Martin Vladimirov Karastoyanov

S2031121

Machine Learning and Data Analytics

Coursework

Word Count: 1665

"Except where explicitly stated, all work in this report, is my own original work and has not been submitted elsewhere in fulfilment of the requirement of this or any other award".

Signed by Student: __MK__ Date: 07/01/2024

Table of Contents

List of Figures

List of Tables

Introduction and Problem Definition

The rise of online marketplaces has led to an assortment of formats, including auctions and direct consumer sales. Notably, eBay emerged as a leading e-commerce website in 2010, capturing the attention of 2.7% of internet users worldwide each day, according to the Internet traffic statistics website - Alexa.com (Dong et al., 2012). Shill bidding poses a significant challenge in online auction environments, as it can lead to inflated prices and compromised auction integrity, impacting the fairness of the marketplace, and potentially causing economic harm to legitimate participants. Shill bidding refers to the common yet unethical involvement of sellers in auctions, aiming to inflate the selling price of an item (Chakraborty & Kosmopoulou, 2004).

The report aims to examine a dataset containing diverse attributes related to bidding. The objective is to develop a machine learning model capable of evaluating the Shill Bidding dataset to forecast if a bidder's behaviour falls within normal parameters or deviates to an abnormal pattern. This task will involve employing a classification method, with the Support Vector Machine (SVM) being the algorithm of choice for execution.

The dataset consists of 6321 instances and 12 attributes, excluding the one class variable (**Figure 1.1**).

```
# Loading Data

df = pd.read_csv('Data/Shill_Bidding_Dataset_With_Head.csv')
print(df.shape)
```

```
(6321, 13)
```

Figure 1.1

The attributes are described in more detail in **Table 1.1**

Attribute	Description
Record ID	Unique identifier of a record in the dataset.
Auction ID	Unique identifier of an auction.
Bidder ID	Unique identifier of a bidder.
Bidder Tendency	A shill bidder participates exclusively in auctions of few sellers rather than a diversified lot. This is a collusive act involving the fraudulent seller and an accomplice.
Bidding Ratio	A shill bidder participates more frequently to raise the auction price and attract higher bids from legitimate participants.
Successive Outbidding	A shill bidder successively outbids himself even though he is the current winner to increase the price gradually with small consecutive increments.
Last Bidding	A shill bidder becomes inactive at the last stage of the auction (more than 90% of the auction duration) to avoid winning the auction.
Auction Bids	Auctions with SB activities tend to have a much higher number of bids than the average of bids in concurrent auctions.
Auction Starting Price	A shill bidder usually offers a small starting price to attract legitimate bidders into the auction.
Early Bidding	A shill bidder tends to bid pretty early in the auction (less than 25% of the auction duration) to get the attention of auction users.
Winning Ratio	A shill bidder competes in many auctions but hardly wins any auctions.
Auction Duration	How long an auction lasted.
Class	0 for normal behavior bidding; 1 for otherwise.

Table 1.1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6321 entries, 0 to 6320
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Record_ID               6321 non-null   int64  
 1   Auction_ID              6321 non-null   int64  
 2   Bidder_ID               6321 non-null   object 
 3   Bidder_Tendency         6321 non-null   float64
 4   Bidding_Ratio           6321 non-null   float64
 5   Successive_Outbidding   6321 non-null   float64
 6   Last_Bidding            6321 non-null   float64
 7   Auction_Bids            6321 non-null   float64
 8   Starting_Price_Average  6321 non-null   float64
 9   Early_Bidding           6321 non-null   float64
 10  Winning_Ratio           6321 non-null   float64
 11  Auction_Duration        6321 non-null   int64  
 12  Class                   6321 non-null   int64  
dtypes: float64(8), int64(4), object(1)
memory usage: 642.1+ KB
```

Figure 1.2

Referencing **Figure 1.2**, which was created following the execution of the 'df.info()' script, the dataset primarily comprises numerical data, with the sole exception being the '**Bidder_ID'** attribute, categorised as the 'object' datatype. Moreover, the absence of null or missing entries within the dataset is noted, indicating its high level of cleanliness, and removing the necessity for any operations aimed at addressing missing values.

The dataset is designed for classification tasks, as opposed to regression models. Considering the dataset's moderate size and absence of missing or null values, the Support Vector Machine (SVM) algorithm emerges as an appropriate selection. This algorithm is noted for its computational efficiency and its consistent performance with small to medium-sized datasets. The use of SVM is likely to ensure accurate classification outcomes while also optimising computational resource utilisation. The decision to use the SVM classifier is based on the need to achieve high accuracy in results without excessive computational demands, which is facilitated by the current state of the dataset (Cervantes et al., 2020; Chi, Feng & Bruzzone, 2008).

Constructing and tuning the model

Data exploring and pre-processing

The dataset, as previously noted, is complete with no missing or inappropriate values, eliminating the need for any instanced to be omitted or any data imputation.

The execution of the script in **Figure 2.1** reveals the absence of any duplicate values in the dataset.

