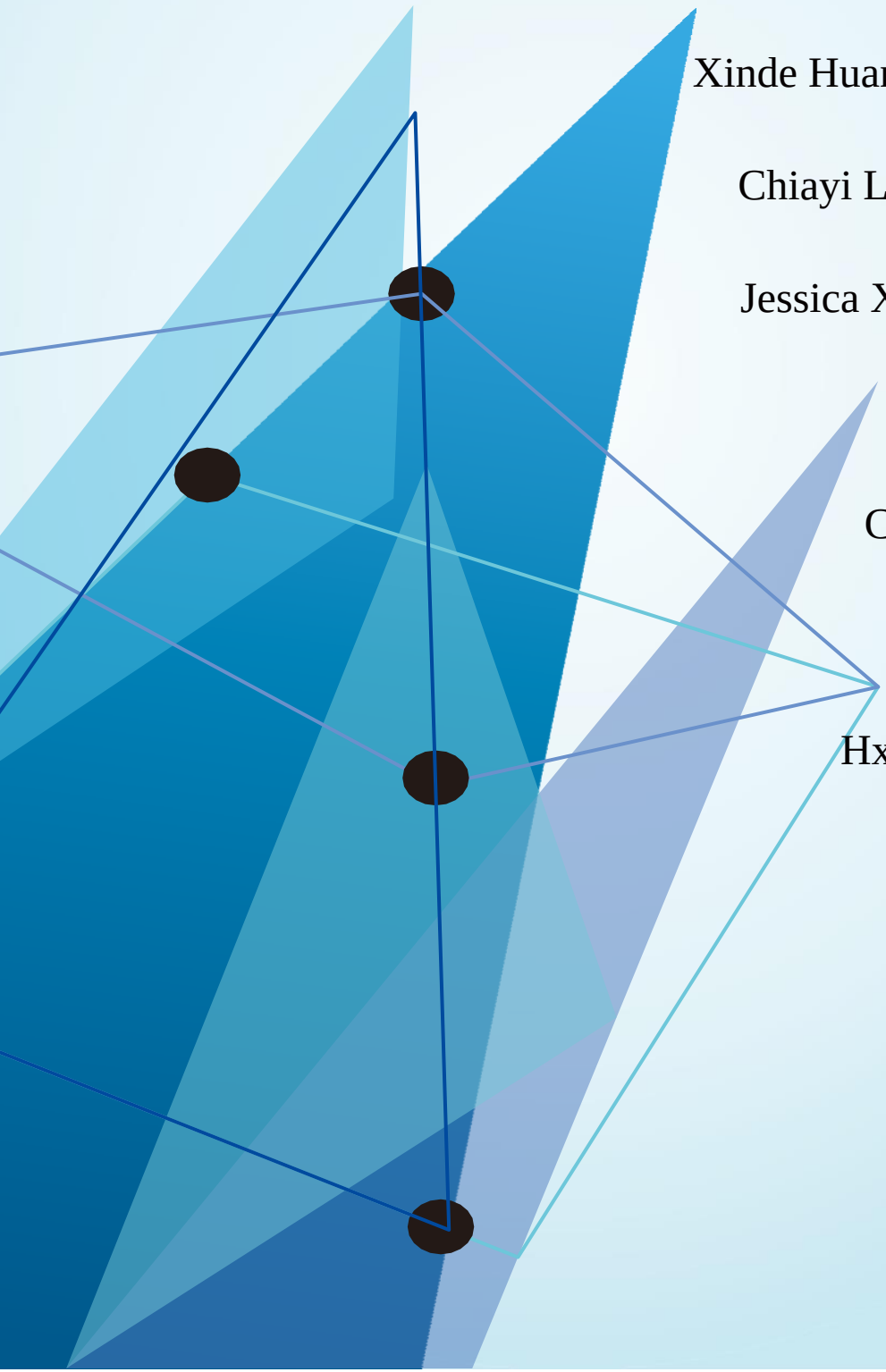
DSO 530
Options Pricing Group Project



Name:

Xinde Huang 1639159627

Chiayi Lin 9704956921

Jessica Xu 2046672670

Group 23

Contact Person:

Jessica Xu

Hxu577@usc.edu

I. Executive Summary

In financial engineering, the option is an essential derivative tool. With the development of financial engineering and the knowledge of options, various options have grown in the market. The European call option is one of the most representative options. The European call option gives holders the right but not the obligation to purchase the security at a given time for a given price. For example, you can use 5 euros to buy the right to acquire the ASML Holding on a specific date (expiration date) for 600 euros (a particular price). However, since the cost of the call option depends on the future value that buyers predict, the call option price fluctuates. The accuracy of prediction of the current option value is vital to work. Nowadays, machine learning has become mature. By combining the trading data and the algorithm, we can apply machine learning to predict the option's current value and improve the management level of the options portfolio and the efficiency of the investment strategy. This project aims to develop algorithms in the regression model and classification model. The regression and classification models would be used to predict the current option value and label if the current price is underestimated or overestimated based on the rule of the Black-Scholes formula.

This report contains the following sections: data preparation, regression exploration, classification exploration, results, business insight, and conclusion.

Data preparation illustrates how to deal with null, zero, and outlier values. We split the training data first and use the rest of the data as a so-called testing data set to validate the performance and the possibility of overfitting in training data. Regression exploration will develop Linear Regression, K Nearest Neighbor, Decision Tree Regression, Random Forest Regressor, and Neural Network Regressor to predict the current call option price. We will modify our hyperparameters until we find the best performance in training data. Then we will tune our hyperparameters and use 10 folds cross-validation to calculate its mean out of sample R square. Furthermore, we validated the split testing dataset and retrained the split training and testing to predict the current call option value in actual testing data without labels.

Last but not least, classification exploration will use Logistic Regression, K Nearest Neighbor, Decision Tree, Random Forest, and Neural Network. It will follow the same methodology as regression exploration to label whether the value is under or over.

II. Data Preparation

We first removed all the outliers from every variable. We excluded r that is bigger than 0.0315 and tau that is bigger than 1. Then we replaced all zero values with null values and filled the null values by using KNN imputer. To further ensure prediction's accuracy, we also checked the data quality of the test data set and made sure that the data is already cleaned. We have one hot encoded BS column, so 0 means undervalue, and 1 means overvalue for classification exploration.

Since we are doing unsupervised learning, to validate the performance of our model and inspect the problem of overfitting, we would like to perform supervised learning first. We separated 70% of option_train into training data and used the rest of 30% as testing data. After cross validating our models and having all parameters tuned for best performance, we trained the selected model with the 70% training and make predictions on the separated 30% testing set. And to see if our model's outstanding performance is only limited to the training set and therefore had the problem of overfitting, we fitted the 70% training data into the model with final parameters, predicted on the 30% testing set, and compared prediction with actual labels as validation of model performance. At the end we retrained the model with the entire option_train dataset and made prediction on the real testing data without labels.

III. Regression

To predict current option value, we explored various regression models varying in complexity. We first used basic Multiple Linear Regression model. The mean r squared after 10 folds cross validation for the training set is 90.95%. After fitting the model, the out-of-sample r-squared for test dataset is 90.04%.

For the K Nearest Neighbor Regression model, we ran a range of K from 1 to 10 to find the optimal performance. We find that when K is 4, the mean R-squared after 10 folds cross validation for the training set is the highest, 97.17%. After fitting the model with K = 4, the out-of-sample R-squared for test dataset is 96.70%.

For the Decision Tree Regression model, we first ran the default parameters, and the mean R-squared is 98.43%. Then, we tuned the hyperparameters. We find that when the splitter is best, max_depth is None, min_samples_split is 2, min_samples_leaf is 1, max_feature is auto and the performance is optimal. The mean r squared after 10 folds cross validation for the training set is 98.44%. After fitting the model, the out-of-sample r-squared for test dataset is 99.11%.

For the Random Forest Regression model, we tuned the hyperparameters. We find that when n_estimators is 500, max_depth is 40, random_state is 1, the mean R-squared after 10 folds cross validation for the training set is the highest, 99.37%. After fitting the model the out-of-sample R-squared for test dataset is 99.62%.

