MSIN0025 Data Analytics II

Individual Report of DA II
Data source:
https://www.kaggle.com/competitions/stat441datachallenge1/overview
Word count: 1998

1. The Problem

The owner of a jewelry shop wants to increase revenue through website improvement and predict the number of success conversions. To identify the most relevant factors that might affect customers' shopping willingness, all the factors with significant weighting impacts on a transaction including quality of products, the ease of navigating the online shop, and customer services should be included.

The question can be viewed from a broader scope. In 2019, online sales constituted 19.4% of all retail sector, let alone the trend being spurred up by the Covid-19 pandemic [1]. Retail store players should investigate online visitors' reaction to capture their tastes via UX (User Experience) improvement and thus larger sales.

With successful predictions on online customers' interaction, precautions could be made through:

1) Ensuring sufficient stock level to fulfil upcoming demand.
2) UX optimization – enhance brand loyalty and define customer journeys on products most conducive to business success.

The dataset source was set on Kaggle official website two years ago. Since Kaggle did not specify if the dataset is real-world data, we cannot determine the date of retrieving the data. The outcome of this study may provide insights related to webpage management; however, companies (including the ones in the jewelry industry) should not solely rely on this dataset to conduct webpage improvement analysis.

The target attribute of this data project is "Revenue" (successful conversion). Instead of being listed as numeric values, the binary outcome variable presents whether the customer's visit had resulted in revenue (0 = False, 1 = True).

2. Understand the Data

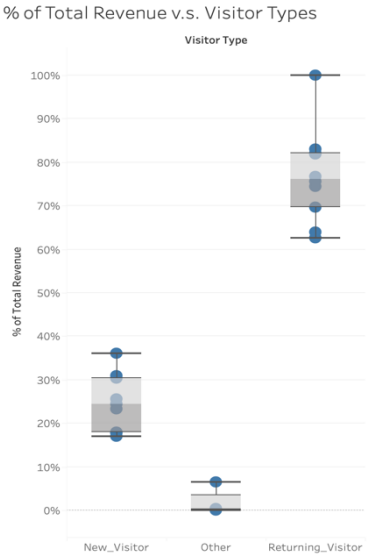
The dataset comprises 8630 training dataset and 3700 testing dataset. Variables covers time each user spent on administrative/informational/product related pages, and the nature of customers (i.e. existing/new, region, browsing months/dates). Out of all the attributes, only two attributes (i.e. months and visitor types) are categorical, while the others are numerical. Categorical data is first addressed and followed by feeding all numerical attributes to correlation tables.

According to Figure 1, new visitors have a notably higher conversion rate (25%) than returning visitors (15%). This points out that the shop can leverage the sense of novelty to make a sale [2]. However, in Figure 2, the box-and-whisker plot suggests that the majority of conversion rate is brought in by returning visitors due to the larger population. Hence, introducing customer loyalty strategies are more urgent than attracting new customers. Compared with visitor types, months of website browsing do not seem to have an obvious pattern.

Figure 1: Revenue v.s. Visitor Types

Revenue	New_Visitor	Other	Returning_Visitor
<int>	<dbl>	<dbl>	<dbl>
0	906	46	6343
1	295	12	1028

Figure 2



We then shift our focus to numerical data. We remove categorical attributes from the other attributes to generate the table of the top 10 highest absolute correlated attributes with 'Revenue' (Figure 4). Alternatively, we could visualize the analysis of the top correlated attributes with 'Revenue' in Figure 5.

	cor	abs.cor
	<dbl>	<dbl>
PageValues	0.49786063	0.49786063
ExitRates	-0.20783428	0.20783428
BounceRates	-0.15388837	0.15388837
ProductRelated	0.15306104	0.15306104
ProductRelated_Duration	0.14132871	0.14132871
Administrative	0.12871815	0.12871815
Informational	0.08261679	0.08261679
SpecialDay	-0.07893131	0.07893131
Administrative_Duration	0.07624346	0.07624346
Informational_Duration	0.06749057	0.06749057

