Online Shoppers Intentions

AI Report

By: Muhammad Islam and Tanvir Hassan Ahamed

-- **IN3062 Coursework** --

Table of content

Dataset and problem domain

The dataset was taken from Kaggle website which is about online shoppers intentions[1]. We used this dataset to create an AI model that will predict whether the shopper will purchase from an online website. We were curious about e-commerce and if online stores do get as much sales/revenue in comparison to their traffic. The problem domain is e-commerce since the dataset involves the behaviour of people purchasing on an online website.

This dataset consists of features with 12,330 entries as shown in the image. The target column that we were trying to predict was the “revenue” column where we were looking at how accurately we can predict if users of a website will make a purchase or not. Our target column is a boolean data type since it contains either a true or false value. The columns included 10 numerical and 8 categorical columns. There’s columns like “Bounce Rate” and “Exit Rates" which are said to be analytics taken by Google Analytics for each page of the website. It shows how much activity is on the website.
Our model is a classification considering we have to look at whether users of a website will purchase or not.

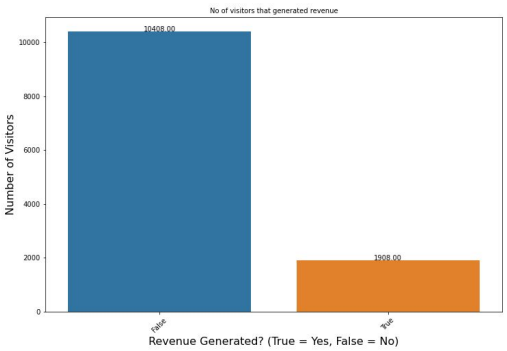
```
[10 rows x 18 columns]

---- DATASET INFO ----
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12316 non-null  float64
 1   Administrative_Duration  12316 non-null  float64
 2   Informational            12316 non-null  float64
 3   Informational_Duration   12316 non-null  float64
 4   ProductRelated           12316 non-null  float64
 5   ProductRelated_Duration  12316 non-null  float64
 6   BounceRates              12316 non-null  float64
 7   ExitRates                12316 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(10), int64(4), object(2)
memory usage: 1.5+ MB
None
```

In terms of questions and analytical tasks, we will be asking things like what is the most important factor that would affect our revenue, if the imbalance of our dataset

[1] Kaggle.com. 2020. *Online Shopper's Intention*. https://www.kaggle.com/roshansharma/online-shoppers-intention [Accessed 7 December 2020].

would cause a negative effect on our results and also how we can make it more balanced and see if our results change in a positive or negative way from this change. Our tasks will be to perform tests on different classification algorithms and select the best one from all of these. Then we will perform cross validation on the best model and extract optimal parameters of this model. We can also look at other things like which operating system is most used when purchasing online which we can put in a graph. The outputs of our model testing is likely to be RandomForest as this model tends to give good results without doing tuning, however we will still see if any other models prevail. We will also run a separate test on RandomForest from 10-100 by itself to see if the scores line up with the grid search which we will do later with the rest of the models including RandomForest. This is to prove that our method is correct and doesn't provide false results.

Data Analysing



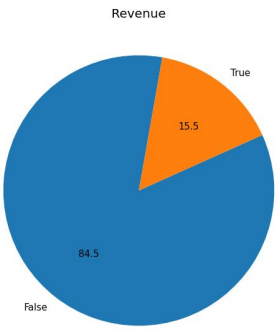

```
---------- SUM THE NULL ELEMENTS ---
Administrative             14
Administrative_Duration    14
Informational              14
Informational_Duration     14
ProductRelated             14
ProductRelated_Duration    14
BounceRates                14
ExitRates                  14
PageValues                  0
SpecialDay                  0
Month                       0
OperatingSystems            0
Browser                     0
Region                      0
TrafficType                 0
VisitorType                 0
Weekend                     0
Revenue                     0
dtype: int64
```

The dataset contains 14 rows with missing values and since they all belong to the false class (majority class) we can simply remove the rows without the need of filling or transforming the missing values. This removal of data will not hinder the performance of the Model since the missing data percentage (0.11 %) is very low and belongs to the majority class.