For the Neural Net Regression model, after tuning, we find that when number of hidden layers and node sizes are 8, max_iter is 6000, solver is 'lbfgs', random_state is 3, the mean R-squared after for training set is the highest, 99.94%. After fitting in the model, the out-of-sample R-squared for test dataset is 99.92%.

IV. Classification

To predict whether the call option will be over or under priced, we used various classification models. We explored models from the simplest ones and gradually increased complexity, including logistic regression, k nearest neighbors, decision tree, random forest, and neural network classifier.

We first used Logistic Regression model. The average classification error out of 10-fold cross validation for the training set is 8.53%, while prediction on testing set returns 8.37% classification error. And the ROC curve (appendix II.1) for logistic regression has an area under the curve equaling to 0.9750.

We then moved on to K Nearest Neighbor model for classification. By running 10-folds cross validation on number of neighbors ranging from 1 to 10, we found that when there are 9 groups, the model gives the highest accuracy with an average classification error of 7.28%. And the prediction on testing set returns 9.00% classification error. And the ROC curve for K-Nearest Neighbor (appendix II.2) has the area under the curve equaling to 0.9758.

The third model we tried is Decision Tree. Decision tree has the advantage in terms of interpretation. It more closely mirror human decision-making and does not contain any complex mathematical formula for communication. The 10-folds cross validation shows us the best combination of parameter in terms of classification error: (splitter='best', max_depth=30, max_features='auto', min_samples_split=4, min_samples_leaf=1, random_state=3). And the average classification error with these parameters is 8.89%, while the predictions made on testing set returns a result of 9.00%. And the ROC curve (appendix II.3) has a resulted AUC of score 0.9610.

Although decision tree has the advantage in interpretation, the resulted accuracy is not as ideal as we expected, we therefore moved on to a more complex method built on decision tree: Random Forest. Random forests provide an improvement over simple decision trees by bagging trees together and splitting on a random subset of features on each split to decollerate trees. The combination of parameters that can give us highest accuracy is as following: (n_estimators=500, max_depth=50, min_samples_leaf = 2, random_state=1). And the resulted average classification error is 7.00%, while the classification error on the 30% testing set is 7.53%. The ROC curve (appendix II.4) for our random forest model has an AUC score of 0.9804.

The final and the most complex model that we put into consideration is Neural Network's Multi-layer Perceptron classifier. This classifier has the advantage that the same accuracy score can still be reached even when our dataset becomes smaller, which can be more beneficial to our final prediction on the real test set without labels. After running 10-fold cross-validations, we altered two parameters and reached the best performance: we adjusted maximum iteration to 4000 and random state to 6. The resulted average classification error is 6.65% and the prediction returns a 6.49% classification error. Overfitting is also not a problem for this model. And the 0.9848 AUC score is also very promising (appendix II.5).

V. Results

Based on the results returned from the models we selected, we came to our final approach that we would use to make predictions on the real test set without labels. To predict Value, we would use Muti-layer Perceptron Regressor with the customized parameters: hidden_layer_sizes=(8,8),max_iter=6000,solver='lbfgs',random_state=3. It provides the highest average R squared of 99.94% after cross validation and also 99.92% on the artificial testing set which helps rule out the problem of overfitting. This R squared score indicates that MLP Regressor will be able to expain most of the variabilities in the dataset.

As for predicting whether the call option is over or under priced, we would also use Multi-layer Perceptron Classifier with the 4000 maximum iteration and 6 as random state. This model has the lowest average classification error of 6.65% after cross validation and 6.49% error on the artificial testing set. It also has the highest AUC score of 0.9848, which indicates that the MLP classifier has the strongest the ability to distinguish between classes.

And therefore, based on MLP Regressor's and MLP Classifier's outstanding performance on the entire option_train dataset, we believe that these two models will be able to provide the most accurate prediction on the test set.

VI. Business Insights

In our perspective, prediction accuracy takes more importance than interpretation. Accuracy in machine learning demonstrates how accurately the model performs to predict the occurrence of value by using data. As the accuracy increases, the complexity of the model grows exponentially, which illustrates that it would be harder to interpret. In the real-world financial scenario, interpretability is also vital but is secondary. As long as the more accurate model performs better in actual-world data than the balanced accuracy and interpretation model, the company can achieve a better portfolio to gain more profit for stakeholders. It would be a challenging situation if you could interpret your model, but it does not have a desirable return. Hence, we think that prediction accuracy takes more weight on the trade-off in this case.

Moreover, we believe machine learning might outperform BS in predicting option values. Black-Scholes model was invented by using current stock prices, expected dividends, the option's strike price, expected interest rates, time to expiration, and expected volatility. In our perspective, it is a fixed equation with these variables under some assumptions. In machine learning, we can furthermore make some feature engineering and feature selection to find out more factors or features that outperform the BS. Moreover, machine learning works because the computer is identifying patterns, and finding out underneath connections and insights from data. Technically, the more data model can train, the better performance it will be. In extracting patterns, the computer might have better performance than humans.

From the overall business perspective, variable r is the risk-free rate / annual interest rate which demonstrates in a business sense that the longer time of maturity, the more powerful that r is. Since the biggest tau in our cleaned dataset is still smaller than 0.4

year, the r might not have a strong prediction power. Also, based on our correlation heatmap, r is the least correlated variable with value so in the regression model, r may not contribute too much. Hence we think r can be excluded.

Despite the power of the machine learning model and the Black Scholes model, there are some weaknesses and limitations. Since our variables are used to predict the value and classify labels under the rule of a European call option, Tesla is an American stock that follows the direction of the American option, which means that it could be exercised before the expiration date. Our model might not predict value as good as we did in our test data set. Moreover, we are doing a machine learning model under the assumptions that no dividends are paid out, no transactions cost in the call option, and the risk-free rate and underlying volatility are constant. In the real-world situation, other than above variable, we also have to consider taxes, commission, trading costs, and some market movement. For example, Elon Musk said that customers could use dogecoins to buy Tesla, or the government announced that Tesla would be gov-reserve vehicles for government events. These news, policy, and even five sigma event such as Covid are the things that our model cannot predict. In this case, we cannot directly use our model to predict Tesla's stock correctly.

VII. Conclusion

1. Summary

We appended our exploratory data analysis in the appendix. We cleaned the data and dealt with outliers including tau and r. We then scaled the option_train dataset and divided this whole train set into training and testing to inspect the problem of overfitting. We used different regression and classification models to analyze the data and we came into the conclusion that neural network works best both on predicting value and classifying the label for the call option.

2. Future work

With the current model in mind, we want to continue optimizing the model and making more accurate predictions on future real world data in terms of call option pricing. For the first step, we can directly use the entire dataset as the training set, which will allow us to have more samples for training. In this way, the model will have the chance to learn more thoroughly, and have a better understanding of the interactions between the data. Only by continuing to train the model with more data can the prediction of future data be more accurate.

And after freezing our choices on hyperparameters and the model architecture, we can continuously obtain more up-to-date option pricing data and use the new data as our test and validation part to observe the model’s performance on accurate prediction. And with even more data coming in, the model training process will become a cycle: we can further merge the test and validation data and apply these new data onto our existing model for training purposes. And by consistently inspecting how our current model works on the incoming data, we can further adjust the hyperparameters used and gradually improve our model’s accuracy on doing real-world prediction.