Figure 4

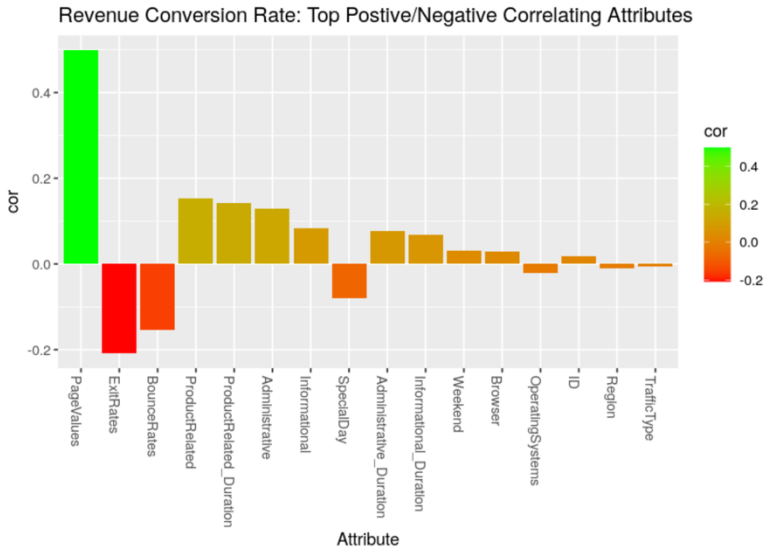


Figure 5: Bar plot on the correlated attributes with "Revenue

Based on Figure 4 & 5, we can set the threshold as 0.1 for absolute correlation and select the attributes that are larger than or equal to the threshold as our relevant attributes. Consequently, 'PageValues', 'ExitRates', 'BounceRates', 'ProductRelated', 'ProductRelatedDuration', and 'Administrative' are most relevant to our problem. A matrix of correlation values between these attributes are listed as Figure 6.

A matrix: 6 × 6 of type dbl

	Administrative	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
Administrative	1.0000000	0.41561898	0.34725096	-0.2249286	-0.3172540	0.10272475
ProductRelated	0.4156190	1.00000000	0.85360018	-0.2036301	-0.2879916	0.04640724
ProductRelated_Duration	0.3472510	0.85360018	1.00000000	-0.1798511	-0.2419708	0.04582507
BounceRates	-0.2249286	-0.20363007	-0.17985106	1.0000000	0.9149374	-0.11952249
ExitRates	-0.3172540	-0.28799165	-0.24197081	0.9149374	1.0000000	-0.17425098
PageValues	0.1027247	0.04640724	0.04582507	-0.1195225	-0.1742510	1.00000000

Figure 6

'ExitRates' and 'BounceRates' are the attributes with the second and third highest absolute correlation. From Figure 6, they have significantly high correlation (0.915); the same trend is presented as a scatter plot in Figure 7. With such a high correlation, these two attributes are complementary as either customers are not interested in the information shown, or technical difficulties of navigating the sites, which make visitors want to terminate the browsing, and thus high negative correlation.

Figure 7: 'ExitRates' and 'BounceRates' have significantly high correlation

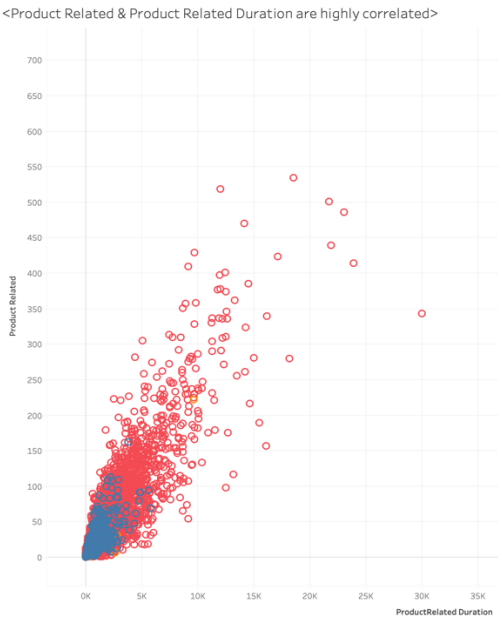


Figure 8: 'ProductRelated' and 'ProductRelatedDuration' are complementary attributes

'ProductRelated' and 'ProductRelatedDuration' indicate the number of product related pages and the time spent on product related pages, respectively. While these two attributes have high correlation with 'Revenue' as customers show genuine interests by browsing more websites, the attributes themselves are highly correlated (0.854).

Since people who visit more pages may be genuinely interested in the products, ‘PageValues’ has the highest correlation out of all the attribute. Finally, there is a weak correlation between the number of administrative pages visited (‘Administrative’) and ‘Revenue’.