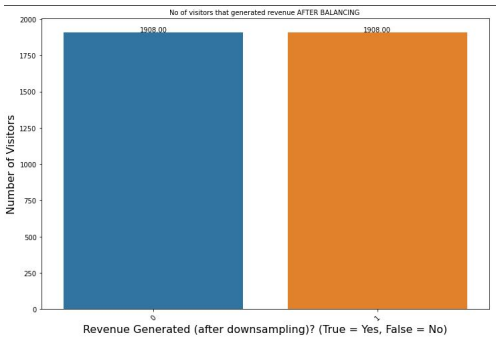
We managed to cope with the missing data by removing null elements completely. This was achieved by using **df.dropna(axis = 0)** which looked over all the rows (hence axis = 0) and removed any null values. After we did this, we managed to eliminate our null values as shown in the image on the right.

```
---- ONCE THE MISSING DATA ARE REMOVED ----
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

We applied many techniques to understand our dataset such as by using lines like **df.info** to show our different columns and the data types as well as using **df.describe** to see a description of our various columns with their count, mean, the standard deviation, min and max. In addition, we used various graphs such as a pie and bar chart which shows the percentage of "false" and "true" values from our target column with the exact number. This allowed us to see that our dataset was **imbalanced** so using this information, we managed to balance it later on. We also had other various graphs which can be found in our code output such as bar charts to show the different visitor types in comparison to our revenue, a heat map, and also the browser counts.



From the dataset information we retrieved by using **df.info** (shown in the first image), we notice that the majority of the columns were floats and the rest were integers. Our target column was a boolean which was obvious considering it consisted of "TRUE" and "FALSE" values (which we later converted to 1's and 0's to keep it numerical).



With our balancing, we omitted a few data when we downsampled to make it more equal with the true values. However, this didn't hinder our results as much as we would have thought. We also omitted the null values as mentioned above.

In terms of encoding, we used the sklearn preprocessing library "LabelEncoder" to convert our categorical variables into a label encoder. This also helped us normalise labels and we were able to transform the non-numerical labels to numerical to be the same as the other columns.

We balanced our dataset by using the sklearn.util library called resample. The reason why we did this is because before balancing, we were getting a decent accuracy score but our recall score was very low, so was f1 score. This indicated that the dataset was unbalanced, the amount of falses completely outweighed the true values. Using the resample library, we downsampled the false values equal to the same as trues (1908). By doing this, we managed to get it 50/50 which was balanced.

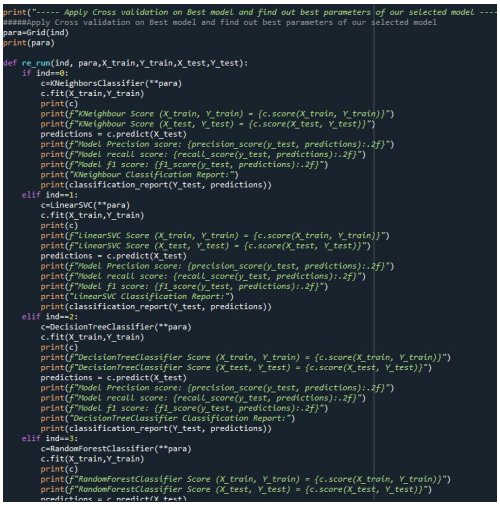
```
df_majority = df[df.Revenue==0]
df_minority = df[df.Revenue==1]
df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples= 1908,
                                   random_state=123)
```

Models used

We used a large variety of models such as KNeighborsClassifier, LinearSVC, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier, XGBClassifier, MLPClassifier, LogisticRegression and finally Sequential to build a neural network. There were various pros and cons of using these models. For example, with random forest, it was easy to use as it works with both classification and regression problems. In addition, random forest can handle large data sets with higher dimensionality. The downside, however, is the fact that it feels like we have little control of the parameters since the model provides good results regardless and tuning can only make small differences (which will be shown later). With Sequential search for building a neural network, the benefits was that it's quite flexible to use as it can be used for both classification and regression problems plus any numerical data can be used in the model which was the case for our dataset. Also once trained, it can predict data pretty fast. However, the downside is that it was quite time consuming to us to run and was computationally expensive as the power of our CPU/GPU influenced the speed of training.

