Terry Zou
Hsin Ruei Lee

Online Shopper's Intention Final Report

Background Information:

In the ecommerce industry, recognizing a customer's purchasing intention is crucial in performing precision marketing, optimizing website performance, and increasing total sales. For this project, our goal is to predict whether a shopper will make a purchase after entering the website by looking at what factors yield a customer to finalize a transaction online. To do this, we will use various classification models such as decision tree, random forest, and support vector machines. Bagging methods can also be tested out in order to improve robustness and deal with the imbalance of data.

Our Data:

We will be using the "Online Shoppers Purchasing Intention Dataset" obtained through the UC Irvine Machine Learning Repository. The creators of this dataset, C. Sakar and Yomi Kastro, inputted feature vectors belonging to 12,330 sessions. Each session in the dataset belongs to a different user in a 1-year period to avoid any tendency to a specific campaign, special day, user profile, or period.

There are 18 variables in this dataset, with 10 numerical and 8 categorical attributes. This is an imbalanced dataset with 10422 'False' response variable and 1908 'True' response variable.

Numerical Variables:

- Administrative: Number of pages visited by the visitor about account management
- Administrative Duration: Total amount of time (in seconds) spent by visitor on Administrative type pages
- Informational: Number of pages visited by the visitor about Web site, communication and address information of the e-commerce site
- Informational Duration: Total amount of time (in seconds) spent by visitor on informational type pages
- Product Related: Number of pages visited by visitor about product related pages
- Product Related Duration: Total amount of time (in seconds) spent by visitor on product related type pages

- Bounce rate: Average percentage of visitors that entered the website but left (“bounced”) without triggering any requests from the analytics server
- Exit Rate: Average percentage of visitors that exit the website to go to a different website
- Page Value: average value of a website that a user visited before completing an e-commerce transaction
- Special Day: the closeness between the site visiting time and a special occasion (e.g. Mother’s Day, Christmas) in which it resulted in a translation

Categorical Variables:

- Month: Month of the visit
- Operating System: Operating system of the visitor
- Browser: The type of browser used by the visitor
- Region: The geographic region from which the visitor started its session
- Traffic Type: Describes the traffic source by which the visitor arrived at the website from (e.g. banner, SMS, direct, email)
- Visitor Type: Status of visitor (returning, new, or other)
- Weekend: Whether the data of the visit is weekend
- Revenue: Whether the visitor made a purchase or not

Preliminary Analysis:

Exploratory Data Analysis

```
 Administrative   Administrative_Duration Informational     Informational_Duration
 Min.   : 0.000   Min.   :   0.00         Min.   : 0.0000   Min.   :   0.00
 1st Qu.: 0.000   1st Qu.:   0.00         1st Qu.: 0.0000   1st Qu.:   0.00
 Median : 1.000   Median :   7.00         Median : 0.0000   Median :   0.00
 Mean   : 2.297   Mean   :  80.04         Mean   : 0.4957   Mean   :  34.14
 3rd Qu.: 4.000   3rd Qu.:  92.04         3rd Qu.: 0.0000   3rd Qu.:   0.00
 Max.   :27.000   Max.   :2720.50         Max.   :24.0000   Max.   :2256.92
 ProductRelated   ProductRelated_Duration  BounceRates        ExitRates
 Min.   :  0.00   Min.   :    0.0         Min.   :0.000000   Min.   :0.00000
 1st Qu.:  7.00   1st Qu.:  183.6         1st Qu.:0.000000   1st Qu.:0.01429
 Median : 18.00   Median :  597.3         Median :0.003203   Median :0.02516
 Mean   : 31.61   Mean   : 1185.5         Mean   :0.022361   Mean   :0.04320
 3rd Qu.: 37.00   3rd Qu.: 1451.8         3rd Qu.:0.016933   3rd Qu.:0.05000
 Max.   :705.00   Max.   :43171.2         Max.   :0.200000   Max.   :0.20000
   PageValues        SpecialDay        Month           OperatingSystems
 Min.   :  0.000   Min.   :0.00000   Length:9864        Min.   :1.000
 1st Qu.:  0.000   1st Qu.:0.00000   Class :character   1st Qu.:2.000
 Median :  0.000   Median :0.00000   Mode  :character   Median :2.000
 Mean   :  5.962   Mean   :0.06066                      Mean   :2.129
 3rd Qu.:  0.000   3rd Qu.:0.00000                      3rd Qu.:3.000
 Max.   :361.764   Max.   :1.00000                      Max.   :8.000
```

```
    Browser           Region       TrafficType     VisitorType
 Min.   : 1.000   Min.   :1.000   Min.   : 1.000   Length:9864
 1st Qu.: 2.000   1st Qu.:1.000   1st Qu.: 2.000   Class :character
 Median : 2.000   Median :3.000   Median : 2.000   Mode  :character
 Mean   : 2.356   Mean   :3.168   Mean   : 4.093
 3rd Qu.: 2.000   3rd Qu.:4.000   3rd Qu.: 4.000
 Max.   :13.000   Max.   :9.000   Max.   :20.000
    Weekend          Revenue
 Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.0000   1st Qu.:0.0000
 Median :0.0000   Median :0.0000
 Mean   :0.2306   Mean   :0.1565
 3rd Qu.:0.0000   3rd Qu.:0.0000
 Max.   :1.0000   Max.   :1.0000
```