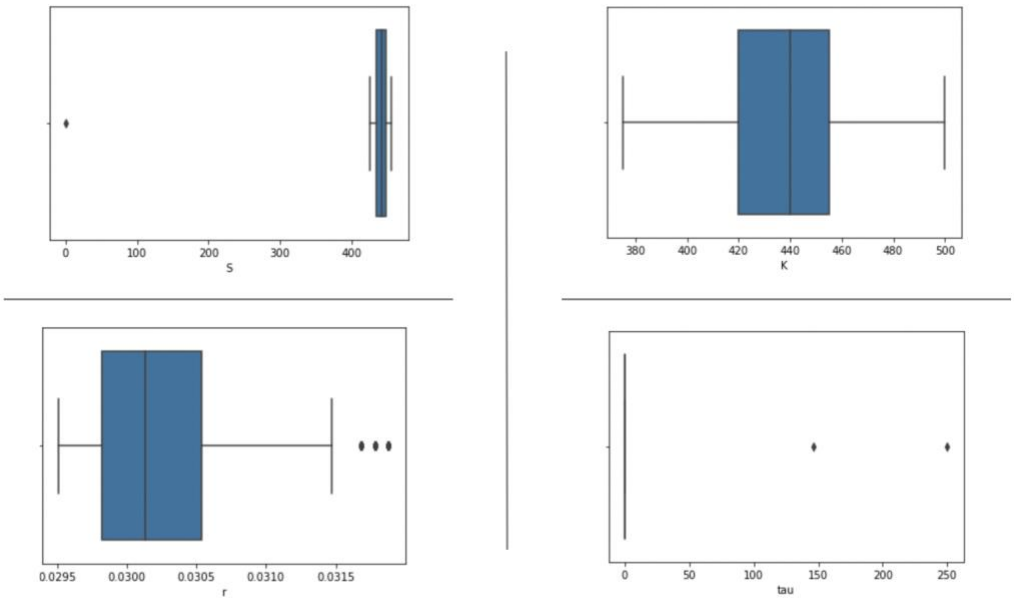
Appendix

I. EDA

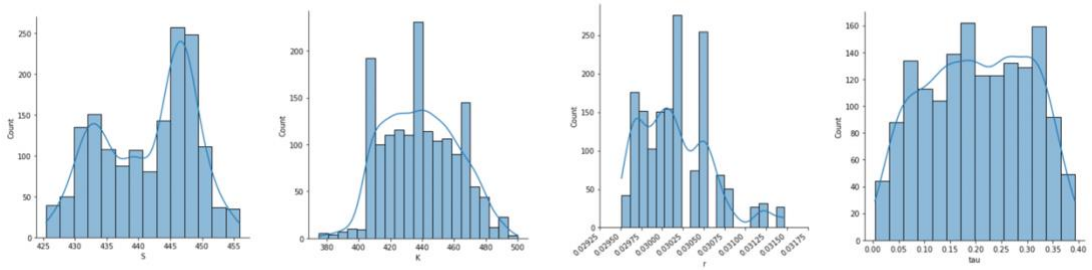
EDA demonstrates the original dataset, contains a boxplot of each variable, scatter plot of the dependent variable and independent variable, distribution of each variable, and correlation.

```
RangeIndex: 1680 entries, 0 to 1679
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Value   1679 non-null   float64
 1   S       1679 non-null   float64
 2   K       1678 non-null   float64
 3   tau     1679 non-null   float64
 4   r       1680 non-null   float64
 5   BS      1680 non-null   object
```

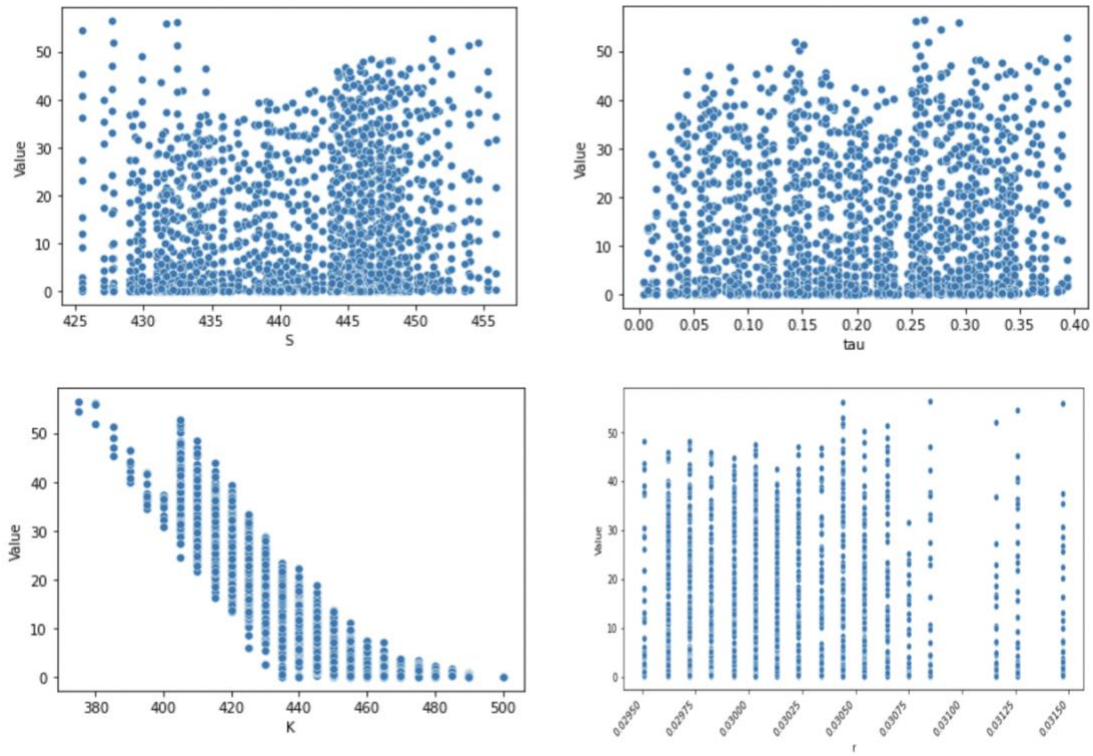
In the dataset, Value means current option value, S means current asset value, k means strike price, r means the annual interest rate, and tau implies the time of maturity. Bs is the label that used the Black-Scholes formula to predict the Value. If the expected value is larger than the Value, it will be labeled as over, and vice versa. We should pretend we don't know the BS formula in this project.



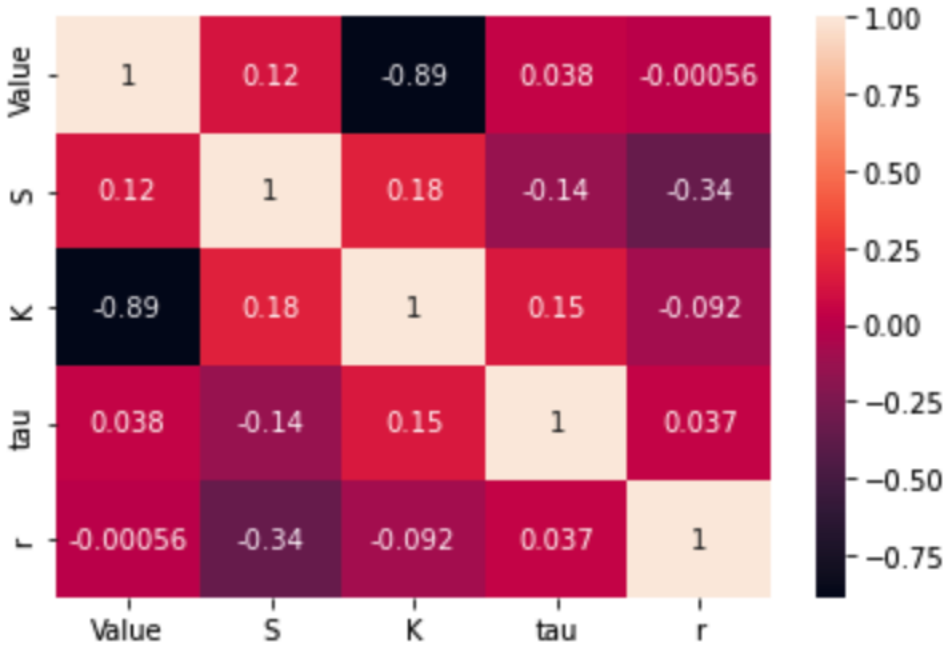
From the box plot of each variable, we can observe that S has one outlier, tau has two outliers, and r has three outliers.