```
print("")
print("-------------------------------------------------------------------------")
print("--------------  TESTING DIFFERENT MODELS AFTER BALANCING  --------------")
# run a pipeline in a loop for each classifier, along with its validation scores
for classifier in classifiers:
    pipe = Pipeline(steps=[('scaler', StandardScaler()),
                           ('classifier', classifier)])
    pipe.fit(X_train, y_train)
    print(classifier)
    S=pipe.score(X_test, y_test)
    L.append(S)
    print(f'MODEL SCORE: {pipe.score(X_test, y_test)}')
    predictions = pipe.predict(X_test)
    print(f"Model Precision score: {precision_score(y_test, predictions):.2f}")
    print(f"Model recall score: {recall_score(y_test, predictions):.2f}")
    print(f"Model f1 score: {f1_score(y_test, predictions):.2f}")
    print(f"CLASSIFICATION REPORT FOR {classifier}")
    print(classification_report(y_test, predictions))
    print("-------------------------------------------------------------------")
    print("")

## Recall score and f1 score is improved significantly on the balanced dataset

########## IMPROVING THE MODELS ############

###Select index of Best model
ind=L.index(max(L))

def Grid(ind):
    if ind==0:
        param_grid = dict(n_neighbors=[1,2,3,4,5,6,7,8])
        c1_B = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid,cv=5)
        c1_B.fit(X_train, y_train)
        print(c1_B,c1_B.best_params_)
        return c1_B.best_params_

    elif ind==1:
        c2_B = GridSearchCV(LinearSVC(),param_grid={"C": [0.001, 0.01, 0.1, 1.0, 10.0]})
        c2_B.fit(X_train,y_train)
        print(c2_B,c2_B.best_params_)
        return c2_B.best_params_

    elif ind==2:
        param_grid = {
            'criterion':['gini','entropy'],
            'max_depth':[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]
            }
        c3_B = GridSearchCV(DecisionTreeClassifier(), param_grid=param_grid, cv=5)
        c3_B.fit(X_train,y_train)
        print(c3_B,c3_B.best_params_)
        return c3_B.best_params_

    elif ind==3:
        param_grid={
```

We used the sklearn train_test_split to get our X and y data such as X_train, X_test, y_train and y_test.
We decided to use a test_size of 0.30 so we can allow 70% of our data to be the training size and we test with 30%. Since our unbalanced dataset had over 12,000 entries, this was more than enough to get good results. We expect RandomForest to get the best results since it works well for most datasets without the need for tuning, so having it as our best model won't necessarily be a surprise to us.



```
print("----- Apply Cross validation on Best model and find out best parameters of our selected model ----")
#####Apply Cross validation on Best model and find out best parameters of our selected model
para=Grid(ind)
print(para)