Figure 1: Summary Statistics of the 80% Training Set

We split the Shoppers dataset into 80% for training and 20% for testing.

Administrative	Administrative_Duration	Informational

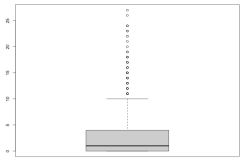
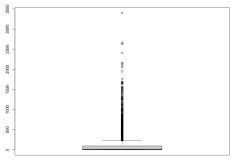
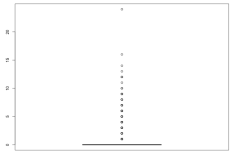
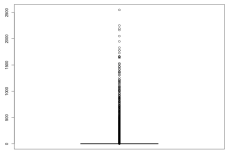
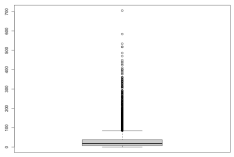
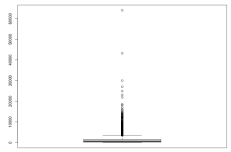
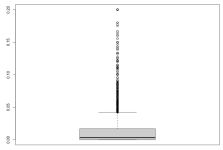
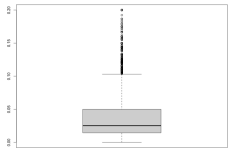
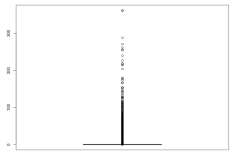
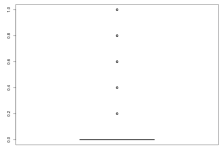
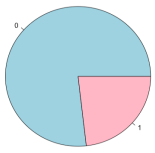
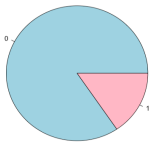
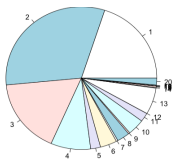
		
Informational_Duration	ProductRelated	ProductRelated_Duration
		
BounceRates	ExitRates	PageValues
		
SpecialDay		
		

Figure 2: Boxplots of Numerical Variables

Figure 2 displays the numerical variable's boxplots. Variables 'Administrative_Duration', 'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates', and 'Page Values' hold the most outliers clusters around the maximum.