From the distribution of each variable, we found out that S has right skewness, K tends to have slight right skewness, r has left skewness, and tau has slight right skewness.



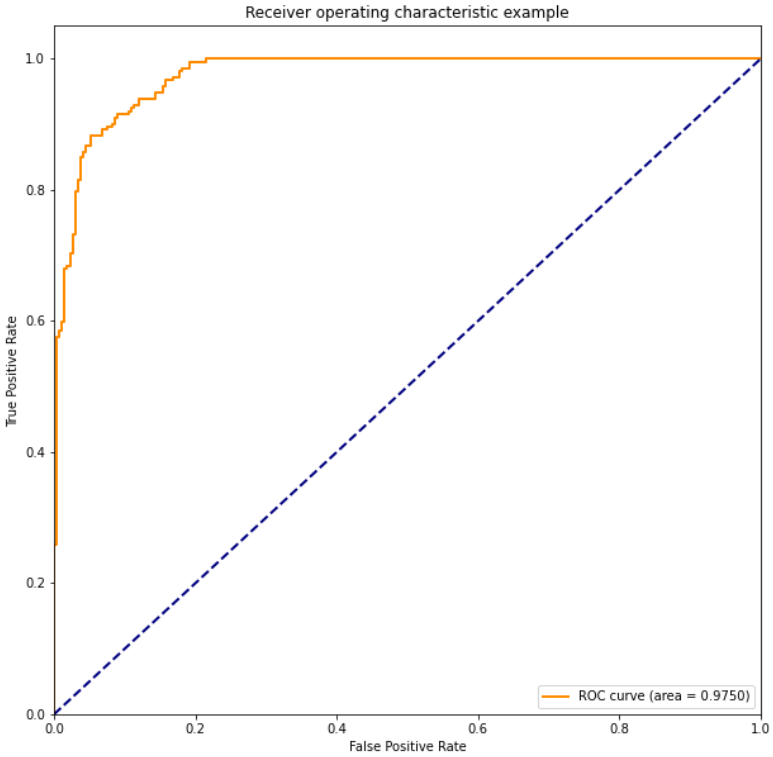
From the scatter plot of each variable, S, tau, and r show no linear relationship with value. However, as K increases, the value goes down.



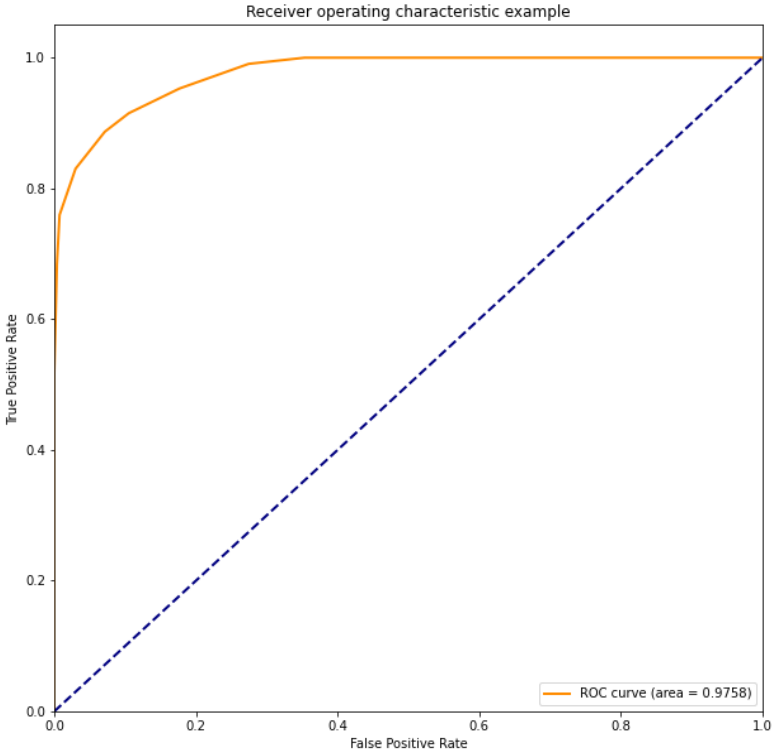
From correlation heatmap, it illustrates that K is the most correlated variable with value and r is the least correlated.

