	KFolds	Mean R Squared	Mean MSE
Lasso	5	0.924447	1182.78
LinearRegression	5	0.924657	1179.48
Ridge	5	0.924658	1179.47
KNeighborsRegressor	5	0.990365	150.12
DecisionTreeRegressor	5	0.992236	120.97
RandomForestRegressor	5	0.996515	54.40
Lasso	10	0.924599	1183.26

LinearRegression	10	0.924821	1180.05
Ridge	10	0.924821	1180.04
KNeighborsRegressor	10	0.991001	140.81
DecisionTreeRegressor	10	0.993242	105.15
RandomForestRegressor	10	0.996776	50.43

Table 1: K-Fold Cross Validation on Regression Models

Random Forest Regressor is the model with the highest mean R-squared(0.996) and lowest MSE(54.40) compared to other regression models for both 5 and 10 fold cross validation.

With Random Forest as the optimal regression model, our team decided to perform hyperparameter tuning on the random forest regressor, starting with RandomForestRegressor(random_state=0, oob_score=True) as the base model. Utilizing GridSearchCV(), allowed us to evaluate a range of parameter values and select the configuration that yielded the best performance: RandomForestRegressor(max_depth=30, max_features=3, min_samples_leaf=2, n_estimators=1000, oob_score=True, random_state=0). However, when implementing 5-fold cross validation on the two models, the base model still resulted in higher mean R-squared and lower MSE, therefore we decided to stay with the base random forest model as our final model for regression.

For our final prediction on Value from the test data, we used StandardScaler transform() on the test data and made the prediction on our final regression model fitted on the whole training dataset (with scaled X values and y training data).

Further Steps

To enhance our model further, we can broaden our scope by incorporating additional parameter values during the tuning process. Additionally, we could expand our exploration by incorporating other regression models, such as Neural Networks, to leverage their potential for capturing complex relationships within the data.

II. Classification

Review of Approaches

Classification methods were used in predicting Black Sholes' performance (whether the options value is over or underestimated) from the same 4 variables used to predict Value in regression. The models we tried for classification include Logistic Regression, KNN for classification, Decision Tree, Gradient Boosting, Random Forest and Support Vector Classifier.

In order to assess the accuracy of our BS predictions based on those parameters, we investigated the impact of feature scaling on our results. To evaluate the different models, we split the training data into training and test data sets to fit the best model and compare the accuracy scores of different models. In addition, we also employed K-Fold cross validation in the similar manner we did to compare regression models.

Selection and Summary of Final Approach

To find the best model, our team used k-fold cross validation, with 5 and 10 folds, with the training data, across the models and compared them based on prediction accuracy of classification. Table 2 below illustrates the comparison of the classification models.

Model	KFolds	Accuracy
LogisticRegression	5	0.8784
KNeighborsClassifier	5	0.8552
DecisionTree	5	0.9124
RandomForestRegressor	5	0.9334
GradientBoosting	5	0.9264
SupportVectorClassifier	5	0.8856
LogisticRegression	10	0.8732
KNeighborsClassifier	10	0.8594
DecisionTree	10	0.9128
RandomForestRegressor	10	0.9366
GradientBoosting	10	0.9268
SupportVectorClassifier	10	N/A

Table 2: K-Fold Cross Validation on Classification Models

With Random Forest as the optimal classification model (mean accuracy of 93.66% in 10 fold cross validation), our team decided to perform hyperparameter tuning on the random forest classifier, starting with RandomForestClassifier(random_state=1, n_estimators=200) as the base model. We then used GridSearchCV() again to evaluate a range of parameter values and select the configuration that yielded the best performance: RandomForestClassifier(max_depth=6, max_leaf_nodes=9, n_estimators=200, random_state=1).

However, when implementing 10-fold cross validation on the two models, the base model still resulted in higher prediction accuracy (93.66% vs. 89.38%), therefore we decided to stay with the base random forest model as our final model for classification. With our best model, we

compared the 10-fold cross validation score when training using scaled versus non-scaled parameters, and found that the prediction accuracy improved by 0.02% from 93.66% to 93.68%. Therefore, for our final prediction on BS from the test data, we used StandardScaler transform() on the test data and made the prediction on our final classification model fitted on the whole training dataset (with scaled X values and y training data).

Further Steps

To improve our models in the future, we can explore other boosting models that can help us have a broader understanding of the dataset, such as XGBoost, LightGBM and CatBoost, and apply Neural Networks to the dataset as well. Additionally, we can carefully observe the distribution of our raw data and remove the outliers before training models.

Conclusion

Based on our findings, machine learning models can significantly enhance financial analytics, providing more dynamic and robust tools for price prediction. In the future, further enhancements in model accuracy and adaptability under various market conditions could be explored. In this project, our team has explored the application of multiple supervised machine learning models to predict the pricing of European call options on the S&P 500 and found that the Random Forest Models were the best models for both regression and classification tasks.

Machine learning models excel over traditional methods like Black-Scholes due to their adeptness at managing non-linear complexities and adapting to changing market conditions, continuously learning from new data and integrating diverse inputs to capture the complexities of financial markets. However, when applying such models to specific contexts like predicting option values for high volatility stocks, challenges arise due to limited historical data, the complexity of market dynamics, and variations in regional and industrial characteristics. These factors underscore the importance of cautious consideration and adaptation when leveraging machine learning in real-world financial scenarios.

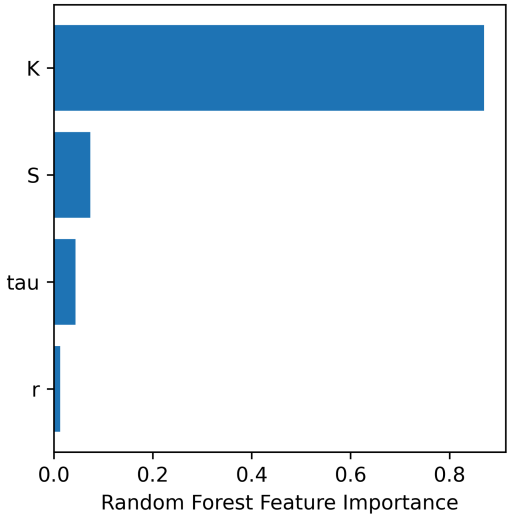
Thus, our team does not suggest solely relying on ML predictions for decision-making. Instead, we recommend integrating these insights with robust risk management strategies, market dynamics, expert opinion, and quantitative financial Analysis. This balanced approach ensures that while leveraging the predictive power of AI, investors also safeguard against potential uncertainties in financial markets.

References

- Python Code:
  DSO530 Python Tutorial 8
  DSO530 Python Tutorial 9
  DSO530 Python Tutorial 10
  DSO530 Python Tutorial 11

Appendix

1. Figure 1: Best Subset Selection on OLS Regression
   When performing best subset selection on OLS regression, the model with 3 parameters is selected to be the best subset in terms of adjusted R-squared, AIC and BIC, however it is very similar to the values from the model with 4 parameters.
2. Figure 2: Feature Importances on Random Forest Regression



The following pages part of the appendix are the PDFs of our .ipynb code. A brief overview of the codes and content in the pdfs appended are below:

3. Final Prediction Code
   a. DSO530Project_Final_Prediction.ipynb code in PDF
      - Code for final predictions of 'Value' and 'BS' using our best models on the given test data
4. Regression Code
   a. CV_Regression_Models.ipynb code in PDF
      - K-fold cross validation across different regression models
      - Hyperparameter tuning on Random Forest Regressor
      - Feature importances of random forest regressor

   b. Linear_Regression.ipynb code in PDF
      - Comparison of 3 vs. 4 parameter linear regression models
      - Best Subset Selection for linear regression model
5. Classification Code
   a. Model without scaled features
      - K-fold cross validation different classification models
      - Hyperparameter tuning on Random Forest Classifier
      - Tuning the hyperparameter for Support Vector Classifier on 10-fold cross validation cannot run on the computer
   b. Model with scaled features on 10-fold cross validation
      - Pick the models that have better performance from the unscaled version
      - Compare the cross validation across different classification models
   c. Train_Test_Split on 10-fold cross validation
      - Utilized the train_test_split on the training data to find the best model
      - Tuning the hyperparameter for Support Vector Classifier on 10-fold cross validation cannot run on the computer