Weekend	Revenue	TrafficType
		
OperatingSystems	Region	Browser

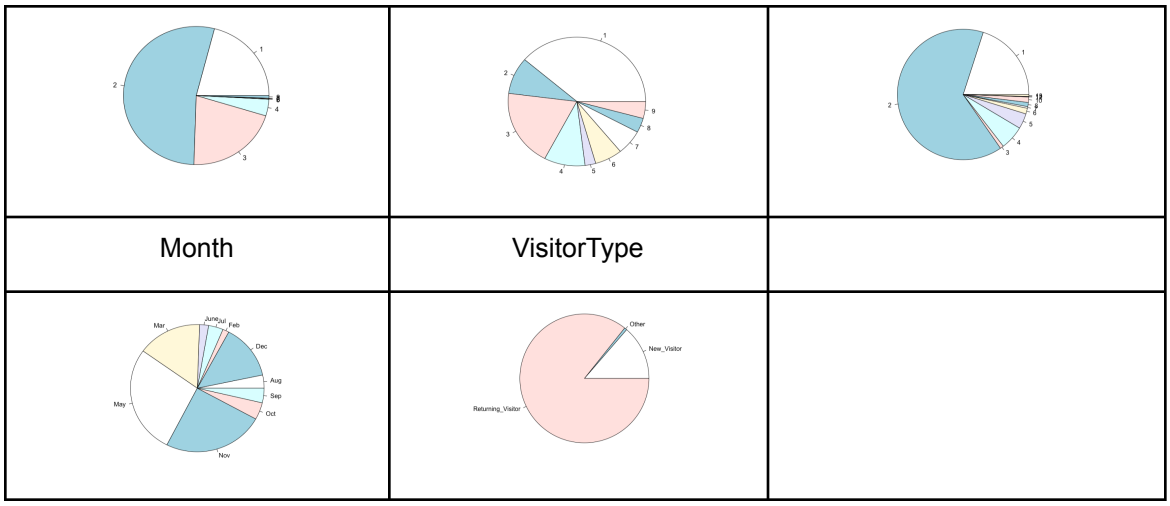


Figure 3: Pie Charts for Categorical Variables

We have plotted the pie charts for the categorical variables shown in Figure 3. Starting with the 'Weekend' variable, we know that users don't normally visit the site on weekends. And most of the time, the visitor does not finalize their transaction on the website. We see that traffic type #1 and #2 are the most effective ways of attracting more traffic for the website. Operating system #2 is most commonly used amongst the visitors. Region #1 is the area in which most of the users are located. Many users prefer to use browser #2 when visiting the website. May, November, March, and December are the months that attract the highest number of visitors. And lastly, a majority of the visitors for the e-commerce site are returning visitors.

Dealing with Imbalanced Data

	True	False
True	x	negativeWeight
False	positiveWeight	x

```
##train model with weights
positiveWeight = 1.0 / (nrow(subset(shoppers_train, Revenue == TRUE)) / nrow(shoppers_train))
negativeWeight = 1.0 / (nrow(subset(shoppers_train, Revenue != TRUE)) / nrow(shoppers_train))
# Create a weight vector based on the class imbalance
weights <- ifelse(shoppers_train$Revenue == TRUE, positiveWeight, negativeWeight)
```

Figure 4: Calculation of Weights Applied

To handle the imbalance of False values against True values within our response variable, we apply a weight to both false positive and false negative calculations. Instead of having a set weight, we use the ratio of 1/ percentage of Revenue that equal true or false. The positive weight (consisting of less true values, therefore a higher ratio) gets applied to the false negative value. And the negative weight (consisting of more false values, therefore a lower ratio) will be calculated with the false positive value.

Decision Tree

For the initial model, we chose the decision tree model because the model allows for simple interpretations and visualizations, also it can capture nonlinear relationships that logistic regression can not. When training the model, we apply the weight above to deal with the imbalance data.

a. Pruning

We started to build the model with a low complexity parameter (cp = 0.001) to create a large tree (Figure 5). To find the optimal complexity parameter, we plot out the cp figure and find the optimal cp value to be 0.0045.