II. ROC Curves for Classification

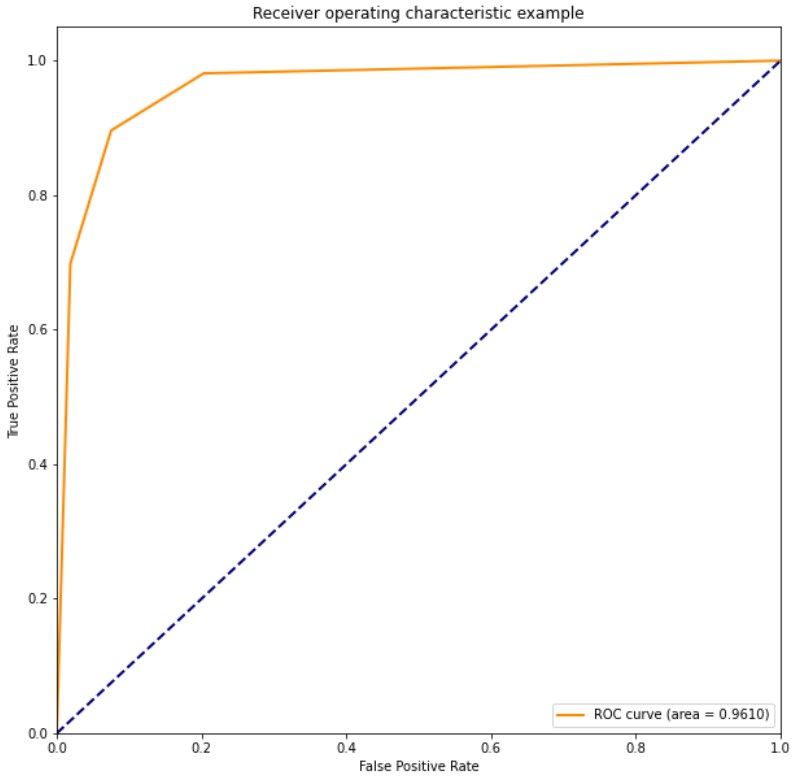
1. Logistic Regression



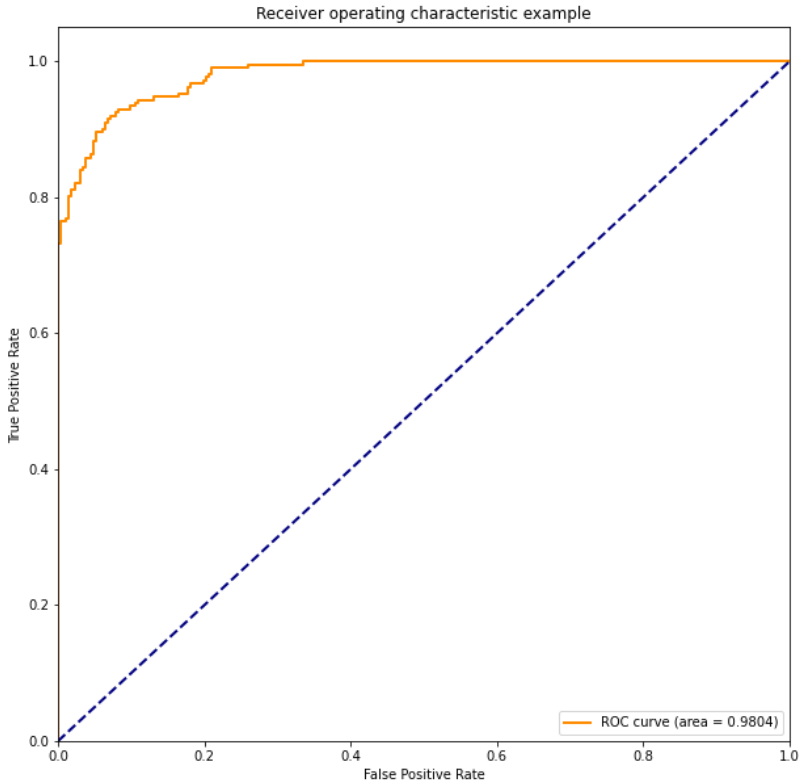
2. K-Nearest Neighbor



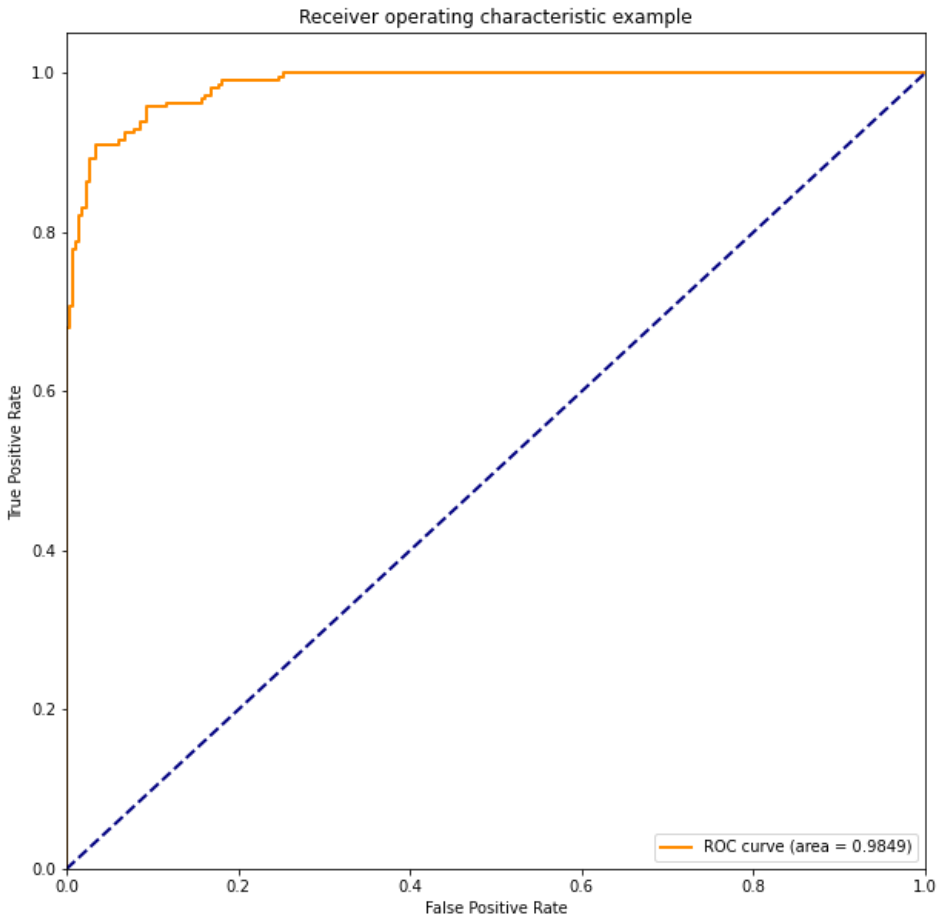
3. Decision Tree



4. Random Forest

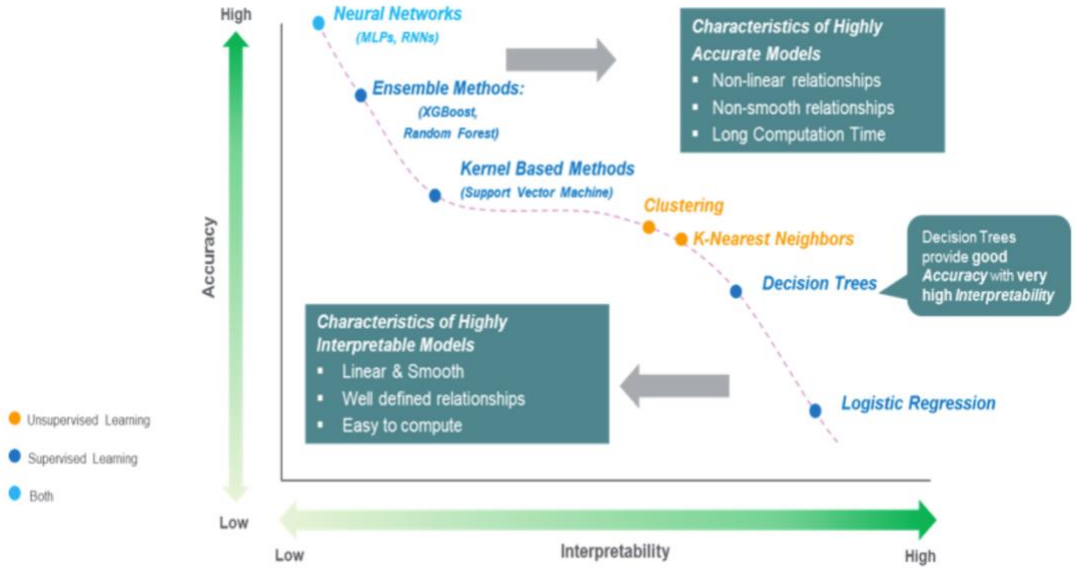


5. Neural Network - MLP Classifier



III. Business Insights

Interpretability vs. Accuracy



IV. Our Codes

Group Project - train-test-split 2.0

May 5, 2022

[1]:
```
import pandas as pd
df=pd.read_csv('option_train.csv')
```

[2]:
```
import numpy as np
import itertools
import seaborn as sns
import matplotlib.pyplot as plt
```

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

[3]:

	Value	S	K	tau	r	BS
0	21.670404	431.623898	420.0	0.341270	0.03013	Under
1	0.125000	427.015526	465.0	0.166667	0.03126	Over
2	20.691244	427.762336	415.0	0.265873	0.03116	Under
3	1.035002	451.711658	460.0	0.063492	0.02972	Over
4	39.553020	446.718974	410.0	0.166667	0.02962	Under

0.1 Drop Outliers

[8]:
```
df=df.loc[df['r']<0.0315]
```

[9]:
```
df['S']=df['S'].replace(0, np.nan)
```

[10]:
```
df=df.loc[df['tau']<1]
```

[21]:
```
bs= list(df.BS.values)
```

[22]:
```
df=df.drop('BS', axis=1)
```

[23]:
```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5, weights="uniform")
```

[24]:
```
x = df.to_numpy()
```

[25]:
```
x=imputer.fit_transform(x)
```

[26]:
```
df=pd.DataFrame(x, columns=['Value', 'S', 'K', 'tau', 'r'])
df['BS']=bs
```

[27]:
```
bs={'Under':0, 'Over':1}
```

[28]:
```
df.BS=df.apply(lambda x: 0 if x['BS']=='Under' else 1, axis=1)
```

[29]:
```
df.sort_values(by='r', ascending=False)
```

[29]:

	Value	S	K	tau	r	BS
850	16.278014	431.618600	425.0	0.293651	0.03147	0
833	5.065004	432.633296	440.0	0.182540	0.03147	1
1494	25.680681	434.772855	410.0	0.043651	0.03147	0
26	13.074008	431.618600	430.0	0.293651	0.03147	0
1176	7.000001	431.618600	440.0	0.293651	0.03147	1
…	…	…	…	…	… ..	
1158	3.940000	449.367877	460.0	0.198413	0.02951	1
1167	2.940001	448.218393	470.0	0.309524	0.02951	1
1177	1.125000	430.346700	455.0	0.222222	0.02951	1
794	0.280001	451.335247	465.0	0.059524	0.02951	1
654	38.996764	448.218393	415.0	0.309524	0.02951	0

[1591 rows x 6 columns]

[30]:
```
data = df.copy()
```

1 Regression

[31]:
```
from sklearn.model_selection import KFold ## for regression
from sklearn.model_selection import cross_val_score
```