```
duplicate_rows_df = df[df.duplicated()]
print('number of duplicate rows: ', duplicate_rows_df)
```

```
number of duplicate rows:  Empty DataFrame
Columns: [Record_ID, Auction_ID, Bidder_ID, Bidder_Tendency, Bidding_Ratio, Successive_Outbidding, Last_Bidding, Auction_Bids,
Starting_Price_Average, Early_Bidding, Winning_Ratio, Auction_Duration, Class]
Index: []
```

Figure 2.1

A careful inspection of the dataset reveals that some attributes can be modified or dropped from the data frame.

For example, the **'Record_ID'** attribute, which is described as unique identifier of each instance, can be used as an index column (**Figure 2.2**).

```
df = pd.read_csv('Data/Shill_Bidding_Dataset_With_Head.csv', index_col='Record_ID')
print(df.shape)
df.head(5)
```

```
(6321, 12)
```

	Auction_ID	Bidder_ID	Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	W
Record_ID										
1	732	_***i	0.200000	0.400000	0.0	0.000028	0.0	0.993593	0.000028	
2	732	g***r	0.024390	0.200000	0.0	0.013123	0.0	0.993593	0.013123	
3	732	t***p	0.142857	0.200000	0.0	0.003042	0.0	0.993593	0.003042	
4	732	7***n	0.100000	0.200000	0.0	0.097477	0.0	0.993593	0.097477	
5	900	z***z	0.051282	0.222222	0.0	0.001318	0.0	0.000000	0.001242	

Figure 2.2

Another attribute that is a bit more special is the '**Bidder_ID**' (**Figure 1.2**), it is the only non-numerical data type. The attribute consists of encrypted unique names, which do not provide any useful information for the model, so it will be better for this attribute to be dropped using the script in **Figure 2.3**.

```
df = df.drop(columns = ['Bidder_ID'])
df.head(5)
```

	Auction_ID	Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Winning_Ratic
Record_ID									
1	732	0.200000	0.400000	0.0	0.000028	0.0	0.993593	0.000028	0.666667
2	732	0.024390	0.200000	0.0	0.013123	0.0	0.993593	0.013123	0.944444
3	732	0.142857	0.200000	0.0	0.003042	0.0	0.993593	0.003042	1.00000C
4	732	0.100000	0.200000	0.0	0.097477	0.0	0.993593	0.097477	1.00000C
5	900	0.051282	0.222222	0.0	0.001318	0.0	0.000000	0.001242	0.50000C

Figure 2.3

Executing the 'df.Auction_ID.value_counts()' script (**Figure 2.4**) reveals that there are 807 unique auctions. Given its nature as a unique identifier, retaining it in the data frame might introduce unnecessary noise. Consequently, it will be discarded from the data frame.

```
Auction_ID
589     26
1872    26
256     24
658     24
2498    23
        ..
1756     1
548      1
1971     1
458      1
2329     1
Name: count, Length: 807, dtype: int64
```

Figure 2.4

To gather statistical insights, the 'df.describe()' script is executed.