def re_run(ind, para,X_train,Y_train,X_test,Y_test):
    if ind==0:
        c=KNeighborsClassifier(**para)
        c.fit(X_train,Y_train)
        print(c)
        print(f"KNeighbour Score (X_train, Y_train) = {c.score(X_train, Y_train)}")
        print(f"KNeighbour Score (X_test, Y_test) = {c.score(X_test, Y_test)}")
        predictions = c.predict(X_test)
        print(f"Model Precision score: {precision_score(y_test, predictions):.2f}")
        print(f"Model recall score: {recall_score(y_test, predictions):.2f}")
        print(f"Model f1 score: {f1_score(y_test, predictions):.2f}")
        print("KNeighbour Classification Report:")
        print(classification_report(Y_test, predictions))
    elif ind==1:
        c=LinearSVC(**para)
        c.fit(X_train,Y_train)
        print(c)
        print(f"LinearSVC Score (X_train, Y_train) = {c.score(X_train, Y_train)}")
        print(f"LinearSVC Score (X_test, Y_test) = {c.score(X_test, Y_test)}")
        predictions = c.predict(X_test)
        print(f"Model Precision score: {precision_score(y_test, predictions):.2f}")
        print(f"Model recall score: {recall_score(y_test, predictions):.2f}")
        print(f"Model f1 score: {f1_score(y_test, predictions):.2f}")
        print("LinearSVC Classification Report:")
        print(classification_report(Y_test, predictions))
    elif ind==2:
        c=DecisionTreeClassifier(**para)
        c.fit(X_train,Y_train)
        print(c)
        print(f"DecisionTreeClassifier Score (X_train, Y_train) = {c.score(X_train, Y_train)}")
        print(f"DecisionTreeClassifier Score (X_test, Y_test) = {c.score(X_test, Y_test)}")
        predictions = c.predict(X_test)
        print(f"Model Precision score: {precision_score(y_test, predictions):.2f}")
        print(f"Model recall score: {recall_score(y_test, predictions):.2f}")
        print(f"Model f1 score: {f1_score(y_test, predictions):.2f}")
        print("DecisionTreeClassifier Classification Report:")
        print(classification_report(Y_test, predictions))
    elif ind==3:
        c=RandomForestClassifier(**para)
        c.fit(X_train,Y_train)
        print(c)
        print(f"RandomForestClassifier Score (X_train, Y_train) = {c.score(X_train, Y_train)}")
        print(f"RandomForestClassifier Score (X_test, Y_test) = {c.score(X_test, Y_test)}")
        predictions = c.predict(X_test)
```

To test our models, we created a method that uses the sklearn Pipeline library to test multiple models and then store their score into an array called “L”. I then store the index of the best value from the “L” array in a variable called “ind”. This ind value is then used in another method we created called “Grid” which uses GridSearchCV to select the specific model that was the best and then train it with the best parameters. The image on the right isn’t the full method but you can see how it uses an if statement to select the best index and then apply GridSearchCV to find the best model to train.