[32]:
```
# test and train
from sklearn.model_selection import train_test_split
x, y = df.iloc[:, 1:5].values, df.iloc[:, 0].values
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.3,
                                                    random_state=0)

# standardized x
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train = stdsc.fit_transform(x_train)
X_test = stdsc.transform(x_test)
```

1.0.1 Linear Regression

[33]:
```
from sklearn.linear_model import LinearRegression
kfolds_regresssion = KFold(n_splits = 10, random_state = 0, shuffle = True)
lr = LinearRegression()
r2_lr_cv = cross_val_score(lr, X_train, y_train, cv=kfolds_regresssion)
print("Linear Regression Training Set: \n")
print("r squared of 10-folds:",r2_lr_cv,"(mean r squared:",np.
 ↪mean(r2_lr_cv),")")
```

```
Linear Regression Training Set:

r squared of 10-folds: [0.90764632 0.90522653 0.91814715 0.9163956  0.90469132
0.8949252
 0.90855433 0.91369437 0.90632311 0.91974475] (mean r squared:
0.9095348676198004 )
```

[34]:
```
from sklearn.metrics import r2_score
```

[35]:
```
lr.fit(X_train,y_train)
y_pred = lr.predict(X_test)
r2_score(y_pred,y_test)
```

[35]:
```
0.9004014209325477
```

1.0.2 K Nearest Neighbor

[36]:
```
from sklearn.neighbors import KNeighborsRegressor
# K Nearest Neighbor Regression
knn_regression = KNeighborsRegressor()
r2_knnr_cv = cross_val_score(knn_regression, X_train, y_train,␣
 ↪cv=kfolds_regresssion)
print("K Nearest Neighbor Regression: \n")
print("r squared of 10-folds:",r2_knnr_cv,"(mean r squared:",np.
 ↪mean(r2_knnr_cv),")")
```

```
K Nearest Neighbor Regression:

r squared of 10-folds: [0.96505082 0.97621055 0.9710316  0.97177925 0.97631752
0.98035226
 0.96291611 0.97535314 0.95698319 0.97421883] (mean r squared:
0.9710213262897239 )
```

[37]:
```
# K Nearest Neighbor Regression
for K in range(1,8):
    knn_regression = KNeighborsRegressor(K)
```

```
    r2_knnr_cv_tune = cross_val_score(knn_regression, X_train, y_train,␣
↪cv=kfolds_regresssion)
    print(f"K Nearest Neighbor Regression: {K}\n",)
    print("r squared of 10-folds:",r2_knnr_cv_tune,"(mean r squared:",np.
↪mean(r2_knnr_cv_tune),")\n")
```

```
K Nearest Neighbor Regression: 1

r squared of 10-folds: [0.93697086 0.94668903 0.94916178 0.9433132  0.94989316
0.95254533
 0.93987829 0.93141116 0.94075897 0.9322506 ] (mean r squared:
0.9422872389757003 )

K Nearest Neighbor Regression: 2

r squared of 10-folds: [0.95709878 0.96366634 0.97561089 0.96175162 0.97169095
0.9771769
 0.95786818 0.96615742 0.96434436 0.96760311] (mean r squared:
0.9662968539101087 )

K Nearest Neighbor Regression: 3

r squared of 10-folds: [0.96332859 0.97088076 0.97366857 0.97168078 0.97553323
0.97981382
 0.96589391 0.97027952 0.96221611 0.97329067] (mean r squared:
0.9706585968285376 )

K Nearest Neighbor Regression: 4

r squared of 10-folds: [0.96443909 0.9748268  0.97223981 0.97109051 0.97016579
0.9816394
 0.966263   0.97542042 0.96154308 0.97898762] (mean r squared:
0.9716615532786748 )

K Nearest Neighbor Regression: 5

r squared of 10-folds: [0.96505082 0.97621055 0.9710316  0.97177925 0.97631752
0.98035226
 0.96291611 0.97535314 0.95698319 0.97421883] (mean r squared:
0.9710213262897239 )

K Nearest Neighbor Regression: 6

r squared of 10-folds: [0.96126921 0.97437968 0.97011133 0.96666313 0.97226743
0.9798009
 0.95569599 0.97343791 0.95669525 0.97405214] (mean r squared:
0.9684372966551763 )
```

```
K Nearest Neighbor Regression: 7

r squared of 10-folds: [0.9604012  0.97247361 0.96763264 0.96813807 0.96953026
0.9810096
 0.95502368 0.96862513 0.95430389 0.97052868] (mean r squared: 0.966766675666802
)
```

[38]:
```
# With K = 4
knn_regression = KNeighborsRegressor(4)
r2_knnr_cv = cross_val_score(knn_regression, X_train, y_train,␣
 ↪cv=kfolds_regresssion)
print("K Nearest Neighbor Regression: \n")
print("r squared of 10-folds:",r2_knnr_cv,"(mean r squared:",np.
 ↪mean(r2_knnr_cv),")")
```

```
K Nearest Neighbor Regression:

r squared of 10-folds: [0.96443909 0.9748268  0.97223981 0.97109051 0.97016579
0.9816394
 0.966263   0.97542042 0.96154308 0.97898762] (mean r squared:
0.9716615532786748 )
```

[39]:
```
knn_regression.fit(X_train,y_train)
y_pred = knn_regression.predict(X_test)
r2_score(y_pred,y_test)
```

[39]:
```
0.9669632030427612
```

1.0.3 Decision Tree Regression

[40]:
```
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
```

[41]:
```
# Decision Tree with 2 maximum depth
decision_tree_regression = DecisionTreeRegressor()
r2_dcr_cv = cross_val_score(decision_tree_regression, X_train, y_train,␣
 ↪cv=kfolds_regresssion)
print("Decision Tree for Regression: \n")
print("r squared of 10-folds:",r2_dcr_cv,"(mean r squared:",np.
 ↪mean(r2_dcr_cv),")")
```

```
Decision Tree for Regression:

r squared of 10-folds: [0.98553197 0.99322641 0.98275564 0.98866475 0.97337953
0.98878382
```

```
 0.97860333 0.97829937 0.98684748 0.98713265] (mean r squared:
0.9843224963396293 )
```

[42]:
```
splitte = ['best','best','best','best','random','random','random']
max_dept = [None,None,None,None,10,10,10]
min_samples_spli = [2,5,10,15,2,5,10]
min_samples_lea = [1,2,5,10,20,40,40]
max_feature = ['auto','auto','auto','auto','auto','log2','log2']
```

[43]:
```
for i in range(len(splitte)):
    dcr = DecisionTreeRegressor(splitter=splitte[i],␣
↪max_depth=max_dept[i],max_features=max_feature[i],min_samples_split=min_samples_spli[i],
                                                    ␣
↪min_samples_leaf=min_samples_lea[i],random_state=0)
    r2_dcr_cv_tune = cross_val_score(dcr, X_train, y_train,␣
↪cv=kfolds_regresssion)
    print("Decision Tree:")
    print(f'Combinations:
↪{min_samples_spli[i],min_samples_lea[i],max_dept[i],min_samples_lea[i],splitte[i],max_featu
    print("r squared of 10-folds:",r2_dcr_cv_tune,"(mean r squared:",np.
↪mean(r2_dcr_cv_tune),")")
```

```
Decision Tree:
Combinations:(2, 1, None, 1, 'best', 'auto')

r squared of 10-folds: [0.98454272 0.99357656 0.98291973 0.98879693 0.97908586
0.98961796
 0.97900608 0.97814886 0.98452448 0.9852228 ] (mean r squared:
0.9845441976133241 )
Decision Tree:
Combinations:(5, 2, None, 2, 'best', 'auto')

r squared of 10-folds: [0.9814974  0.9909117  0.98601592 0.98787381 0.97169302
0.98724245
 0.97496691 0.98126535 0.98118136 0.98429445] (mean r squared:
0.9826942361403826 )
Decision Tree:
Combinations:(10, 5, None, 5, 'best', 'auto')

r squared of 10-folds: [0.98018178 0.98735549 0.98297044 0.978911   0.97442219
0.98303735
 0.97732694 0.96966466 0.96418544 0.97621794] (mean r squared:
0.9774273231029629 )
Decision Tree:
Combinations:(15, 10, None, 10, 'best', 'auto')

r squared of 10-folds: [0.96577272 0.9824225  0.97156771 0.9758569  0.9592801
```

```
0.96838788
 0.96844258 0.96206448 0.97274885 0.96542699] (mean r squared:
0.9691970730807806 )
Decision Tree:
Combinations:(2, 20, 10, 20, 'random', 'auto')

r squared of 10-folds: [0.83615367 0.87231628 0.85436659 0.85414059 0.82851403
0.87044509
 0.83248925 0.87406077 0.76385517 0.84787014] (mean r squared: 0.843421157665803
)
Decision Tree:
Combinations:(5, 40, 10, 40, 'random', 'log2')

r squared of 10-folds: [0.77881251 0.31820571 0.37620806 0.54911252 0.44290239
0.42424199
 0.32260595 0.33072281 0.2553267  0.41680532] (mean r squared:
0.4214943958501579 )
Decision Tree:
Combinations:(10, 40, 10, 40, 'random', 'log2')

r squared of 10-folds: [0.77881251 0.31820571 0.37620806 0.54911252 0.44290239
0.42424199
 0.32260595 0.33072281 0.2553267  0.41680532] (mean r squared:
0.4214943958501579 )
```