```
df.describe()
```

	Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average
count	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000	6321.000000
mean	0.142541	0.127670	0.103781	0.463119	0.231606	0.472821
std	0.197084	0.131530	0.279698	0.380097	0.255252	0.489912
min	0.000000	0.011765	0.000000	0.000000	0.000000	0.000000
25%	0.027027	0.043478	0.000000	0.047928	0.000000	0.000000
50%	0.062500	0.083333	0.000000	0.440937	0.142857	0.000000
75%	0.166667	0.166667	0.000000	0.860363	0.454545	0.993593
max	1.000000	1.000000	1.000000	0.999900	0.788235	0.999935

Figure 2.5.1

Early_Bidding	Winning_Ratio	Auction_Duration	Class
6321.000000	6321.000000	6321.000000	6321.000000
0.430683	0.367731	4.615093	0.106787
0.380785	0.436573	2.466629	0.308867
0.000000	0.000000	1.000000	0.000000
0.026620	0.000000	3.000000	0.000000
0.360104	0.000000	5.000000	0.000000
0.826761	0.851852	7.000000	0.000000
0.999900	1.000000	10.000000	1.000000

Figure 2.5.2

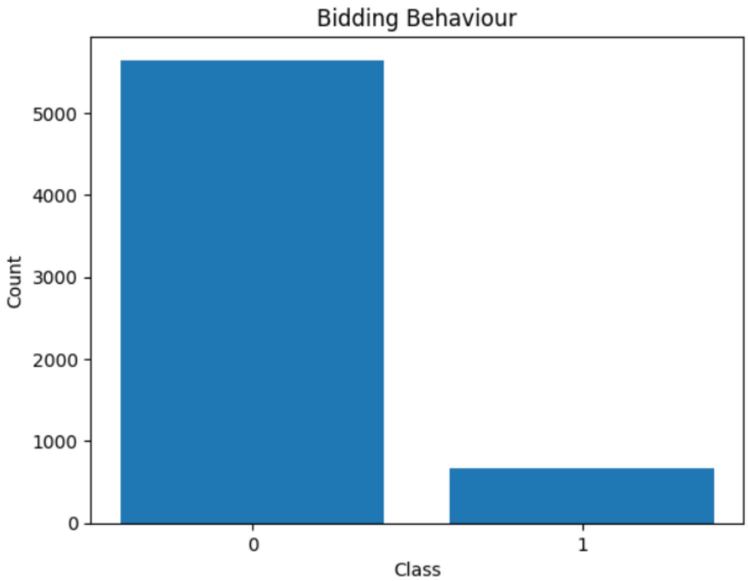
Reviewing the data in **Figures 2.5.1** and **Figure 2.5.2** reveals that the entire dataset has values ranging from 0 to 1, except for the **'Auction_Duration'** attribute. Given the sensitivity of the SVM classifier to the scale of input features (Singh & Singh, 2020), it would be advisable to normalize the **'Auction_Duration'**. Normalization, in this context, involves scaling the values to a range between 0 and 1 (Muhammad Ali & Faraj, 2014).

```
from sklearn.preprocessing import MinMaxScaler

df['Auction_Duration'] = MinMaxScaler().fit_transform(df[['Auction_Duration']])
df.head(5)
```

Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Winning_Ratio	Auction_Duration	Class
0.200000	0.400000	0.0	0.000028	0.0	0.993593	0.000028	0.666667	0.444444	0
0.024390	0.200000	0.0	0.013123	0.0	0.993593	0.013123	0.944444	0.444444	0
0.142857	0.200000	0.0	0.003042	0.0	0.993593	0.003042	1.000000	0.444444	0
0.100000	0.200000	0.0	0.097477	0.0	0.993593	0.097477	1.000000	0.444444	0
0.051282	0.222222	0.0	0.001318	0.0	0.000000	0.001242	0.500000	0.666667	0

Figure 2.6




```
Class
0    5646
1     675
Name: count, dtype: int64
```

Figure 2.7.1

The plot (**Figure 2.7.1**) generated by the script in **Figure 2.7.2**, depicting the distribution of normal (**0**) and abnormal (**1**) behavioural cases during the auctions. The visual representation provides insights into the imbalance or balance between the two classes.