After doing so, we ran the (left) image’s method by writing “re_run(ind, para,X_train,y_train,X_test,y_test)”. This method outputs the best models score, the precision, recall and f1 score as well as the classification report. In our code, we run it twice but the second without “random forest classifier” because “random classifier” always seems to return the best results even after tuning.

We also ran separate tests before this that tests RandomForestClassifier to test the various n_estimators from 10 to 100. We did this before and after balancing to see the differences in results and also to test how effective randomforest is before doing a large scale testing like the methods above.

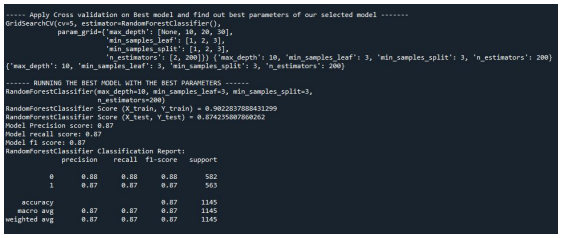
Results

In terms of the result, when we did not balance the dataset, we got the following scores:

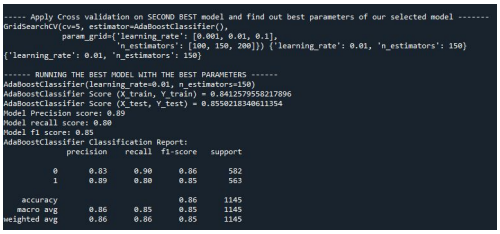
Models	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC over ROC**
Sequential	89.3%	N/A	N/A	N/A	N/A
KNeighborClassifier	87%	69%	36%	48%	79%
LinearSVC	88%	81%	33%	47%	N/A
DecisionTreeClassifier	85%	52%	55%	54%	73%
RandomForestClassifier - I also ran a separate test on this with 10-100 estimators which is shown in the graph in the evaluation section	90%	74%	58%	65%	92%
AdaBoostClassifier	89.1%	68%	55%	61%	92%
XGBClassifier	89.9%	72%	56%	63%	92%
MLPClassifier	89.8%	73%	55%	63%	90%
LogisticRegression	88.7%	78%	37%	50%	89%

After balancing we got these results below. We can see how all the results caused a significant drop in accuracy but increase in precision, recall and f1 score.

Models	Accuracy	Precision	Recall	F1	AUC over ROC
Sequential	83.6%	N/A	N/A	N/A	N/A
KNeighborClassifier	77.7%	77%	78%	77%	86%
LinearSVC	81.8%	89%	72%	80%	N/A
DecisionTreeClassifier	80.6%	79%	80%	80%	81%
RandomForestClassifier - I also ran a separate test on this with 10-100 estimators which is shown in the graph in the evaluation section	86.3%	86%	86%	86%	93%
AdaBoostClassifier	85.5%	89%	89%	85%	92%
XGBClassifier	84.8%	85%	85%	85%	92%
MLPClassifier	83.9%	87%	81%	84%	92%
LogisticRegression	82.4%	86%	74%	81%	90%



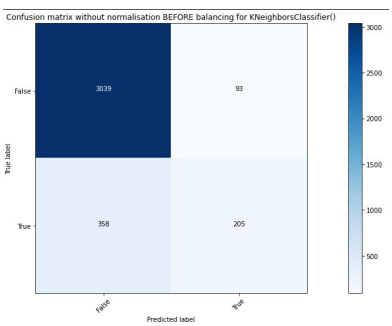
After running the tests on our balanced datasets, we ran two cross validation tests to select the best and second best model and tuning both. When the program ran the tests, it came out that, unsurprisingly, RandomForestClassifier was the best model. It then did tuning to the model with its best parameters using the parameters we defined in the Grid method which I have shown above. It used the tuning parameters: max_depth=10, min_samples_leaf=3, min_samples_split=3 and n_estimators=200 The tuning ended up **improving** random forest slightly. It caused the score to increase to 87.4% from the before 86.3%. This is an improvement of 1.1%!

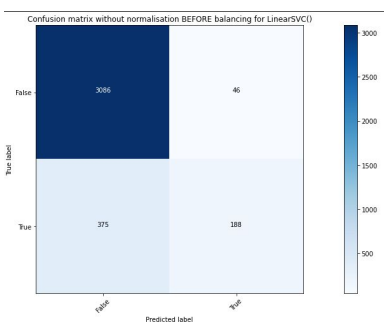


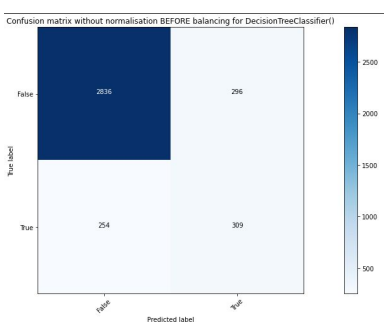
We also mentioned how it ran a second test. With the second test, it leaves out Random Forest. This ended up making AdaBoostClassifier the second best. It used the tuning parameters of: learning_rate of 0.01 and n_estimators of 150 as shown in the image. However after we did the tuning, it did not change the results. The results were 85.5% which is the same as the above 85.5%. This could indicate that AdaBoost was at its best already and tuning any further wouldn't have done anything.