[44]:
```
decision_tree_regression =␣
 ↪DecisionTreeRegressor(splitter='best',max_depth=None,max_features='auto',min_samples_split=
r2_dcr_cv = cross_val_score(decision_tree_regression, X_train, y_train,␣
 ↪cv=kfolds_regresssion)
print("Decision Tree for Regression: \n")
print("r squared of 10-folds:",r2_dcr_cv,"(mean r squared:",np.
 ↪mean(r2_dcr_cv),")")
```

```
Decision Tree for Regression:

r squared of 10-folds: [0.98450509 0.99251502 0.98335147 0.98881712 0.97964369
0.98831637
 0.97899339 0.97705026 0.98515233 0.98550843] (mean r squared:
0.9843853168945014 )
```

[45]:
```
decision_tree_regression.fit(X_train,y_train)
y_pred = decision_tree_regression.predict(X_test)
r2_score(y_pred,y_test)
```

[45]: 0.9910501584558301

1.0.4 Random Forest Regressor

[46]:
```
from sklearn.ensemble import RandomForestRegressor
```

[48]:
```
random_forest_regression =␣
↪RandomForestRegressor(n_estimators=500,max_depth=40,random_state=1)
r2_rfr_cv = cross_val_score(random_forest_regression, X_train, y_train,␣
↪cv=kfolds_regresssion)
print("Random Forest for Regression: \n")
print("r squared of 10-folds:",r2_rfr_cv,"(mean r squared:",np.
↪mean(r2_rfr_cv),")")
```

```
Random Forest for Regression:

r squared of 10-folds: [0.99376534 0.99558265 0.99436321 0.99436028 0.99369353
0.9956534
 0.9899065  0.9928785  0.99540778 0.99150969] (mean r squared:
0.9937120879725899 )
```

[49]:
```
random_forest_regression.fit(X_train,y_train)
y_pred = random_forest_regression.predict(X_test)
r2_score(y_pred,y_test)
```

[49]:
```
0.9962033728359565
```

1.0.5 Neural Network Regressor

[50]:
```
from sklearn.neural_network import MLPRegressor
```

[51]:
```
nn_regression =␣
↪MLPRegressor(hidden_layer_sizes=(8,8),max_iter=6000,solver='lbfgs',random_state=3)
r2_nnr_cv = cross_val_score(nn_regression, X_train, y_train,␣
↪cv=kfolds_regresssion)
print("Neural Network for Regression: \n")
print("r squared of 10-folds:",r2_nnr_cv,"(mean r squared:",np.
↪mean(r2_nnr_cv),")")
```

```
Neural Network for Regression:

r squared of 10-folds: [0.99952537 0.99951923 0.99940415 0.99948144 0.99923212
0.99936448
 0.99910418 0.99942231 0.99948502 0.99937052] (mean r squared:
0.9993908814563459 )
```

[52]:
```
nn_regression.fit(X_train,y_train)
y_pred = nn_regression.predict(X_test)
r2_score(y_pred,y_test)
```

[52]: 0.9992286999470409

2 Classification

[54]:
```
# test and train
from sklearn.model_selection import train_test_split
x, y = df.iloc[:, 1:5].values, df.iloc[:, 5].values
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.3,
                                                    random_state=0,
                                                    stratify=y)

# standardized x
from sklearn.preprocessing import StandardScaler
stdsc = StandardScaler()
X_train = stdsc.fit_transform(x_train)
X_test = stdsc.transform(x_test)
```

2.0.1 Logistic Regression

[55]:
```
# ML Logit
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
```

[56]:
```
kfolds_classification = StratifiedKFold(n_splits = 10, random_state = 1,␣
↪shuffle = True)
```

[202]:
```
logistic_regression = LogisticRegression(max_iter=300)
error_lr_cv = cross_val_score(logistic_regression, X_train, y_train,␣
↪cv=kfolds_classification)
print("Logisgic Regression: \n")
print("accuracies of 10-folds:",error_lr_cv,"(mean classification error:",1-np.
↪mean(error_lr_cv),")")
```

```
Logisgic Regression Training Set:

accuracies of 10-folds: [0.90178571 0.84821429 0.94642857 0.95495495 0.90990991
0.91891892
 0.90990991 0.93693694 0.91891892 0.9009009 ] (mean classification error:
0.08531209781209781 )
```

[206]:
```
logistic_regression.fit(X_train,y_train)
y_pred = logistic_regression.predict(X_test)
print(1-(logistic_regression.score(X_test,y_test)))
```

```
0.08368200836820083
```

2.0.2 K Nearest Neighbor

[212]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[62]:
```
# K Nearest Neighbor Regression
knn_classification = KNeighborsClassifier()
error_knn_cv = cross_val_score(knn_classification, X_train, y_train,␣
 ↪cv=kfolds_classification)
print("K Nearest Neighbor Classification: \n")
print("accuracies of 10-folds:",error_knn_cv,"(mean classification error:",1-np.
 ↪mean(error_knn_cv),")\n")
```

```
K Nearest Neighbor Classification:

accuracies of 10-folds: [0.94642857 0.86607143 0.92857143 0.92792793 0.90990991
0.93693694
 0.95495495 0.91891892 0.89189189 0.89189189] (mean classification error:
0.082649613899614 )
```