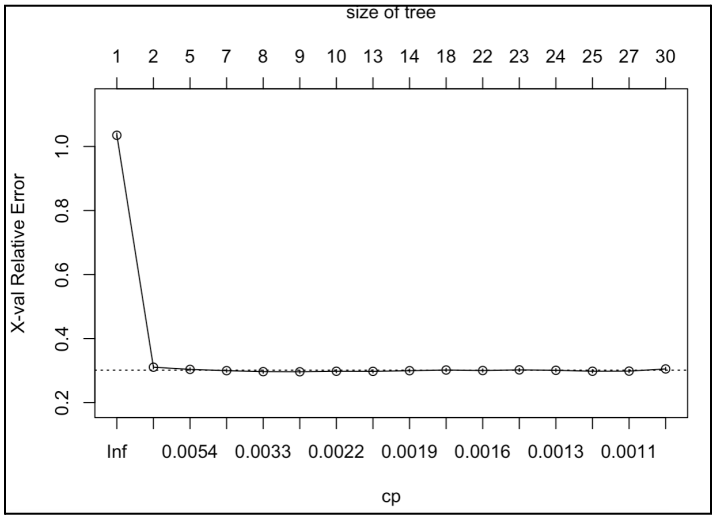


Figure 5: Complexity Parameter Plot

The optimal model is visualized below (Figure 6). The tree shows that variables like PageValues, Month, TrafficType, ProductRelated, and Administrative are decisive for the classification problem. Administrative and ProductRelated represent the number of different types of pages visited by the visitor in that session, indicating that the type of pages are crucial to whether a shopper makes a purchase or not.

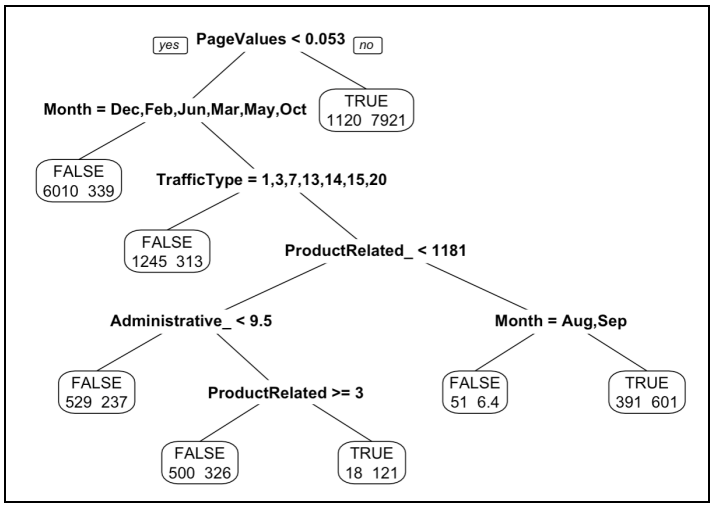


Figure 6: Optimal Tree Visualization

b. In-sample Performance

The in-sample misclassification matrix is below (Figure 7). And we examine the model performance by precision, accuracy, recall, and F1 score (Figure 7.1). The decision tree we built has a precision of 0.845, accuracy of 0.849, recall of 0.974, and F1 score of 0.905.

	Pred	
Truth	FALSE	TRUE
FALSE	7031	1290
TRUE	191	1352

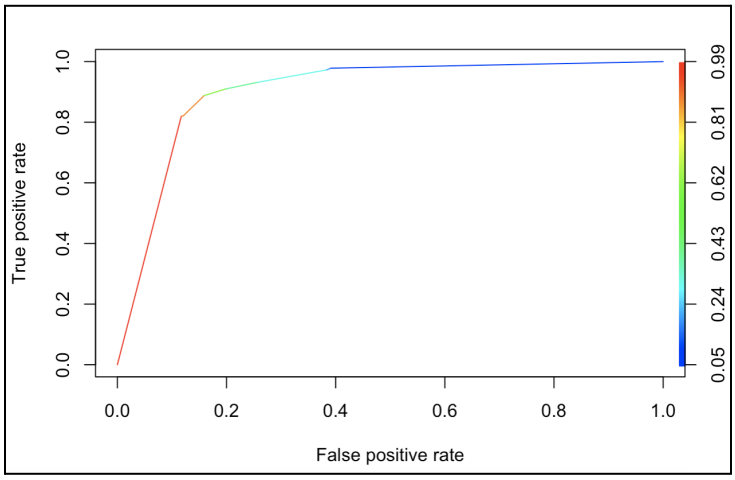
Figure 7: In-sample Misclassification Matrix