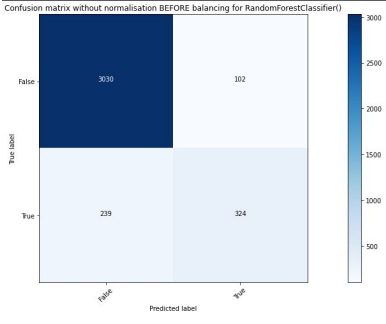
We also looked at confusion matrices before and after balancing for each model:

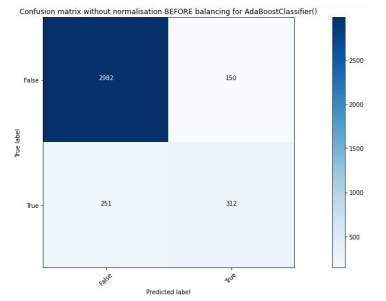
Before Balancing:

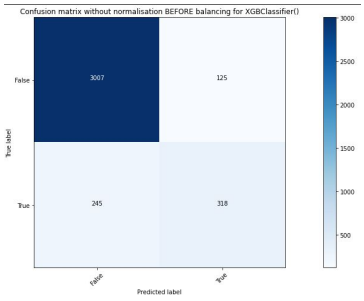


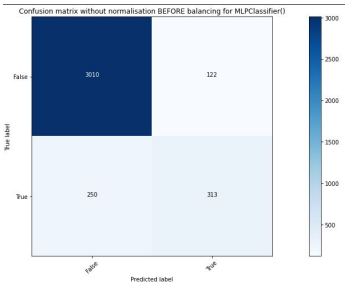


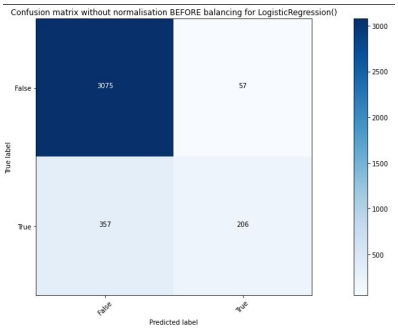




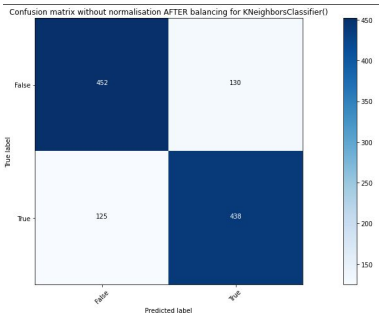


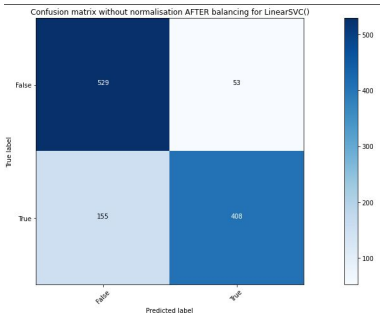


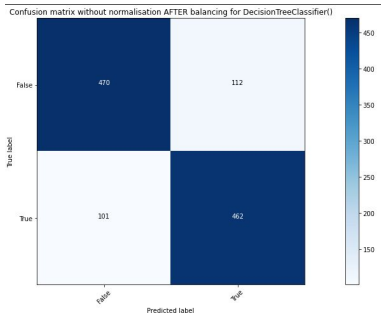


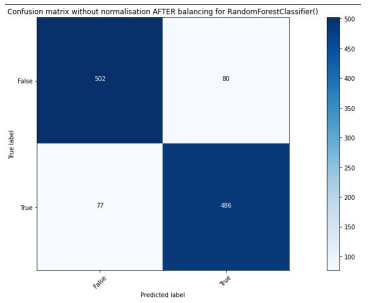


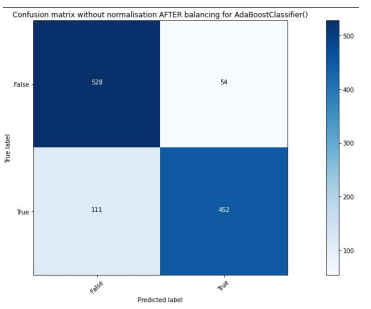
After balancing:

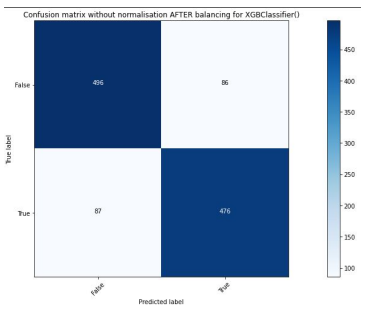


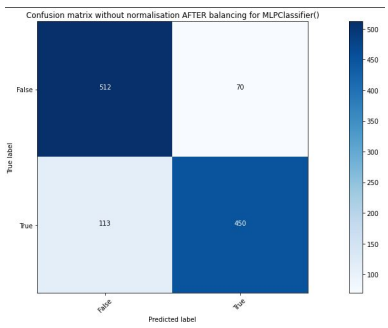


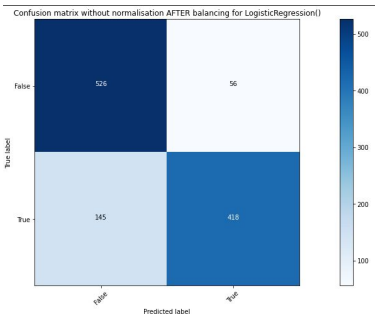




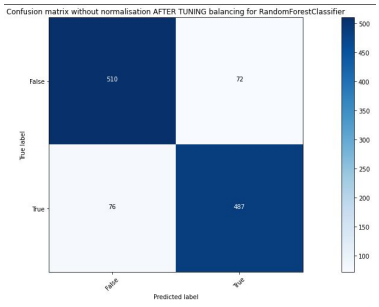








From these changes, we can see that balancing the dataset also makes the confusion matrices more balanced. We can see that the true negative values and the true positive values are more balanced, hence the darker colours diagonally. In the unbalanced confusion matrices, the models predicted the true negative (false revenue) values very well but were failing to predict the true positive (true revenue). This isn't a surprise since our dataset was extremely imbalanced with over 10,000 false values compared to 1908 true values.



After tuning our RandomForestClassifier, there was only a small change in our confusion matrix. This seemed to have reflected the percentage change in the initial tuning we did.

Finally, we also ran separate tuning for every other model apart from RandomForest and AdaBoost (since these two were the top two). We got these following results: KNeighbourClassifier: 72%, LinearSVC: 71.7%, DecisionTreeClassifier: 86.1%, XGBClassifier: 86.8%, MLPClassifier: 81% and LogisticRegression: 81.3%. Despite some improvements such as DecisionTree going from 80.6% to 86.1% and XGBClassifier going from 84.6% to 86.8%, we found it weird that KNeighborClassifier, LinearSVC, MLPClassifier and

LogisticRegression did not improve, rather it went down. We did try many different parameters which did not change our outcome.

Evaluation

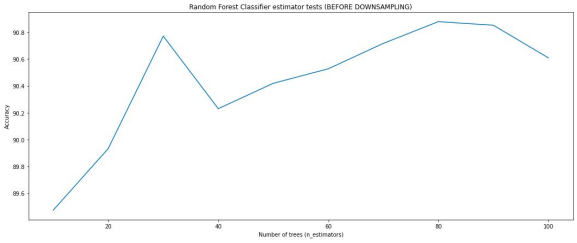
Based on the results, it appears there were a variety of outcomes. When we balanced the dataset, it was apparent that the results accuracies decreased significantly. This is proof that the previous unbalanced dataset was misleading because it provided false results which caused our results to look higher than normal. Since all the results were a reduction, we calculated the percentage decrease of each result:

- Sequential() = 6.3%
- LinearSVC = 7%
- RandomForestClassifier = 4.1%
- XGBClassifier = 5.6%
- LogisticRegression = 7.1%
- KNeighborsClassifier = 10.6%
- DecisionTreeClassifier = 5.1%
- AdaBoostClassifier = 4%
- MLPClassifier = 6.5%

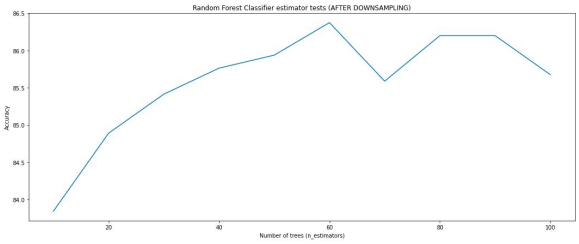
Based on these results, it's apparent that KNeighborClassifier had the most impact by the balancing since it caused a 10.6% decrease followed by LogisticRegression with 7.1% and LinearSVC with 7%.