DSO530Project_Final_Prediction

May 2, 2024

1 DSO 530 Project: Final Prediction

Group 49: Jessica Bratahani, Pin Hsuan Chang, Suhan Ho, Sheena Huang, Yunchi Lee

[1]:
```
import numpy as np
import pandas as pd
import time
import itertools
import matplotlib.pyplot as plt
import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.ensemble import RandomForestRegressor,RandomForestClassifier

from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import accuracy_score

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

1.1 Training and Test Data

[2]:
```
df_train=pd.read_csv('option_train.csv',index_col=0)

#Drop Duplicates if Any:
df_train = df_train.dropna()

X_train = df_train[['S','K','tau','r']]
y_train = df_train['Value']
```

[3]:
```
df_test=pd.read_csv('option_test_nolabel.csv',index_col=0)
```

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

[4]:

	S	K	tau	r
1	1409.28	1325	0.126027	0.0115
2	1505.97	1100	0.315068	0.0110
3	1409.57	1450	0.197260	0.0116
4	1407.81	1250	0.101370	0.0116
5	1494.50	1300	0.194521	0.0110

1.2 Regression - Best Model: Random Forest Regressor

[5]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(df_train[['S','K','tau','r']])
X_test_scaled = scaler.transform(df_test[['S','K','tau','r']])
```

[6]:
```
def kfolds_cv_regression(regressor, kfolds, x_train, y_train):
    kfolds_regression = KFold(n_splits = kfolds, random_state = 1, shuffle =␣
 ↪True)
    r2_model_1_cv = cross_val_score(regressor, x_train, y_train,␣
 ↪cv=kfolds_regression)
    neg_mse_model_1_cv = cross_val_score(regressor, x_train, y_train,␣
 ↪cv=kfolds_regression,scoring = 'neg_mean_squared_error')
    return np.mean(r2_model_1_cv), -np.mean(neg_mse_model_1_cv)
```

[7]:
```
regressor=RandomForestRegressor(random_state=0, oob_score=True)
KFolds=5
mean_r2, mean_mse=kfolds_cv_regression(regressor, KFolds, X_train_scaled,␣
 ↪y_train)
print('Model:', type(regressor).__name__, 'KFolds:',KFolds,' Mean R Squared:',␣
 ↪mean_r2,' Mean MSE:', mean_mse)
```

```
Model: RandomForestRegressor KFolds: 5  Mean R Squared: 0.9965151303907671  Mean
MSE: 54.397492430605475
```

[8]:
```
final_regressor=RandomForestRegressor(random_state=0, oob_score=True)

# fit the final regressor with X_train_scaled and Y_train data
final_regressor.fit(X_train_scaled, y_train)

# Predicting the target values of the test set
y_pred = final_regressor.predict(X_test_scaled)
```

[9]:
```
# Create new dataframe to store our predicted values
df_prediction = pd.DataFrame(y_pred, columns=['Value'])
df_prediction.index = range(1, len(df_prediction)+1)
```

1.3 Classification - Best Model: Random Forest Regressor

[10]:
```
df_train.head()
```

[10]:

	Value	S	K	tau	r	BS
1	348.500	1394.46	1050	0.128767	0.0116	Under
2	149.375	1432.25	1400	0.679452	0.0113	Under
3	294.500	1478.90	1225	0.443836	0.0112	Under
4	3.375	1369.89	1500	0.117808	0.0119	Over
5	84.000	1366.42	1350	0.298630	0.0119	Under

[11]:
```
df_train['BS_class'] = df_train['BS'].map({'Under': 0, 'Over': 1})
y_train_BS = df_train['BS_class']
```

[12]:
```
# Initialize the StratifiedKFold object
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
accuracies_rd = []

# Cross-validation loop
for train_index, test_index in kfolds.split(X_train_scaled, y_train_BS):
    clf_rf = RandomForestClassifier(random_state=1, n_estimators=200)
    clf_rf.fit(X_train_scaled[train_index], y_train_BS.iloc[train_index])  #␣
 ↪Train on the fold's training part
    y_pred_rf = clf_rf.predict(X_train_scaled[test_index])  # Predict on the␣
 ↪fold's testing part
    score = accuracy_score(y_train_BS.iloc[test_index], y_pred_rf)  # Calculate␣
 ↪accuracy
    accuracies_rd.append(score)

# Calculate mean and standard deviation of accuracies
mean_accuracy_rf = sum(accuracies_rd) / len(accuracies_rd)
std_accuracy_rf = np.std(accuracies_rd)

print(f"Random Forest mean accuracy: {mean_accuracy_rf:.4f}")
print(f"Random Forest standard deviation of accuracy: {std_accuracy_rf:.4f}")
```

```
Random Forest mean accuracy: 0.9368
Random Forest standard deviation of accuracy: 0.0096
```

[13]:
```
final_classifier=RandomForestClassifier(random_state=1, n_estimators=200)

# fit the final regressor with X_train and Y_train data
final_classifier.fit(X_train_scaled, y_train_BS)


# Predicting the target values of the test set
y_pred_BS = final_classifier.predict(X_test_scaled)
```

[14]:
```
df_prediction['BS'] = y_pred_BS.tolist()
```

[15]:
```
df_prediction.sample(5)
```

[15]:
```
          Value  BS
151  218.80250   0
252  122.53500   0
17    31.61125   1
422   54.66875   0
443  240.10875   0
```

[16]:
```
df_prediction['BS'].value_counts(normalize=True)
```

[16]:
```
BS
0    0.786
1    0.214
Name: proportion, dtype: float64
```

[17]:
```
df_train['BS_class'].value_counts(normalize=True)
```

[17]:
```
BS_class
0    0.7736
1    0.2264
Name: proportion, dtype: float64
```

[18]:
```
# save file to .csv for submission
df_prediction.to_csv('group_49_prediction.csv',index = False)
```

[ ]:
```

```

CV_Regression_Models

May 2, 2024

1 Cross Validation for Regression Models

DSO 530 Spring 2024
Group 49: Jessica Bratahani, Pin Hsuan Chang, Suhan Ho, Sheena Huang, Yunchi Lee

[1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import time
import itertools
import matplotlib.pyplot as plt
import statsmodels.api as sm

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor


from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

1.1 Load the training dataset and scale the variables using StandardScaler()

[2]:
```
df_train=pd.read_csv('option_train.csv',index_col=0)

#Drop Duplicates if Any:
df_train = df_train.dropna()

X_train = df_train[['S','K','tau','r']]
y_train = df_train['Value']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(df_train[['S','K','tau','r']])
```

1.2 List regression models to be compared using CV

[3]:
```
regression_models= [LinearRegression(),
         Ridge(), Lasso(),
         DecisionTreeRegressor(random_state = 0),
         RandomForestRegressor(random_state=0, oob_score=True),
                   KNeighborsRegressor(n_neighbors=5)]
```

[4]:
```
# Define function to perform k-folds CV
# we can specify regressor to use, number of kfolds, x and y training data
def kfolds_cv_regression(regressor, kfolds, x_train, y_train):
    kfolds_regression = KFold(n_splits = kfolds, random_state = 1, shuffle =␣
 ↪True)
    r2_model_1_cv = cross_val_score(regressor, x_train, y_train,␣
 ↪cv=kfolds_regression)
    #print("r squared of ",kfolds,"-folds:",r2_model_1_cv,"(mean r squared:",np.
 ↪mean(r2_model_1_cv),")")
    neg_mse_model_1_cv = cross_val_score(regressor, x_train, y_train,␣
 ↪cv=kfolds_regression,scoring = 'neg_mean_squared_error')
    #print("mean_squared_error of ",kfolds,"-folds:",-neg_mse_model_1_cv,"(mean␣
 ↪MSE:",-np.mean(neg_mse_model_1_cv),")")
    return np.mean(r2_model_1_cv), -np.mean(neg_mse_model_1_cv)
```