Precision	0.8449706
Accuracy	0.8498581
Recall	0.973553
F1 Score	0.9047159

Figure 7.1: In-sample Performance Metrics

c. Out-of-sample Performance

Below is the ROC curve of the model on the test set (figure 8), and the AUC is reported to be **0.903**, meaning that there is a 90.3% chance that the model will be able to distinguish between the positive class and the negative class.




```
> slot(performance(pred, "auc"), "y.values")[[1]]
[1] 0.9030631
```

Figure 8: ROC Curve and AUC

The out-of-sample misclassification matrix is below (Figure 10). The precision is 0.841, accuracy is 0.848, recall is 0.977, F1 score is 0.904. Comparing the 4 out-of-sample performance metrics to those of the in-sample performance, we can see that the out-of-sample performance is just slightly worse than that of in-sample, which is as expected. Therefore, we can say that the model is robust.

	Pred	
Truth	FALSE	TRUE
FALSE	1767	334
TRUE	41	324

Figure 9: Out-of-sample Misclassification Matrix

	In-sample	Out-of-sample
Precision	0.8449706	0.8410281
Accuracy	0.8498581	0.8479319
Recall	0.973553	0.977323

F1 Score	0.9047159	0.9040675

Figure 10: Out-of-sample Performance Metrics

Random Forest

We chose to apply a random forest model to our data because it can often achieve high accuracy in classification tasks. Aside from producing more stable results, it can handle large datasets and uses bagging techniques that make it less prone to overfitting.

a. In-Sample Performance

```
Call:
 randomForest(formula = Revenue ~ ., data = shoppers_train, proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 9.54%
Confusion matrix:
      False True class.error
False  7986  342  0.04106628
True    599  937  0.38997396
```

Figure 11: Random Forest Base Model with Default Parameters (ntree = 500)

We start off with building a random forest model using default parameters. Here we see that it created 500 trees with 4 variables at each split (Figure 11). The out-of-bag rate is 9.54% which tells us 90.46% of the OOB samples were correctly classified by the random forest. The confusion matrix given by the model demonstrates that there were 599 false positive values, indicating that we predicted a customer would make a purchase even though they didn't. And the false negative value of 342 represents the customers who didn't make a purchase, but were falsely counted as making a purchase.

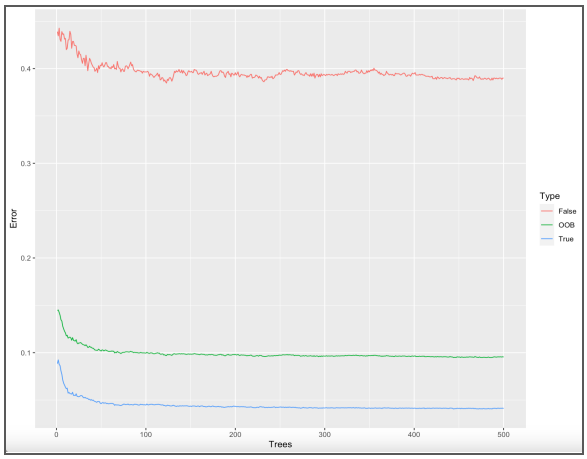


Figure 12: Plot of Error Rate for Base Model

In Figure 12, we are given the plot of error rate for our base model (default parameters). Looking at the graph, we wonder if the error rate decreases with more trees tested within the model.

```
Call:
 randomForest(formula = Revenue ~ ., data = shoppers_train, ntree = 1000,
RUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 4

        OOB estimate of  error rate: 9.58%
Confusion matrix:
      False True class.error
False  7989  339  0.04070605
True    606  930  0.39453125
```

Figure 13: Testing Model with ntree = 1000 trees

Testing our random forest model with ntree = 1000, we notice the out-of-bag rate is 9.58% which is a slight increase from our base model. So we can make the assumption that leaving the model with ntree = 500 will give a better error rate.