Although the accuracy of the models lowered, the precision, recall and f1 score significantly increased. All the models' accuracy score decreased by average of 5.6% but the precision score increased by 13% , recall score by 32.6% and f1 score by 25.88 % . The reason we had higher accuracy on the unbalanced dataset is because the dataset mostly contained false revenue, thus the models were able to predict most of the false revenues. But looking at the confusion matrix, precision, recall score and f1 score we can see that the models were mispredicting the true revenue. The precision score can indicate out of all the (total) True revenues predicted (TP + FP) by the model how many were actually the True revenue (TP), in our case high precision on balanced dataset shows that the models' correctness on predicting true revenue significantly increased. The recall score indicates how many of the actual true revenues were predicted by the models, high recalls shows that the models were able to predict most of the actual true revenues. And finally, the f1 score takes both precision and recall scores into consideration, high f1 score shows the models can predict both true and false revenues equally without being biased. Therefore, we can say that the model trained on the balanced dataset is more accurate despite losing some accuracy score.

In addition to these scores, we also measured the Area Under the Receiver Operating Characteristics score. When we tested it with LinearSVC, it did not seem to work as it did not recognise a particular line of code which the other models do recognise. In general, it seemed like only a few had changed in the score. So for example, KNeighborClassifier went from 79% to 86% and DecisionTreeClassifier went from 73% to 81%. The rest seemed to have little to no effect. This measure shows how much these models are able to distinguish between classes. So the higher the AUC score, the better the model is predicting 0 values as 0 and 1 values as 1. The higher AUC scores show that the model is better at distinguishing between customers who



purchase and don't purchase. It seems like RandomForestClassifier is best at distinguishing which seems to be a common trend among other score testing,



Also based on the graph of the RandomForestClassifier testing we did earlier with 10-100 n_estimators, it can be seen that after balancing, the accuracy did not go anymore past 86%. Meanwhile the unbalanced version was hovering around 90-91% accuracy. As aforementioned, the decrease in accuracy isn't necessarily a bad thing as we realised unbalanced datasets could produce inaccurate and misleading information. So balancing our dataset made our information more accurate. This was better for our f1, recall and precision score.

Conclusion