2 K-Fold Cross Validation on different regression models, compared on Mean R Squared and Mean MSE

[11]:
```
results = []
for regressor in regression_models:
    for KFolds in [5,10,15]:
        model_name=type(regressor).__name__

        # Perform cross-validation and get scores
        mean_r2, mean_mse=kfolds_cv_regression(regressor, KFolds,␣
 ↪X_train_scaled, y_train)
        results.append({'Model': model_name, 'KFolds':KFolds,'Mean R Squared':␣
 ↪mean_r2, 'Mean MSE': mean_mse})

# Print the results DataFrame
df_results = pd.DataFrame(results)
```

[12]:
```
df_results.sort_values(['KFolds','Mean R Squared'])
```

[12]:

	Model	KFolds	Mean R Squared	Mean MSE
6	Lasso	5	0.924447	1182.778282
0	LinearRegression	5	0.924657	1179.476688
3	Ridge	5	0.924658	1179.470433

```
15    KNeighborsRegressor       5        0.990365   150.120516
9   DecisionTreeRegressor       5        0.992236   120.967048
12  RandomForestRegressor       5        0.996515    54.397492
7                   Lasso      10        0.924599  1183.263556
1        LinearRegression      10        0.924821  1180.050385
4                   Ridge      10        0.924821  1180.041713
16    KNeighborsRegressor      10        0.991001   140.809494
10  DecisionTreeRegressor      10        0.993242   105.148155
13  RandomForestRegressor      10        0.996776    50.426409
8                   Lasso      15        0.923961  1183.053413
2        LinearRegression      15        0.924170  1179.717693
5                   Ridge      15        0.924171  1179.713027
17    KNeighborsRegressor      15        0.991436   133.246107
11  DecisionTreeRegressor      15        0.993029   107.897222
14  RandomForestRegressor      15        0.996806    49.589119
```

Based on K-Fold CV results, Random Forest Regressor resulted best in terms of Mean R-Squared and MSE, therefore we choose Random Forest Regressor as our winning model. Thus we would like to try Hyperparameter Tuning to further improve the Random Forest regressor.

2.1 Random Forest Regressor Hyperparameter Tuning

[15]:
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(random_state=0, oob_score=True)
```

[16]:
```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(regressor.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 1.0,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'monotonic_cst': None,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': True,
 'random_state': 0,
```

```
 'verbose': 0,
 'warm_start': False}
```

2.1.1 GridSearchCV for Hyperparameter Tuning

[17]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'bootstrap': [True],
    'max_depth': [10,20, 30],
    'max_features': [2, 3, 4],
    'min_samples_leaf': [2, 3],
    'min_samples_split': [2,4,6],
    'n_estimators': [1000, 1100, 1200, 1500]
}
# Create a based model
regressor = RandomForestRegressor(random_state=0, oob_score=True)
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = regressor, param_grid = param_grid,
                           cv = 5, n_jobs = -1, verbose = 0)
```

[18]:
```
# Fit the grid search to the data
grid_search.fit(X_train_scaled, y_train)
# Get the model with best hyperparameters
grid_search.best_params_
```

[18]:
```
{'bootstrap': True,
 'max_depth': 30,
 'max_features': 3,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 1000}
```

[19]:
```
best_grid = grid_search.best_estimator_
best_grid
```

[19]:
```
RandomForestRegressor(max_depth=30, max_features=3, min_samples_leaf=2,
                      n_estimators=1000, oob_score=True, random_state=0)
```

Run k-fold CV on BASE Random Forest Regressor

[20]:
```
results = []
for regressor in [RandomForestRegressor(random_state=0, oob_score=True)]:
    for KFolds in [5,10,15]:
        model_name=type(regressor).__name__

        # Perform cross-validation and get scores
        mean_r2, mean_mse=kfolds_cv_regression(regressor, KFolds,␣
 ↪X_train_scaled, y_train)
```

```
        results.append({'Model': model_name, 'KFolds':KFolds,'Mean R Squared':␣
 ↪mean_r2, 'Mean MSE': mean_mse})

# Print the results DataFrame
df_results = pd.DataFrame(results)
df_results
```

[20]:

	Model	KFolds	Mean R Squared	Mean MSE
0	RandomForestRegressor	5	0.996515	54.397492
1	RandomForestRegressor	10	0.996776	50.426409
2	RandomForestRegressor	15	0.996806	49.589119

Run k-fold CV on best model from GridSearchCV Random Forest Regressor

[21]:
```
results = []
#for regressor in [RandomForestRegressor(max_depth=80, max_features=3,␣
 ↪min_samples_leaf=3,
 #                    min_samples_split=8, n_estimators=1000, oob_score=True,
  #                   random_state=0)]:
for regressor in [RandomForestRegressor(max_depth=30, max_features=3,␣
 ↪min_samples_leaf=2,
                     min_samples_split=2, n_estimators=1000, oob_score=True,
                     random_state=0)]:
    for KFolds in [5,10,15]:
        model_name=type(regressor).__name__

        # Perform cross-validation and get scores
        mean_r2, mean_mse=kfolds_cv_regression(regressor, KFolds,␣
 ↪X_train_scaled, y_train)
        results.append({'Model': model_name, 'KFolds':KFolds,'Mean R Squared':␣
 ↪mean_r2, 'Mean MSE': mean_mse})

# Print the results DataFrame
df_results = pd.DataFrame(results)
df_results
```

[21]:

	Model	KFolds	Mean R Squared	Mean MSE
0	RandomForestRegressor	5	0.996317	57.472959
1	RandomForestRegressor	10	0.996614	52.862028
2	RandomForestRegressor	15	0.996610	52.606638

After Hyperparameter Tuning, our Mean R-Squared and MSE did not improve, therefore we chose to stick to our base RandomForest regressor.

2.2 Feature Importance on Random Forest Model

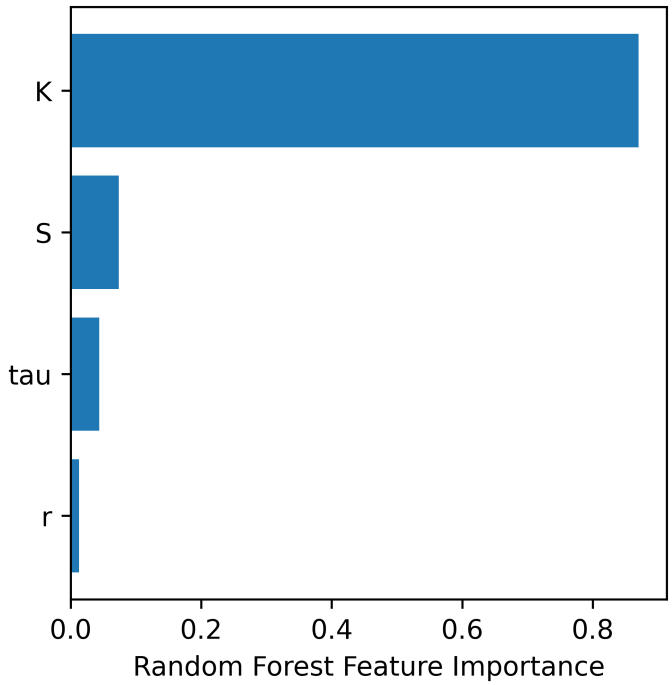
After choosing the best Model, we also explored which features are more important

[22]:
```
regressor = RandomForestRegressor(random_state=0, oob_score=True)
regressor.fit(X_train_scaled, y_train)
global_importances_random = pd.Series(regressor.feature_importances_,␣
 ↪index=list(X_train.columns))
```

[23]:
```
regressor.feature_importances_
```

[23]:
```
array([0.07348851, 0.87014665, 0.0438318 , 0.01253305])
```

[24]:
```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=600)
sorted_idx = regressor.feature_importances_.argsort()[-10:] ## remove [-10:]␣
 ↪to␣get a plot for all features
plt.barh(X_train.columns[sorted_idx], regressor.
 ↪feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
plt.savefig('plot.png')
```