```python
import matplotlib.pyplot as plt

class_counts = df['Class'].value_counts()

# Create a bar plot
plt.bar(class_counts.index.astype(str), class_counts.values)

# Set the title and labels
plt.title('Bidding Behaviour')
plt.xlabel('Class')
plt.ylabel('Count')

# Display the plot
plt.show()

print(class_counts)
```

Figure 2.7.2

The graph and corresponding **'class_counts'** data confirm an imbalance within the **'Class'** variable: the majority class (**0**) has 5646 entries, while the minority class (**1**) has significantly fewer, with only 675 entries. This discrepancy indicates that the dataset is skewed towards the 0 class, highlighting that there might be need for a strategy to address these imbalances during analysis (Ramyachitra & Manikandan, 2014).

Constructing the model

The dataset is divided into X variables and a y target. The X variables comprise the data frame excluding the class, while the y target consists of all instances of the class attribute.

```
X = df.drop(['Class'], axis=1)
y = df.Class
```

Figure 2.8

The **'train_test_split'** function is used to split the dataset (**X** and **y**) into training (**X_train** and **y_train**) and temporary data (**X_tmp** and **y_tmp**). The temporary data (**X_tmp** and **y_tmp**) is further split into validation (**X_validation and y_validation**) and test data (**X_test** and **y_test**).

This step ensures that the model is trained on a distinct subset of data, validated on another, and tested on yet another, preventing data leakage and providing a fair evaluation. The data is split in 70-15-15 (training-validation-test) ratio. Proper data splitting is essential for robust model training and evaluation. The 70-15-15 split ratio is commonly used to strike a balance between having sufficient data for model learning and ensuring a substantial dataset for validation and testing (Kahloot and Ekler, 2021).

```
X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, test_size=0.3, random_state=1)

print("Size of training X: ", X_train.shape)

X_validation, X_test, y_validation, y_test = train_test_split(X_tmp, y_tmp, test_size=0.5, random_state=1)

print("Size of validation X: ", X_validation.shape)
print("Size of training X: ", X_test.shape)
```

```
Size of training X:  (4424, 9)
Size of validation X:  (948, 9)
Size of training X:  (949, 9)
```

Figure 2.9

The initial SVM model is established using default parameters, functioning as a benchmark for gauging performance prior to refinement. This foundational model enables an initial evaluation of the SVM algorithm's effectiveness with the current dataset, offering a baseline against which the outcomes of further parameter optimization can be measured.

```
clf1 = SVC()

clf1 = clf1.fit(X_train, y_train)

y_pred1 = clf1.predict(X_validation)

print("Accuracy: ", metrics.accuracy_score(y_validation, y_pred1))
```

```
Accuracy:  0.9757383966244726
```

Figure 2.10

The initial model, set with default parameters, demonstrates a high degree of accuracy, registering a success rate in predictions of about 97.57% (**Figure 2.10**).

The revised version of the model employs a Linear kernel in place of the default Radial Basis Function (RBF) kernel. This adjustment is made to explore the performance implications of utilising a linear decision boundary as opposed to the default non-linear RBF kernel.

```
clf2 = SVC(kernel = 'linear')

clf2 = clf2.fit(X_train, y_train)

y_pred2 = clf2.predict(X_validation)
```

Figure 2.11

The final tuned iteration of the model utilises the **GridSearchCV** algorithm, which methodically searches for and determines the optimal set of hyperparameters. This approach ensures the enhancement of the model's performance by identifying the combination of hyperparameters that is most conducive to optimal results.