We had various difficulties and problems when working with the dataset. One of which was a constant issue that kept outputting "ValueError: Input contains NaN, infinity or a value too large for dtype('float32')". We were confused on why this was happening but we managed to solve it by dropping all null values. We used the code "df = df.dropna(axis = 0)" which removed all null values in our rows (axis = 0) and also solved that problem. We also had another issue where our balanced X value would keep returning "ValueError: Input X must be non-negative". This confused us since this error did not show up in our previous X and y value for the unbalanced version of the dataset. We then inspected the dataset csv file and noticed a few "-1" values. We managed to fix this by using "X = X.clip(lower = 0)" which replaced all values below -1 with a 0 value. This was found on stackoverflow which we then adapted to work with our own dataset.

From this whole experience, we have learnt many things. One thing we have learnt were many ways to train a model. We realised that we could train models in different ways. We used methods like a for loop that loops over the different parameters and outputs the result. We also figured out we could use GridSearch to train different models and tune at the same time. Another thing we have learnt was how to analyse our dataset properly. For example, we realised pandas has a ton of analyse techniques such as **df.describe, df.info** which allows us to look at key information about our dataset. These provided us useful information which includes seeing if columns contain a null and the different data types of each column. The final thing we have learnt were balancing techniques. Once we did further research, we noticed there are plenty of resources available to balance our dataset such as SMOTE and sklearn's resample library. In addition to learning balancing techniques, we learnt how to use it properly with our dataset which helped us with our analysis since we had more false values than true values.

In terms of future work, we hope to improve our knowledge of machine learning since it is a fun topic to learn plus it would help our employment opportunities. Many jobs nowadays require some knowledge of machine learning for jobs like data science and this module was an inspiration to start learning in our free time.

References

- In line 134, we used a function called **heatmap**, this was inspired by a user on github who had this code:
    - **Zaric, D., 2018. *Step_1.Py*. [online] GitHub. Available at: <https://gist.github.com/drazenz/99e9a0a2b29a275170740eff0e215e4b> [Accessed 9 December 2020].**
- We also got inspiration from this article to get help to balance our dataset when using the resample code:
    - **EliteDataScience. 2020. *How To Handle Imbalanced Classes In Machine Learning*. [online] Available at: <https://elitedatascience.com/imbalanced-classes> [Accessed 9 December 2020].**
- We based our confusion matrix based on this documentation from scikit-learn and adapted it to work with our work.
    - **Scikit-learn.org. 2020. *Confusion Matrix — Scikit-Learn 0.18.2 Documentation*. [online] Available at: <https://scikit-learn.org/0.18/auto_examples/model_selection/plot_confusion_matrix.html> [Accessed 12 December 2020].**