Linear_Regression

May 2, 2024

1 Linear Regression & Best Subset Selection

DSO 530 Spring 2024 Final Project
Group 49: Jessica Bratahani, Pin Hsuan Chang, Suhan Ho, Sheena Huang, Yunchi Lee

[1]:
```
import numpy as np
import pandas as pd
import time
import itertools
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
```

1.1 Training Data

[2]:
```
df_train=pd.read_csv('option_train.csv',index_col=0)
```

[3]:
```
df_train.head()
```

[3]:

	Value	S	K	tau	r	BS
1	348.500	1394.46	1050	0.128767	0.0116	Under
2	149.375	1432.25	1400	0.679452	0.0113	Under
3	294.500	1478.90	1225	0.443836	0.0112	Under
4	3.375	1369.89	1500	0.117808	0.0119	Over
5	84.000	1366.42	1350	0.298630	0.0119	Under

[4]:
```
#checking for any null values
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 5000 entries, 1 to 5000
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Value   5000 non-null   float64
 1   S       5000 non-null   float64
 2   K       5000 non-null   int64
```

```
 3   tau     5000 non-null   float64
 4   r       5000 non-null   float64
 5   BS      5000 non-null   object
dtypes: float64(4), int64(1), object(1)
memory usage: 273.4+ KB
```

[5]:
```
#Correlation matrix
matrix = df_train.corr(numeric_only=True)
print("Correlation matrix: ")
print(matrix)
```

```
Correlation matrix:
          Value         S         K       tau         r
Value  1.000000  0.148884 -0.880802  0.255343 -0.163317
S      0.148884  1.000000  0.128228 -0.020299 -0.983740
K     -0.880802  0.128228  1.000000  0.022948 -0.111107
tau    0.255343 -0.020299  0.022948  1.000000  0.010245
r     -0.163317 -0.983740 -0.111107  0.010245  1.000000
```

1.2 Linear Regression

[6]:
```
mlr = LinearRegression()
X_train = df_train[['S','K','tau','r']].values
y_train = df_train['Value'].values

mlr.fit(X_train,y_train)
r_sq_house=mlr.score(X_train,y_train)
print('In Sample R2:',r_sq_house)
#r_sq_house=lm.score(X_test,y_test)
#print('Out of Sample R2:',r_sq_house)
y_train_pred = mlr.predict(X_train)
MSE = np.square(np.subtract(y_train,y_train_pred)).mean()
print('MSE: ',MSE)
```

```
In Sample R2: 0.9249785018883101
MSE:  1174.8847708553976
```

[7]:
```
mlr = LinearRegression()
X_train = df_train[['S','K','tau']].values
y_train = df_train['Value'].values

mlr.fit(X_train,y_train)
r_sq_house=mlr.score(X_train,y_train)
print('In Sample R2:',r_sq_house)

y_train_pred = mlr.predict(X_train)
MSE = np.square(np.subtract(y_train,y_train_pred)).mean()
print('MSE: ',MSE)
```

```
In Sample R2: 0.9249766211573232
MSE:  1174.9142243087429
```

1.3 Best Subset Selection for Linear Regression

Best Subset Selection is still feasible in our case, because we only have 4 parameters to predict y(Value), therefore our group chose to proceed with Best Subset Selection compared to Forward or Backward Stepwise selection.

[8]:
```
def processSubset(feature_set,y):
    # Fit model on feature_set and calculate RSS
    X1 = sm.add_constant(X[list(feature_set)])
    model = sm.OLS(y,X1)
    regr = model.fit()
    RSS = ((regr.predict(X1) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}
```

[9]:
```
def getBest(k):
    tic = time.time()
    results = []
    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo,y_train))
    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    # Choose the model with the smallest RSS
    best_model = models.loc[models['RSS'].idxmin()]
    # idxmin() function returns index of first occurrence of minimum.
    toc = time.time()
    print("Processed ", models.shape[0], "models on", k, "predictors␣
 ↪in",(toc-tic), "seconds.")
    # Return the best model, along with some other useful information about the␣
 ↪model
    return best_model
```

[10]:
```
X= df_train[['S','K','tau','r']]
models = pd.DataFrame(columns=["RSS", "model"])
tic = time.time()
for i in range(0,5):
    models.loc[i] = getBest(i)
toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

```
Processed  1 models on 0 predictors in 0.0030939579010009766 seconds.
Processed  4 models on 1 predictors in 0.011147022247314453 seconds.
Processed  6 models on 2 predictors in 0.010612964630126953 seconds.
Processed  4 models on 3 predictors in 0.004215717315673828 seconds.
Processed  1 models on 4 predictors in 0.0011038780212402344 seconds.
```

```
Total elapsed time: 0.032962083816528320 seconds.
```

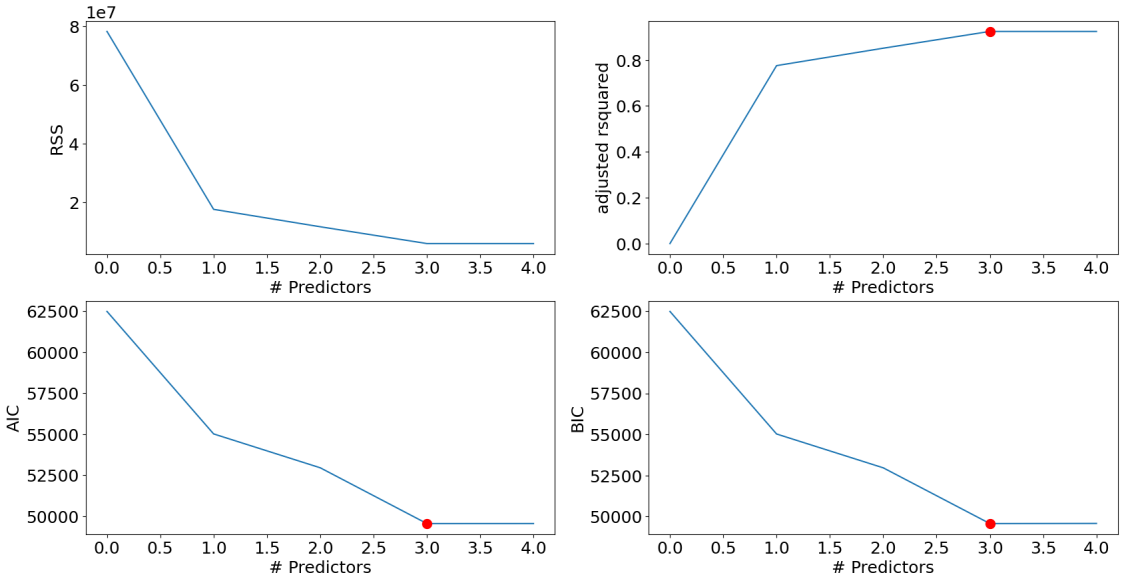
[11]:
```
models.sort_values('RSS')
```

[11]:

	RSS	model
4	5874423.854277	<statsmodels.regression.linear_model.Regressio…
3	5874571.121544	<statsmodels.regression.linear_model.Regressio…
2	11605816.160791	<statsmodels.regression.linear_model.Regressio…
1	17554565.66617	<statsmodels.regression.linear_model.Regressio…
0	78303206.442656	<statsmodels.regression.linear_model.Regressio…

[12]:
```
plt.figure(figsize=(20,10))
plt.rcParams.update({'font.size': 18, 'lines.markersize': 10})