```
from sklearn.model_selection import GridSearchCV

param_grid = {'C': [0.1, 1, 10, 100], 'gamma': ['scale', 'auto', 0.1, 1, 10], 'kernel': ['linear', 'rbf', 'poly']}
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

Figure 2.12

The model is then trained using the best parameters identified by the algorithm.

```
# Create and train a new SVM model with the best parameters
tuned_clf = SVC(C=100, gamma=1, kernel='rbf')
tuned_clf.fit(X_train, y_train)

# Predict on the validation set
y_pred_tuned = tuned_clf.predict(X_validation)
```

Figure 2.13

Testing results

The initial model exhibited a commendable accuracy score, achieving approximately 97.57%. In this context, the scope for enhancement was relatively marginal. Nonetheless, through the adoption of a linear kernel, the augmented second iteration of the model achieved a modest increase in accuracy, registering a success prediction rate of around 97.68%.

```
print("Accuracy (Linear Kernel): ", metrics.accuracy_score(y_validation, y_pred2))
```

```
Accuracy (Linear Kernel):  0.9767932489451476
```

Figure 3.1

Further refinement was realised in the final version of the model, which attained a remarkably high accuracy of 99.37% in success rate prediction. This represents an improvement, exceeding the second model's performance by over 1.5% and surpassing the initial model's accuracy by nearly 2%.

```
print("Accuracy (Tuned Model):", metrics.accuracy_score(y_validation, y_pred_tuned))
```

```
Accuracy (Tuned Model): 0.9936708860759493
```

Figure 3.2

The performance of the model employing the linear kernel is slightly better than that of the default RBF kernel when evaluated on the validation dataset (**Figure 3.1-3.3**). This observation implies that a linear decision boundary could potentially be well-matched for the dataset at hand. It is important to highlight that the RBF kernel model, upon tuning, achieves superior accuracy. This suggests that the precise hyperparameter configuration pinpointed by the **GridSearchCV** technique contributes to an enhancement in the model's ability to generalise efficiently.

```
print("Accuracy (Initial Model): ", metrics.accuracy_score(y_validation, y_pred1))
print("Accuracy (Linear Kernel): ", metrics.accuracy_score(y_validation, y_pred2))
print("Accuracy (Tuned Model):", accuracy_tuned)
```

```
Accuracy (Initial Model):  0.9757383966244726
Accuracy (Linear Kernel):  0.9767932489451476
Accuracy (Tuned Model): 0.9936708860759493
```

Figure 3.3

Discussion

Data exploring and pre-processing

The process of constructing an effective machine learning model commences with thorough data pre-processing. This step is crucial as it directly impacts the subsequent performance of the algorithm. The transformation of attributes, specifically setting **'Record_ID'** as the index column and removing **'Bidder_ID'** was a necessary step to enhance model efficiency and remove any redundant data. The **'Auction_ID'** was discarded due to its nature as a unique identifier and to reduce potential noise in the dataset. Normalization, as applied to **'Auction_Duration'**, is crucial for SVM, known for its sensitivity to feature scales (Li and Liu, 2011). The successful use of **MinMaxScaler** ensures a consistent range, fostering more effective model training.

Model performance and evaluation

The performance evaluation of the models is paramount and accuracy scores provide a concise overview. The initial model achieved a commendable accuracy of 97.57%, showcasing SVM's inherent capabilities. The transition to a Linear kernel marginally improved accuracy, emphasising the dataset's compatibility with a simpler decision boundary. However, the most substantial performance leap occurred with the hyperparameter-tuned RBF kernel, achieving an accuracy of 99.37%.

To examine deeper the performance of the model, consideration of confusion matrices is essential, particularly given the observed class imbalance. The graphical representation in **Figure 2.7.1** highlights a significant disparity between normal and abnormal cases, necessitating a closer examination of true positives, true negatives, false positives, and false negatives.