[65]:
```
for K in range(1,11):
    knn = KNeighborsClassifier(n_neighbors=K)
    error_knn_cv_tune = cross_val_score(knn, X_train, y_train,␣
 ↪cv=kfolds_classification)
    print(f"K Nearest Neighbor {K}: \n")
    print("accuracies of 10-folds:",error_knn_cv_tune,"(mean classification␣
 ↪error:",1-np.mean(error_knn_cv_tune),")\n")
```

```
K Nearest Neighbor 1:

accuracies of 10-folds: [0.91964286 0.84821429 0.91071429 0.92792793 0.87387387
0.91891892
 0.91891892 0.92792793 0.93693694 0.85585586] (mean classification error:
0.09610682110682112 )

K Nearest Neighbor 2:

accuracies of 10-folds: [0.88392857 0.86607143 0.91964286 0.92792793 0.86486486
0.9009009
 0.90990991 0.93693694 0.91891892 0.82882883] (mean classification error:
0.10420688545688539 )

K Nearest Neighbor 3:

accuracies of 10-folds: [0.92857143 0.85714286 0.91964286 0.95495495 0.90990991
0.93693694
 0.97297297 0.92792793 0.88288288 0.88288288] (mean classification error:
0.08261743886743866 )
```

```
K Nearest Neighbor 4:

accuracies of 10-folds: [0.91071429 0.86607143 0.92857143 0.94594595 0.9009009
0.92792793
 0.93693694 0.91891892 0.9009009  0.88288288] (mean classification error:
0.08802284427284413 )

K Nearest Neighbor 5:

accuracies of 10-folds: [0.94642857 0.86607143 0.92857143 0.92792793 0.90990991
0.93693694
 0.95495495 0.91891892 0.89189189 0.89189189] (mean classification error:
0.082649613899614 )

K Nearest Neighbor 6:

accuracies of 10-folds: [0.9375     0.89285714 0.9375     0.94594595 0.90990991
0.92792793
 0.93693694 0.89189189 0.9009009  0.89189189] (mean classification error:
0.08267374517374526 )

K Nearest Neighbor 7:

accuracies of 10-folds: [0.95535714 0.86607143 0.9375     0.94594595 0.90990991
0.93693694
 0.95495495 0.91891892 0.9009009  0.91891892] (mean classification error:
0.07545849420849415 )

K Nearest Neighbor 8:

accuracies of 10-folds: [0.95535714 0.89285714 0.94642857 0.94594595 0.9009009
0.91891892
 0.94594595 0.90990991 0.9009009  0.91891892] (mean classification error:
0.07639157014157016 )

K Nearest Neighbor 9:

accuracies of 10-folds: [0.9375     0.88392857 0.94642857 0.94594595 0.9009009
0.95495495
 0.93693694 0.91891892 0.90990991 0.93693694] (mean classification error:
0.07276383526383534 )

K Nearest Neighbor 10:

accuracies of 10-folds: [0.91964286 0.88392857 0.94642857 0.94594595 0.9009009
0.92792793
 0.94594595 0.92792793 0.92792793 0.90990991] (mean classification error:
```

```
0.0763513513513514 )
```

[220]:
```
# K = 9
knn_classification = KNeighborsClassifier(n_neighbors=3)
error_knn_cv = cross_val_score(knn_classification, X_train, y_train,␣
 ↪cv=kfolds_classification)
print("K Nearest Neighbor Classification: \n")
print("accuracies of 10-folds:",error_knn_cv,"(mean classification error:",1-np.
 ↪mean(error_knn_cv),")\n")
```

```
K Nearest Neighbor Classification:

accuracies of 10-folds: [0.92857143 0.85714286 0.91964286 0.95495495 0.90990991
0.93693694
 0.97297297 0.92792793 0.88288288 0.88288288] (mean classification error:
0.08261743886743866 )
```

[221]:
```
knn_classification.fit(X_train,y_train)
y_pred = knn_classification.predict(X_test)
print(1-(knn_classification.score(X_test,y_test)))
```

```
0.08786610878661083
```

2.0.3 Decision Tree

[75]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

[115]:
```
decision_tree_classification = DecisionTreeClassifier(splitter='best',␣
 ↪max_depth=30,max_features='auto',min_samples_split=4,
                                              min_samples_leaf=1,random_state=3)
error_dcc_cv = cross_val_score(decision_tree_classification, X_train, y_train,␣
 ↪cv=kfolds_classification)
print("Decision Tree Classification: \n")
print("accuracies of 10-folds:",error_dcc_cv,"(mean classification error:",1-np.
 ↪mean(error_dcc_cv),")")
```

```
Decision Tree Classification Training Set:

accuracies of 10-folds: [0.86607143 0.875      0.91964286 0.91891892 0.9009009
0.91891892
 0.94594595 0.91891892 0.91891892 0.92792793] (mean classification error:
0.08888352638352637 )
```

[116]:
```
decision_tree_classification.fit(X_train,y_train)
y_pred = decision_tree_classification.predict(X_test)
print(1-(decision_tree_classification.score(X_test,y_test)))
```

```
0.08995815899581594
```

2.0.4 Random Forest

[117]:
```
from sklearn.ensemble import RandomForestClassifier
```

[193]:
```
random_forest =␣
↪RandomForestClassifier(n_estimators=500,max_depth=50,min_samples_leaf =␣
↪2,random_state=1)
error_rf_cv = cross_val_score(random_forest, X_train, y_train,␣
↪cv=kfolds_classification)
print("Random Forest Training Set: \n")
print("accuracies of 10-folds:",error_rf_cv,"(mean classification error:",1-np.
↪mean(error_rf_cv),")")
```

```
Random Forest Training Set:

accuracies of 10-folds: [0.90178571 0.88392857 0.94642857 0.92792793 0.93693694
0.93693694
 0.95495495 0.93693694 0.90990991 0.96396396] (mean classification error:
0.07002895752895755 )
```

[194]:
```
random_forest.fit(X_train,y_train)
y_pred = random_forest.predict(X_test)
print(1-(random_forest.score(X_test,y_test)))
```

```
0.07531380753138073
```

2.0.5 Neural Network

[136]:
```
from sklearn.neural_network import MLPClassifier
```

[185]:
```
nn_classification = MLPClassifier(max_iter=4000,random_state=6)
error_nn_cv = cross_val_score(nn_classification, X_train, y_train,␣
↪cv=kfolds_classification)
print("Neural Network: \n")
print("accuracies of 10-folds:",error_nn_cv,"(mean classification error:",1-np.
↪mean(error_nn_cv),")")
```

```
Random Forest Training Set:

accuracies of 10-folds: [0.9375     0.89285714 0.95535714 0.93693694 0.91891892
0.92792793
```

```
 0.95495495 0.92792793 0.92792793 0.95495495] (mean classification error:
0.06647361647361638 )
```

[186]:
```
nn_classification.fit(X_train,y_train)
y_pred = nn_classification.predict(X_test)
print(1-(nn_classification.score(X_test,y_test)))
```

```
0.06485355648535562
```

3 Prediction

[4]:
```
test = pd.read_csv('option_test_wolabel.csv')
```

[5]:
```
test
```

[5]:

	S	K	tau	r
0	431.618600	460	0.293651	0.03147
1	432.633296	420	0.182540	0.03147
2	432.633296	430	0.182540	0.03147
3	431.618600	415	0.293651	0.03147
4	434.772855	420	0.043651	0.03147
…	…	…	…	…
1115	440.067417	435	0.182540	0.02962
1116	439.081203	485	0.293651	0.02962
1117	439.081203	475	0.293651	0.02962
1118	442.490015	420	0.043651	0.02962
1119	440.067417	430	0.182540	0.02962

[1120 rows x 4 columns]

3.0.1 Value Prediction

[23]:
```
x, y = df.iloc[:, 1:5].values, df.iloc[:, 0].values
X = stdsc.fit_transform(x)
```

[39]:
```
x_validate = test.loc[:,:].values
X_validate = stdsc.transform(x)
```

[40]:
```
nn_regression =␣
↪MLPRegressor(hidden_layer_sizes=(8,8),max_iter=6000,solver='lbfgs',random_state=3)
```

[43]:
```
nn_regression.fit(X,y)
nn_regression.predict(X_validate)
```

[43]:
```
array([20.79154958,  0.16186091, 20.15703806, …, 33.7880619 ,
        0.22885454,  0.30120202])
```

[42]:
```
Value = list(nn_regression.predict(X_validate))
```

3.0.2 BS Prediction

[47]:
```
y1 = df.iloc[:,5].values
```

[45]:
```
nn_classification = MLPClassifier(max_iter=4000,random_state=6)
```

[48]:
```
nn_classification.fit(X,y1)
```

[48]:
```
MLPClassifier(max_iter=4000, random_state=6)
```

[49]:
```
nn_classification.predict(X_validate)
```

[49]:
```
array([0, 1, 0, …, 0, 1, 1])
```

[50]:
```
BS = list(nn_classification.predict(X_validate))
```

[54]:
```
result = pd.DataFrame({'Value':Value,'BS':BS})
```

[55]:
```
result
```

[55]:

	Value	BS
0	20.791550	0
1	0.161861	1
2	20.157038	0
3	0.976790	1
4	39.390134	0
…	…	..
1586	13.831893	0
1587	0.721464	1
1588	33.788062	0
1589	0.228855	1
1590	0.301202	1

[1591 rows x 2 columns]

[56]:
```
result.to_csv('group_23_prediction.csv')
```