# Set up a 2x2 grid so we can look at 4 plots at once
plt.subplot(2, 2, 1)
# We will now plot a curve to show the relationship between the number of␣
 ↪predictors and the RSS
plt.plot(models["RSS"])
plt.xlabel('# Predictors')
plt.ylabel('RSS')
# We will now plot a red dot to indicate the model with the largest adjusted␣
 ↪R^2 statistic.
# The idxmax() function can be used to identify the location of the maximum␣
 ↪point of a vector
rsquared_adj = models.apply(lambda row: row[1].rsquared_adj, axis=1)
plt.subplot(2, 2, 2)
plt.plot(rsquared_adj)
plt.plot(rsquared_adj.idxmax(), rsquared_adj.max(), "or")
plt.xlabel('# Predictors')
plt.ylabel('adjusted rsquared')
# We'll do the same for AIC and BIC, this time looking for the models with the␣
 ↪SMALLEST statistic
aic = models.apply(lambda row: row[1].aic, axis=1)
plt.subplot(2, 2, 3)
plt.plot(aic)
plt.plot(aic.idxmin(), aic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('AIC')
bic = models.apply(lambda row: row[1].bic, axis=1)
plt.subplot(2, 2, 4)
plt.plot(bic)
plt.plot(bic.idxmin(), bic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('BIC')
plt.savefig('BSS_LinearReg.png')
```



1.3.1 Best OLS Regression Results with 3 predictors

[13]:
```
print(getBest(3)["model"].summary())
```

```
Processed  4 models on 3 predictors in 0.023597002029418945 seconds.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.925
Model:                            OLS   Adj. R-squared:                  0.925
Method:                 Least Squares   F-statistic:                 2.053e+04
Date:                Thu, 02 May 2024   Prob (F-statistic):               0.00
Time:                        14:36:35   Log-Likelihood:                -24767.
No. Observations:                5000   AIC:                         4.954e+04
Df Residuals:                    4996   BIC:                         4.957e+04
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        137.0005     12.594     10.878      0.000     112.311     161.690
S              0.6093      0.009     69.815      0.000       0.592       0.626
K             -0.6684      0.003   -235.952      0.000      -0.674      -0.663
tau          152.6895      2.099     72.744      0.000     148.575     156.804
==============================================================================
Omnibus:                     3141.362   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            45094.076
Skew:                           2.788   Prob(JB):                         0.00
Kurtosis:                      16.615   Cond. No.                     5.15e+04
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.15e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1.3.2 Best OLS Regression Results with 4 predictors

[14]:
```
print(getBest(4)["model"].summary())
```

```
Processed  1 models on 4 predictors in 0.0049169063568115234 seconds.
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.925
Model:                            OLS   Adj. R-squared:                  0.925
Method:                 Least Squares   F-statistic:                 1.540e+04
Date:                Thu, 02 May 2024   Prob (F-statistic):               0.00
Time:                        14:36:35   Log-Likelihood:                -24767.
No. Observations:                5000   AIC:                         4.954e+04
Df Residuals:                    4995   BIC:                         4.958e+04
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        185.5711    137.834      1.346      0.178     -84.644     455.786
S              0.5924      0.049     12.203      0.000       0.497       0.688
K             -0.6684      0.003   -235.028      0.000      -0.674      -0.663
tau          152.6475      2.103     72.601      0.000     148.526     156.769
r          -2142.4445   6054.403     -0.354      0.723    -1.4e+04    9726.844
==============================================================================
Omnibus:                     3143.142   Durbin-Watson:                   2.002
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            45196.502
Skew:                           2.789   Prob(JB):                         0.00
Kurtosis:                      16.632   Cond. No.                     2.48e+07
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.48e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

1.3.3 Verifying using K-Folds CV

[15]:
```
from sklearn.model_selection import KFold ## for regression
from sklearn.model_selection import cross_val_score

kfolds_regresssion = KFold(n_splits = 5, random_state = 1, shuffle = True)

regression_model = LinearRegression()
#use r2 (the default) as the criterion for CV score, for other option add:␣
 ↪scoring = neg_mean_squared_error
r2_model_1_cv = cross_val_score(regression_model, df_train[['S', 'K', 'tau',␣
 ↪'r']], df_train['Value'], cv=kfolds_regresssion)
r2_model_2_cv = cross_val_score(regression_model, df_train[['S', 'K', 'tau']],␣
 ↪df_train['Value'], cv=kfolds_regresssion)

print("Linear Regression:")
print("Model 1 (4 features): r squared of 5-folds:",r2_model_1_cv,"(mean r␣
 ↪squared:",np.mean(r2_model_1_cv),")")
print("Model 2 (3 features): r squared of 5-folds:",r2_model_2_cv,"(mean r␣
 ↪squared:",np.mean(r2_model_2_cv),")")

#mse_model_1_cv = cross_val_score(regression_model, df_train[['S', 'K', 'tau',␣
 ↪'r']], df_train['Value'], cv=kfolds_regresssion, scoring =␣
 ↪neg_mean_squared_error)
#mse_model_2_cv = cross_val_score(regression_model, df_train[['S', 'K',␣
 ↪'tau']], df_train['Value'], cv=kfolds_regresssion, scoring =␣
 ↪neg_mean_squared_error)
#print("Linear Regression:")
#print("Model 1: mse of 5-folds:",r2_model_1_cv,"(mean mse:",np.
 ↪mean(mse_model_1_cv),")")
#print("Model 2: mse of 5-folds:",r2_model_2_cv,"(mean mse:",np.
 ↪mean(mse_model_2_cv),")")
```

```
Linear Regression:
Model 1 (4 features): r squared of 5-folds: [0.93093568 0.92857914 0.93000769
0.91180558 0.92195926] (mean r squared: 0.9246574701346504 )
Model 2 (3 features): r squared of 5-folds: [0.93094208 0.92858512 0.93001469
0.91183742 0.92198531] (mean r squared: 0.924672923615202 )
```

When looking closely at the OLS regression results from k-fold cross validation with 3 vs. 4 features, the difference in mean R-squared is less than 0.00001, therefore we decided to do regression on all 4 variables

[ ]:
```
```

Models without scaled

In [10]:
```
import numpy as np
import pandas as pd

import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

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

In [2]:
```
data=pd.read_csv('option_train.csv')
data.head()
```

Out[2]:

	Unnamed: 0	Value	S	K	tau	r	BS
0	1	348.500	1394.46	1050	0.128767	0.0116	Under
1	2	149.375	1432.25	1400	0.679452	0.0113	Under
2	3	294.500	1478.90	1225	0.443836	0.0112	Under
3	4	3.375	1369.89	1500	0.117808	0.0119	Over
4	5	84.000	1366.42	1350	0.298630	0.0119	Under

In [3]:
```
data = data.drop('Unnamed: 0', axis=1)
data.head()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Value   5000 non-null   float64
 1   S       5000 non-null   float64
 2   K       5000 non-null   int64  
 3   tau     5000 non-null   float64
 4   r       5000 non-null   float64
 5   BS      5000 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 234.5+ KB
```

In [4]:
```
df_train = data.drop('Value', axis=1)
df_train['BS'] = df_train['BS'].map({'Under': 0, 'Over': 1})

X_train = df_train.iloc[:,:4]
y_train = df_train['BS']

print(X_train)
```

```
            S     K       tau       r
0     1394.46  1050  0.128767  0.0116
1     1432.25  1400  0.679452  0.0113
2     1478.90  1225  0.443836  0.0112
3     1369.89  1500  0.117808  0.0119
4     1366.42  1350  0.298630  0.0119
...       ...   ...       ...     ...
4995  1465.15  1175  0.424658  0.0111
4996  1480.87  1480  0.101370  0.0111
4997  1356.56  1500  0.673973  0.0120
4998  1333.36  1200  0.309589  0.0122
4999  1480.87  1475  0.504110  0.0111

[5000 rows x 4 columns]
```

In [ ]:
```

```

Logistic Regression CV5

In [11]:
```
logistic_reg = LogisticRegression()
scores = cross_val_score(logistic_reg, X_train,  y_train, cv=5)
print("Logistic Regression: ")
print("accuracies of 5-folds: ",np.mean(scores))
```

```
Logistic Regression:
accuracies of 5-folds:  0.8892
```