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       860
           1       0.80      0.99      0.88        88

    accuracy                           0.98       948
   macro avg       0.90      0.98      0.93       948
weighted avg       0.98      0.98      0.98       948

Axes(0.125,0.11;0.775x0.77)
```

Figure 4.1

```
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       860
           1       0.80      1.00      0.89        88

    accuracy                           0.98       948
   macro avg       0.90      0.99      0.94       948
weighted avg       0.98      0.98      0.98       948

Axes(0.125,0.11;0.775x0.77)
```

Figure 4.2

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       860
           1       0.97      0.97      0.97        88

    accuracy                           0.99       948
   macro avg       0.98      0.98      0.98       948
weighted avg       0.99      0.99      0.99       948

Axes(0.125,0.11;0.775x0.77)
```

Figure 4.3

In evaluating the three classifications reports and confusion matrices, Matrix 3(**Figure 4.3**) representing the final tuned model, emerges as the standout performer, demonstrating flawless precision and recall for Class 0 and maintaining high accuracy for Class 1, resulting in an impressive overall accuracy of 0.99. Matrices 1(**Figure 4.1**) and 2(**Figure 4.2**)**,** representing the initial and second models, respectively, share similarities with a commendable accuracy of 0.98. However, Matrix 2 exhibits a slight enhancement in recall for Class 1, showcasing iterative improvements. Both Matrices 1 and 2 consistently show high precision for Class 0 and a balanced trade-off between precision and recall for Class 1.

Further analysing the model's performance, it is evident that across the three scenarios the model consistently demonstrates robust predictive capabilities. In the initial two instances (Matrices 1 and 2), the model impressively achieves high accuracy for the majority class (0 class), accompanied by minimal false positives. Even in the third scenario (Matrix 3), where there is a marginal increase in false positives for the 0 class and a slight uptick in false negatives for the 1 class, the overall performance remains robust, emphasising the iterative tuning process and the final model's effectiveness in making accurate predictions.

References:

Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L. and Lopez, A., (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. Neurocomputing, 408, pp.189-215. Available at: https://doi.org/10.1016/j.neucom.2019.10.118 [Accessed 20 December 2023].

Chakraborty, I. and Kosmopoulou, G., (2004). Auctions with shill bidding. Economic Theory, 24(2), pp.271-287. Available at: https://doi.org/10.1007/s00199-003-0423-y [Accessed 18 December 2023].

Chi, M., Feng, R. and Bruzzone, L., (2008). Classification of hyperspectral remote-sensing data with primal SVM for small-sized training dataset problem. Advances in Space Research, 41(11), pp.1793-1799. Available at: https://doi.org/10.1016/j.asr.2008.02.012 [Accessed 20 December 2023].

Dong, F., Shatz, S.M., Xu, H. and Majumdar, D., (2012). Price comparison: A reliable approach to identifying shill bidding in online auctions? Electronic Commerce Research and Applications, 11(2), pp.171-179. Available at: https://doi.org/10.1016/j.elerap.2011.12.003 [Accessed 18 December 2023].

Kahloot, K.M. and Ekler, P., 2021. Algorithmic Splitting: A Method for Dataset Preparation. IEEE Access, 9, pp.125229-125237. Available at: https://doi.org/10.1109/ACCESS.2021.3110745. [Accessed 4 January 2024].

Li, W. and Liu, Z. (2011) 'A method of SVM with Normalization in Intrusion Detection', Procedia Environmental Sciences, 11(Part A), pp. 256-262. Available at: https://doi.org/10.1016/j.proenv.2011.12.040 [Accessed 6 January 2024].

Muhammad Ali, P.J. and Faraj, R.H., (2014). Data Normalization and Standardization: A Technical Report. Machine Learning Technical Reports, 1(1), pp.1-6.

Ramyachitra, D. and Manikandan, P., (2014). Imbalanced dataset classification and solutions: a review. International Journal of Computing and Business Research (IJCBR), 5(4), pp.1-29.

Singh, D. and Singh, B., (2020). Investigating the impact of data normalization on classification performance. Applied Soft Computing, 97(Part B), p.105524. Available at: https://doi.org/10.1016/j.asoc.2019.105524 [Accessed 2 January 2024].

UCI Machine Learning Repository, 2020. Shill Bidding Dataset. [Dataset]. Available at: https://doi.org/10.24432/C5Z611 [Accessed 15 December 2023].