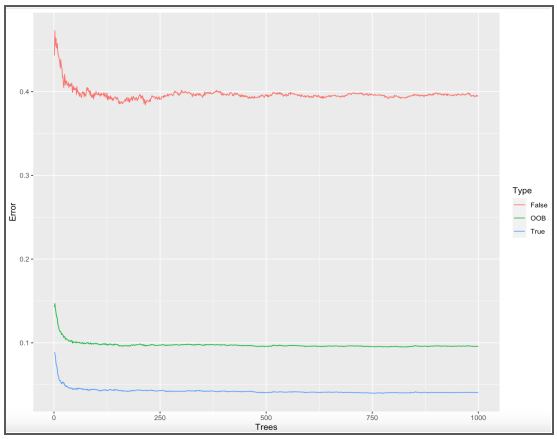


Figure 14: Plot of Error Rate for ntree = 1000

And in Figure 14, the error rate on the plot does not seem to decrease like we thought it would when adding more trees.

```
Call:
 randomForest(formula = Revenue ~ ., data = shoppers_train, weights = weights)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 9.72%
Confusion matrix:
      False True class.error
False  7982  346  0.04154659
True    613  923  0.39908854
```

Figure 15: Applying Weights to the Model with Default Parameters

Applying the weights mentioned in the beginning of this report to handle imbalanced data resulted in a higher out-of-bag error rate. However, we try to understand that this higher error rate can be because we have fine-tuned the model so it gives a more accurate estimate of the OOB estimate.

	Predicted	
Truth	False	True
False	8328	0
True	0	1536

Figure 16: Predict() on Model with Weights

When we look at the misclassification table, our theory of a fine-tuned model stands corrected because the false negatives and false positives are 0. This means the model is accurately predicting whether a customer makes a purchase or not.

```
> oob.values <- vector(length = 10)
> for (i in 1:10) {
+   temp.model <- randomForest(Revenue ~., data = shoppers_train, mtry = i, ntree = 1000)
+   oob.values[i] <- temp.model$err.rate[nrow(temp.model$err.rate), 1]
+ }
```

Figure 17: Testing Out Different Variables Considered at each Split (mtry)

In an attempt to apply more parameters to our model, we test out different mtry values.

```
> oob.values
 [1] 0.15085158 0.10036496 0.09742498 0.09874290 0.09762774 0.09651257 0.09722222 0.09641119 0.09732360 0.09833739
```

Figure 18: OOB Rates from Vector 1:10

A vector of length 10 gets printed out, displaying the iteration of error rates from mtry = 1 to mtry = 10.

```
> #find minimum error
> min(oob.values)
[1] 0.09641119
> #find optimal value for mtry
> which(oob.values == min(oob.values))
[1] 8
```

Figure 19: Optimal mtry Value from Minimum OOB Rate

Using the min() function, we get the minimum OOB rate to be 0.0964, which is the 8th value in the vector.

```
rf_param <- randomForest(Revenue ~., data = shoppers_train,
                         ntree = 1000,proximity = TRUE,
                         mtry = which(oob.values == min(oob.values)),
                         weights = weights )
rf_param
```

```
Call:
 randomForest(formula = Revenue ~ ., data = shoppers_train,
s)),      weights = weights)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 8

        OOB estimate of  error rate: 9.83%
Confusion matrix:
      False True class.error
False  7948  396  0.04745925
True    574  946  0.37763158
```

Figure 20: Random Forest Model with Parameters Defined

In Figure 20, we put all the parameters into our model. We note that the OOB estimate actually has a slight decrease of 0.02%, indicating there is a very small change between the weighted model in Figure 15, and the model we have built here (Figure 20).

	Predicted	
Truth	False	True
False	8344	0
True	0	1520

Figure 21: Prediction Rate of Fine-Tuned Model

And using the predict() function in Figure 21, we still get 0 for false positive and false negative values. This means there is a high chance that after testing out different parameters, we've created a model with a very high accuracy of predicting customers who make a purchase or not.