Logistic Regression CV10

In [12]:
```
logistic_reg = LogisticRegression()
scores = cross_val_score(logistic_reg, X_train,  y_train, cv=10)
print("Logistic Regression: ")
print("accuracies of 10-folds: ",np.mean(scores))
```

```
Logistic Regression:
accuracies of 10-folds:  0.8896000000000001
```

In [ ]:
```

```

K Neighbors Classifier CV5

In [13]:
```
classifier = KNeighborsClassifier(n_neighbors=5)
knn = classifier.fit(X_train, y_train)

scores = cross_val_score(knn, X_train,  y_train, cv=5)
print("KNN: ")
print("accuracies of 5-folds: ", np.mean(scores))
```

```
KNN:
accuracies of 5-folds:  0.8552
```

K Neighbors Classifier CV10

In [15]:
```
classifier = KNeighborsClassifier(n_neighbors=5)
knn = classifier.fit(X_train, y_train)

scores = cross_val_score(knn, X_train,  y_train, cv=10)
print("KNN: ")
print("accuracies of 10-folds: ", np.mean(scores))
```

```
KNN:
accuracies of 10-folds:  0.8593999999999999
```

In [ ]:
```

```

Decision Trees CV5

In [16]:
```
clf_tree = DecisionTreeClassifier(random_state=0)

# Note that although the tree building process looks like a deterministic process, inside the package,
# there is some heuristic iterative algorithm used, so setting a random_state will make sure of reproducibility

path = clf_tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
```

In [17]:
```
from sklearn.model_selection import StratifiedKFold
accuracies = []
kfolds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 1)

for ccp_alpha in ccp_alphas:
    score_for_alpha = []
    for train_index, test_index in kfolds.split(X_train, y_train):
        clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
        # Use .iloc[] for integer-location based indexing
        clf.fit(X_train.iloc[train_index], y_train.iloc[train_index])
        y_pred = clf.predict(X_train.iloc[test_index])
        score = accuracy_score(y_pred, y_train.iloc[test_index])
        score_for_alpha.append(score)
    accuracies.append(sum(score_for_alpha) / len(score_for_alpha))
```

In [20]:
```
print("\nThe index corresponding to the maximum of the accuracies: ",np.argmax(accuracies))
```

```
The index corresponding to the maximum of the accuracies:  30
```

In [21]:
```
# Use the selected alpha to retrain a tree on the entire (X_train,y_train)
alpha_cv = ccp_alphas[np.argmax(accuracies)]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
#clf_tree_final.fit(X_train, y_train)
```

In [22]:
```
# a streamlined way to evaluate the final model's performance using cross-validation <- need to change some deta

from sklearn.model_selection import cross_val_score
import numpy as np

# Assuming X_train, y_train are defined, and clf_tree_final is trained with the best ccp_alpha

# Now perform K-Fold cross-validation on the final model
n_splits = 5
final_scores = cross_val_score(clf_tree_final, X_train, y_train, cv=n_splits, scoring='accuracy')

# Calculate the mean and standard deviation of the cross-validated scores
mean_final_accuracy = np.mean(final_scores)
std_final_accuracy = np.std(final_scores)

print(f"Final model mean accuracy: {mean_final_accuracy:.4f}")
print(f"Final model standard deviation of accuracy: {std_final_accuracy:.4f}")
```

```
Final model mean accuracy: 0.9124
Final model standard deviation of accuracy: 0.0075
```

Decision Trees CV10

In [23]:
```
from sklearn.model_selection import StratifiedKFold
accuracies = []
kfolds = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 1)

for ccp_alpha in ccp_alphas:
    score_for_alpha = []
    for train_index, test_index in kfolds.split(X_train, y_train):
        clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
        # Use .iloc[] for integer-location based indexing
        clf.fit(X_train.iloc[train_index], y_train.iloc[train_index])
        y_pred = clf.predict(X_train.iloc[test_index])
        score = accuracy_score(y_pred, y_train.iloc[test_index])
        score_for_alpha.append(score)
    accuracies.append(sum(score_for_alpha) / len(score_for_alpha))
```

In [24]:
```
print("\nThe index corresponding to the maximum of the accuracies: ",np.argmax(accuracies))
```

```
The index corresponding to the maximum of the accuracies:  19
```

In [25]:
```
alpha_cv = ccp_alphas[np.argmax(accuracies)]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train, y_train)
```

Out[25]:

DecisionTreeClassifier

```
DecisionTreeClassifier(ccp_alpha=0.0001459696969696972, random_state=0)
```

In [26]:
```
# a streamlined way to evaluate the final model's performance using cross-validation <- need to change some deta

from sklearn.model_selection import cross_val_score
import numpy as np

# Assuming X_train, y_train are defined, and clf_tree_final is trained with the best ccp_alpha

# Now perform K-Fold cross-validation on the final model
n_splits = 5
final_scores = cross_val_score(clf_tree_final, X_train, y_train, cv=n_splits, scoring='accuracy')

# Calculate the mean and standard deviation of the cross-validated scores
mean_final_accuracy = np.mean(final_scores)
std_final_accuracy = np.std(final_scores)

print(f"Final model mean accuracy: {mean_final_accuracy:.4f}")
print(f"Final model standard deviation of accuracy: {std_final_accuracy:.4f}")
```

```
Final model mean accuracy: 0.9128
Final model standard deviation of accuracy: 0.0074
```

In [ ]:

Random Forest CV5

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [28]:
```
clf_rf = RandomForestClassifier(random_state=1, n_estimators = 200)
clf_rf.fit(X_train, y_train)
```

Out[28]:

```
RandomForestClassifier(n_estimators=200, random_state=1)
```

In [29]:

```
# Initialize the StratifiedKFold object
kfolds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
accuracies_rd = []

# Cross-validation loop
for train_index, test_index in kfolds.split(X_train, y_train):
    clf_rf = RandomForestClassifier(random_state=1, n_estimators=200)
    clf_rf.fit(X_train.iloc[train_index], y_train.iloc[train_index])  # Train on the fold's training part
    y_pred_rf = clf_rf.predict(X_train.iloc[test_index])  # Predict on the fold's testing part
    score = accuracy_score(y_train.iloc[test_index], y_pred_rf)  # Calculate accuracy
    accuracies_rd.append(score)

# Calculate mean and standard deviation of accuracies
mean_accuracy_rf = sum(accuracies_rd) / len(accuracies_rd)
std_accuracy_rf = np.std(accuracies_rd)

print(f"Random Forest mean accuracy: {mean_accuracy_rf:.4f}")
print(f"Random Forest standard deviation of accuracy: {std_accuracy_rf:.4f}")
```

```
Random Forest mean accuracy: 0.9334
Random Forest standard deviation of accuracy: 0.0027
```

Random Forest CV10

In [30]:

```
clf_rf = RandomForestClassifier(random_state=1, n_estimators = 200)
clf_rf.fit(X_train, y_train)
```

Out[30]:

```
RandomForestClassifier(n_estimators=200, random_state=1)
```

In [31]:

```
# Initialize the StratifiedKFold object
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
accuracies_rd = []

# Cross-validation loop
for train_index, test_index in kfolds.split(X_train, y_train):
    clf_rf = RandomForestClassifier(random_state=1, n_estimators=200)
    clf_rf.fit(X_train.iloc[train_index], y_train.iloc[train_index])  # Train on the fold's training part
    y_pred_rf = clf_rf.predict(X_train.iloc[test_index])  # Predict on the fold's testing part
    score = accuracy_score(y_train.iloc[test_index], y_pred_rf)  # Calculate accuracy
    accuracies_rd.append(score)

# Calculate mean and standard deviation of accuracies
mean_accuracy_rf = sum(accuracies_rd) / len(accuracies_rd)
std_accuracy_rf = np.std(accuracies_rd)

print(f"Random Forest mean accuracy: {mean_accuracy_rf:.4f}")
print(f"Random Forest standard deviation of accuracy: {std_accuracy_rf:.4f}")
```

```
Random Forest mean accuracy: 0.9366
Random Forest standard deviation of accuracy: 0.0096
```