Precision	1
Accuracy	1
Recall	1
F1 score	1

Figure 22: Performance Metric of Confusion Matrix on Fine-Tuned Model

As mentioned previously, we are dealing with the case where the model is accurately predicting a customer's purchase. Therefore the metric for precision, accuracy, recall, and F1 score are all equal to 1.

b. Out-of-Sample Performance

	Predicted	
Truth	False	True
False	1984	94
True	133	255

Figure 23: Out-of-Sample Misclassification Table

In Figure 23, we see our out-of-sample misclassification table using the predict() function. Although the in-sample provided very accurate results, the testing dataset, which hasn't been adjusted to the optimal model, produces results for false positive and false negative values. We get 94 false negative values and 133 false positive values.

Precision	0.95
Accuracy	0.91
Recall	0.94
F1 score	0.95

Figure 24: Out-of-Sample Performance Metric of Confusion Matrix on Fine-Tuned Model

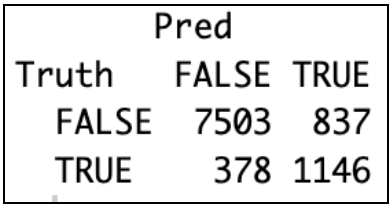
Looking at the metrics in Figure 24, the high precision of 0.95 indicates fewer false positive predictions. The accuracy, 0.91, tells us the overall correctness of the model's predictions. A high recall, 0.94, indicates fewer false negatives. And the F1 score gives us the mean between precision and recall. Looking at all these values, we notice they are generally on the higher end (0.90+). This suggests that our fine-tuned model is performing well in its predictions for the classification of generating revenue.

Support Vector Machine

Support vector machine has good performance on binary-classification problems and can maximize the gap between two classes. It handles imbalanced data and nonlinear relationships well, therefore, it is a great choice of our problem.

a. In-Sample Performance

We build the initial model with a linear kernel, which is the simplest kernel of SVM. However, we have good results from the initial model. The in-sample precision is 0.899, accuracy is 0.877, recall is 0.952, F1 score is 0.925 (Figure 26).



	Pred	
Truth	FALSE	TRUE
FALSE	7503	837
TRUE	378	1146

Figure 25: In-Sample Misclassification Matrix for Support Vector Machine

Precision	0.8996403
Accuracy	0.8996403
Recall	0.9520365
F1 score	0.9250971

Figure 26: In-Sample Performance Metrics

b. Out-of-sample Performance

The out-of-sample performance holds no significant difference than the in-sample performance, indicating the robustness of this model. The precision is 0.895, accuracy is 0.875, recall is 0.954, F1 score is 0.923 (Figure 27).

Precision	0.895
Accuracy	0.0875
Recall	0.954
F1 score	0.923

Figure 26: Out-of-Sample Performance Metrics

Conclusion:

To conclude our findings, we looked at three different models, decision tree, random forest, and support vector machine, to help us predict whether a customer will make a purchase or not. We learned how to deal with data variance and applied weights to fix our problem. And through adjusting various parameters, we saw a more accurate and improved performance from our optimal model.

With using the decision tree we found the variables that might be the most significant towards our response variable, Revenue. And when we applied the random forest model, testing hundreds of trees, it provided very accurate results as we saw in our misclassification table (Figure 21) and the performance metrics (Figure 22). Even with the out-of-sample performance, we saw a high precision, accuracy, recall and F1 score. As for the support vector machine, we found that the simple model with linear kernel generates good results, as seen in Figure 26.

References:

Our Dataset:
https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset

ClassKeita, Z. (2022, September 21). *Classification in Machine Learning: An Introduction*.
https://www.datacamp.com/blog/classification-machine-learning

Brownlee, J. (2020, August 27). *Probabilistic Model Selection with AIC, BIC, and MDL*.
MachineLearningMastery.com.
https://machinelearningmastery.com/probabilistic-model-selection-measures/