In [32]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
params_grid = {
    'n_estimators': [25, 50, 100, 150,200],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [3, 6, 9],
    'max_leaf_nodes': [3, 6, 9],
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=1), param_grid=params_grid, cv=5,verbose=0)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)
```

```
RandomForestClassifier(max_depth=6, max_leaf_nodes=9, n_estimators=200,
                       random_state=1)
```

In [33]:

```
grid_search.best_estimator_
```

Out[33]:

```
RandomForestClassifier(max_depth=6, max_leaf_nodes=9, n_estimators=200,
                       random_state=1)
```

In [34]:
```
# Initialize the StratifiedKFold object
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
accuracies_rd = []

# Cross-validation loop
for train_index, test_index in kfolds.split(X_train, y_train):
    clf_rf = RandomForestClassifier(max_depth=6, max_leaf_nodes=9, n_estimators=200,
                      random_state=1)
    clf_rf.fit(X_train.iloc[train_index], y_train.iloc[train_index])  # Train on the fold's training part
    y_pred_rf = clf_rf.predict(X_train.iloc[test_index])  # Predict on the fold's testing part
    score = accuracy_score(y_train.iloc[test_index], y_pred_rf)  # Calculate accuracy
    accuracies_rd.append(score)

# Calculate mean and standard deviation of accuracies
mean_accuracy_rf = sum(accuracies_rd) / len(accuracies_rd)
std_accuracy_rf = np.std(accuracies_rd)

print(f"Random Forest mean accuracy: {mean_accuracy_rf:.4f}")
print(f"Random Forest standard deviation of accuracy: {std_accuracy_rf:.4f}")
```

```
Random Forest mean accuracy: 0.8938
Random Forest standard deviation of accuracy: 0.0105
```

In [ ]:
```

```

Gradient Boosting CV5

In [35]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

accuracies_gb = []
kfolds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

for train_index, test_index in kfolds.split(X_train, y_train):
  clf_gb = GradientBoostingClassifier(n_estimators=200, random_state=1)
  clf_gb.fit(X_train.iloc[train_index], y_train.iloc[train_index])  # Train on the fold's training part
  y_pred_gb = clf_gb.predict(X_train.iloc[test_index])  # Predict on the fold's testing part
  score = accuracy_score(y_train.iloc[test_index], y_pred_gb)  # Calculate accuracy
  accuracies_gb.append(score)
```

In [36]:
```
mean_accuracy_gb = sum(accuracies_gb) / len(accuracies_gb)
std_accuracy_gb = np.std(accuracies_gb)

print(f"Gradient Boosting mean accuracy: {mean_accuracy_gb:.4f}")
print(f"Gradient Boosting standard deviation of accuracy: {std_accuracy_gb:.4f}")
```

```
Gradient Boosting mean accuracy: 0.9264
Gradient Boosting standard deviation of accuracy: 0.0083
```

Gradient Boosting CV10

In [37]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

accuracies_gb = []
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)

for train_index, test_index in kfolds.split(X_train, y_train):
  clf_gb = GradientBoostingClassifier(n_estimators=200, random_state=1)
  clf_gb.fit(X_train.iloc[train_index], y_train.iloc[train_index])  # Train on the fold's training part
  y_pred_gb = clf_gb.predict(X_train.iloc[test_index])  # Predict on the fold's testing part
  score = accuracy_score(y_train.iloc[test_index], y_pred_gb)  # Calculate accuracy
  accuracies_gb.append(score)

mean_accuracy_gb = sum(accuracies_gb) / len(accuracies_gb)
std_accuracy_gb = np.std(accuracies_gb)

print(f"Gradient Boosting mean accuracy: {mean_accuracy_gb:.4f}")
print(f"Gradient Boosting standard deviation of accuracy: {std_accuracy_gb:.4f}")
```

```
Gradient Boosting mean accuracy: 0.9268
Gradient Boosting standard deviation of accuracy: 0.0092
```

In [ ]:
```

```

Support Vector Classifier CV5

In [38]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

```
svm_classifier = SVC()
param_grid = {'C': [0.1, 1, 10, 50, 80, 100],
              'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
```

```
Best hyperparameters: {'C': 0.1, 'kernel': 'linear'}
```

In [39]:
```
svm_classifier = SVC(kernel='linear', C=0.1)
cv_scores = cross_val_score(svm_classifier, X_train, y_train, cv=5)
print("Mean accuracy:", cv_scores.mean())
```

```
Mean accuracy: 0.8855999999999999
```

Support Vector Classifier CV10

In [ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
svm_classifier = SVC()
param_grid = {'C': [0.1, 1],
              'kernel': ['linear', 'rbf']} #'poly', 'sigmoid']}
grid_search = GridSearchCV(svm_classifier, param_grid, cv=10, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
```

In [ ]:
```

```

Models with Scaled Features on CV10

In [21]:
```
import numpy as np
import pandas as pd

import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
```

In [22]:
```
data=pd.read_csv('option_train.csv')
data.head()
```

Out[22]:

	Unnamed: 0	Value	S	K	tau	r	BS
0	1	348.500	1394.46	1050	0.128767	0.0116	Under
1	2	149.375	1432.25	1400	0.679452	0.0113	Under
2	3	294.500	1478.90	1225	0.443836	0.0112	Under
3	4	3.375	1369.89	1500	0.117808	0.0119	Over
4	5	84.000	1366.42	1350	0.298630	0.0119	Under

In [23]:
```
data = data.drop('Unnamed: 0', axis=1)
data.head()
```

Out[23]:

	Value	S	K	tau	r	BS
0	348.500	1394.46	1050	0.128767	0.0116	Under
1	149.375	1432.25	1400	0.679452	0.0113	Under
2	294.500	1478.90	1225	0.443836	0.0112	Under
3	3.375	1369.89	1500	0.117808	0.0119	Over
4	84.000	1366.42	1350	0.298630	0.0119	Under

In [24]:
```
df_train = data.drop('Value', axis=1)
df_train['BS'] = df_train['BS'].map({'Under': 0, 'Over': 1})

X_train = df_train.iloc[:,:4]
y_train = df_train['BS']

print(X_train)
```

```
            S     K       tau       r
0     1394.46  1050  0.128767  0.0116
1     1432.25  1400  0.679452  0.0113
2     1478.90  1225  0.443836  0.0112
3     1369.89  1500  0.117808  0.0119
4     1366.42  1350  0.298630  0.0119
...       ...   ...       ...     ...
4995  1465.15  1175  0.424658  0.0111
4996  1480.87  1480  0.101370  0.0111
4997  1356.56  1500  0.673973  0.0120
4998  1333.36  1200  0.309589  0.0122
4999  1480.87  1475  0.504110  0.0111

[5000 rows x 4 columns]
```

Standardized the Features

In [25]:
```
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
print(X_train)
```

```
[[-0.57424194 -1.85474684 -0.86021181  0.29531344]
 [ 0.10002663  0.17233687  1.52204229 -0.37382578]
 [ 0.93237985 -0.84120499  0.50276939 -0.59687218]
 ...
 [-1.25047318  0.75150364  1.49833827  1.18749907]
 [-1.66441948 -0.98599668 -0.07797912  1.63359188]
 [ 0.9675296   0.60671195  0.76351362 -0.81991859]]
```

In [ ]:

Decision Trees

In [26]:
```
clf_tree = DecisionTreeClassifier(random_state=0)

# Note that although the tree building process looks like a deterministic␣ process, inside the package,
# there is some heuristic iterative algorithm used, so setting a random_state will make sure of reproducibility

path = clf_tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
```

In [27]:
```
from sklearn.model_selection import StratifiedKFold
accuracies = []
kfolds = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 1)

for ccp_alpha in ccp_alphas:
    score_for_alpha = []
    for train_index, test_index in kfolds.split(X_train, y_train):
        clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
        # Use .iloc[] for integer-location based indexing
        clf.fit(X_train[train_index], y_train[train_index])
        y_pred = clf.predict(X_train[test_index])
        score = accuracy_score(y_pred, y_train[test_index])
        score_for_alpha.append(score)
    accuracies.append(sum(score_for_alpha) / len(score_for_alpha))
```

In [28]:
```
print("\nThe index corresponding to the maximum of the accuracies: ",np.argmax(accuracies))
```

```
The index corresponding to the maximum of the accuracies:  19
```

In [29]:
```
alpha_cv = ccp_alphas[np.argmax(accuracies)]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train, y_train)
```

Out[29]:

DecisionTreeClassifier

```
DecisionTreeClassifier(ccp_alpha=0.0001459696969696972, random_state=0)
```

In [30]:
```
# a streamlined way to evaluate the final model's performance using cross-validation <- need to change some deta

from sklearn.model_selection import cross_val_score
import numpy as np

# Assuming X_train, y_train are defined, and clf_tree_final is trained with the best ccp_alpha

# Now perform K-Fold cross-validation on the final model
n_splits = 10
final_scores = cross_val_score(clf_tree_final, X_train, y_train, cv=n_splits, scoring='accuracy')

# Calculate the mean and standard deviation of the cross-validated scores
mean_final_accuracy = np.mean(final_scores)
std_final_accuracy = np.std(final_scores)

print(f"Final model mean accuracy: {mean_final_accuracy:.4f}")
print(f"Final model standard deviation of accuracy: {std_final_accuracy:.4f}")
```

```
Final model mean accuracy: 0.9184
Final model standard deviation of accuracy: 0.0061
```

In [ ]:

Random Forest

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [32]:
```
clf_rf = RandomForestClassifier(random_state=1, n_estimators = 200)
clf_rf.fit(X_train, y_train)
```

Out[32]:

```
RandomForestClassifier(n_estimators=200, random_state=1)
```

In [33]:

```
# Initialize the StratifiedKFold object
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)
accuracies_rd = []

# Cross-validation loop
for train_index, test_index in kfolds.split(X_train, y_train):
    clf_rf = RandomForestClassifier(random_state=1, n_estimators=200)
    clf_rf.fit(X_train[train_index], y_train[train_index])  # Train on the fold's training part
    y_pred_rf = clf_rf.predict(X_train[test_index])  # Predict on the fold's testing part
    score = accuracy_score(y_train[test_index], y_pred_rf)  # Calculate accuracy
    accuracies_rd.append(score)

# Calculate mean and standard deviation of accuracies
mean_accuracy_rf = sum(accuracies_rd) / len(accuracies_rd)
std_accuracy_rf = np.std(accuracies_rd)

print(f"Random Forest mean accuracy: {mean_accuracy_rf:.4f}")
print(f"Random Forest standard deviation of accuracy: {std_accuracy_rf:.4f}")
```

```
Random Forest mean accuracy: 0.9368
Random Forest standard deviation of accuracy: 0.0096
```

In [ ]:

Gradient Boosting

In [34]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

accuracies_gb = []
kfolds = StratifiedKFold(n_splits=10, shuffle=True, random_state=1)

for train_index, test_index in kfolds.split(X_train, y_train):
  clf_gb = GradientBoostingClassifier(n_estimators=200, random_state=1)
  clf_gb.fit(X_train[train_index], y_train[train_index])  # Train on the fold's training part
  y_pred_gb = clf_gb.predict(X_train[test_index])  # Predict on the fold's testing part
  score = accuracy_score(y_train[test_index], y_pred_gb)  # Calculate accuracy
  accuracies_gb.append(score)
```

In [35]:

```
mean_accuracy_gb = sum(accuracies_gb) / len(accuracies_gb)
std_accuracy_gb = np.std(accuracies_gb)

print(f"Gradient Boosting mean accuracy: {mean_accuracy_gb:.4f}")
print(f"Gradient Boosting standard deviation of accuracy: {std_accuracy_gb:.4f}")
```

```
Gradient Boosting mean accuracy: 0.9266
Gradient Boosting standard deviation of accuracy: 0.0095
```

In [ ]:

Support Vector Classifier

In [36]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
svm_classifier = SVC()
param_grid = {'C': [0.1, 1, 10, 50, 80, 100],
              'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}
grid_search = GridSearchCV(svm_classifier, param_grid, cv=10, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
```

```
Best hyperparameters: {'C': 80, 'kernel': 'rbf'}
```

In [38]:

```
svm_classifier = SVC(kernel='rbf', C=80)
cv_scores = cross_val_score(svm_classifier, X_train, y_train, cv=10)
print("Mean accuracy:", cv_scores.mean())
```

```
Mean accuracy: 0.9192
```

In [ ]:

Train_Test_split on CV10

In [1]:
```
import numpy as np
import pandas as pd

import sklearn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
```

In [2]:
```
data=pd.read_csv('option_train.csv')
data.head()
```

Out[2]:

	Unnamed: 0	Value	S	K	tau	r	BS
0	1	348.500	1394.46	1050	0.128767	0.0116	Under
1	2	149.375	1432.25	1400	0.679452	0.0113	Under
2	3	294.500	1478.90	1225	0.443836	0.0112	Under
3	4	3.375	1369.89	1500	0.117808	0.0119	Over
4	5	84.000	1366.42	1350	0.298630	0.0119	Under

In [3]:
```
data = data.drop('Unnamed: 0', axis=1)
data.head()
```

Out[3]:

	Value	S	K	tau	r	BS
0	348.500	1394.46	1050	0.128767	0.0116	Under
1	149.375	1432.25	1400	0.679452	0.0113	Under
2	294.500	1478.90	1225	0.443836	0.0112	Under
3	3.375	1369.89	1500	0.117808	0.0119	Over
4	84.000	1366.42	1350	0.298630	0.0119	Under

In [4]:
```
df_train = data.drop('Value', axis=1)
df_train['BS'] = df_train['BS'].map({'Under': 0, 'Over': 1})

X = df_train.iloc[:,:4]
y = df_train['BS']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=3, stratify=y)
```

In [ ]:
```

```

Decision Tree

In [6]:
```
clf_tree = DecisionTreeClassifier(random_state=0)

path = clf_tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
```

In [7]:
```
from sklearn.model_selection import StratifiedKFold
kfolds = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 1)

accuracies = []
for ccp_alpha in ccp_alphas:
  score_for_alpha = []
  for train_index, test_index in kfolds.split(X_train, y_train):
      clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
      clf.fit(X_train.iloc[train_index], y_train.iloc[train_index])
      y_pred = clf.predict(X_train.iloc[test_index])
      score = accuracy_score(y_pred, y_train.iloc[test_index])
      score_for_alpha.append(score)
  accuracies.append(sum(score_for_alpha)/len(score_for_alpha))
```

In [8]:
```
print("\nThe index corresponding to the maximum of the accuracies: ",np.argmax(accuracies))
```

```
The index corresponding to the maximum of the accuracies:  53
```

In [9]:
```
alpha_cv = ccp_alphas[np.argmax(accuracies)]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train, y_train)
# Evaluate on the (X_test,y_test)
y_pred_test = clf_tree_final.predict(X_test)
score_test = accuracy_score(y_test, y_pred_test)
print(score_test)
```

```
0.9104
```

In [ ]:

Random Forest

In [10]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [11]:
```
clf_rf = RandomForestClassifier(random_state=1, n_estimators = 200)
clf_rf.fit(X_train, y_train)
y_pred_rf = clf_rf.predict(X_test)
score_test_rf = accuracy_score(y_test, y_pred_rf)
print(score_test_rf)
```

```
0.9232
```

In [ ]:

Gradient Boosting

In [12]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

clf_gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=1)
clf_gb.fit(X_train, y_train)

y_pred_gb = clf_gb.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Accuracy:", accuracy_gb)
```

```
Accuracy: 0.9104
```

In [ ]:

Support Vector Classifier

In [ ]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
svm_classifier = SVC()
param_grid = {'C': [0.1, 1, 10, 50, 80, 100],
              'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}
grid_search = GridSearchCV(svm_classifier, param_grid, cv=10, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
```