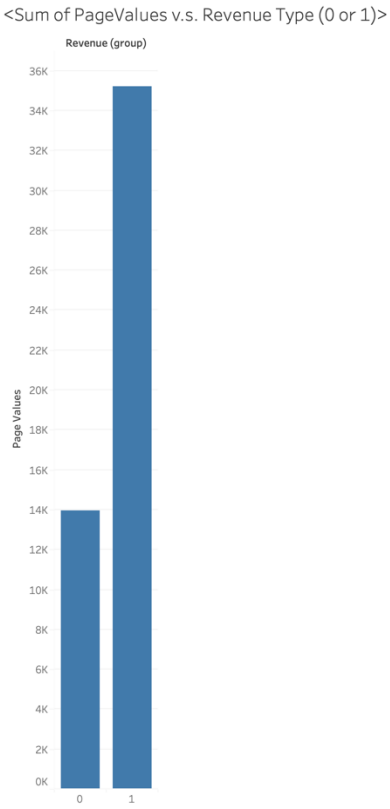


Figure 9: Higher Page Values result in more conversions

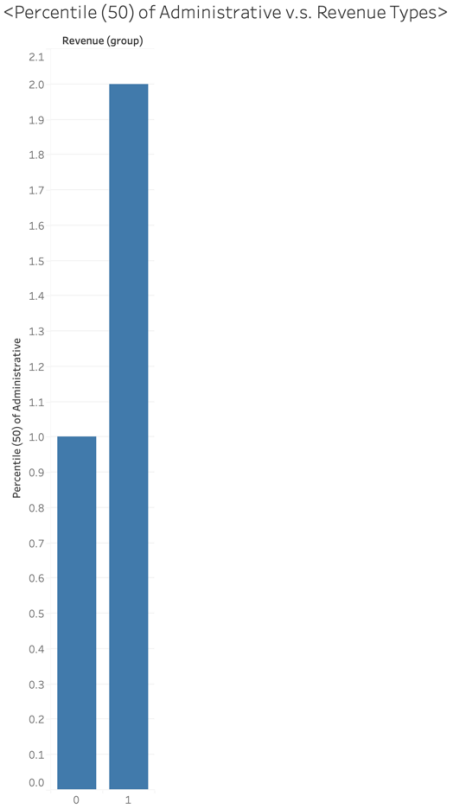


Figure 10: Higher Administrative Pages result in more conversions

3. Prepare the Data

3.1 Missing Values

There were 8 entries that had 8 missing attributes. As we should remove columns with missing proportions greater than 0.5%, we compute the proportion of missing values. Since all the proportions are 0.093%, we ended up keeping all the relevant columns,

Missing values are further imputed with mean values of its corresponding attributes, on both training and testing dataset.

```
# 1.address missing values
# count the number of NAs for each attribute
countNAs <- function(v){
    sum(ifelse(is.na(v)|v == "",1,0))
}

web.countNAs <- sapply(web.df, countNAs)
web.countNAs
```

ID: 0 **Administrative:** 8 **Administrative_Duration:** 8 **Informational:** 8 **Informational_Duration:** 8 **ProductRelated:** 8 **ProductRelated_Duration:** 8 **BounceRates:** 8 **ExitRates:** 8 **PageValues:** 0 **SpecialDay:** 0 **Month:** 0 **OperatingSystems:** 0 **Browser:** 0 **Region:** 0 **TrafficType:** 0 **VisitorType:** 0 **Weekend:** 0 **Revenue:** 0

```
# For those variables contain missing values, calculate the proportion of missing values.
round(web.countNAs[web.countNAs != 0]/nrow(web.df),5)
```

Administrative: 0.00093 **Administrative_Duration:** 0.00093 **Informational:** 0.00093 **Informational_Duration:** 0.00093 **ProductRelated:** 0.00093 **ProductRelated_Duration:** 0.00093 **BounceRates:** 0.00093 **ExitRates:** 0.00093

3.2 Perform one-dot-encoding of categorical attributes (Visitor types)

In Set 1, only ‘VisitorTypes’ is categorical. One-hot-encoding is leveraged to convert the categorical feature in a form that can be easily processed by a computer [3]. Hence, ‘VisitorTypes’ is transformed into three columns, which are ‘VisitorTypeNew_Visitor’, ‘VisitorTypeOther’, and ‘VisitorTypeOld_Visitor’, with each row records its visitor type with the value 0 or 1.

3.3 Train/Validation Split

The dataset came as a package, including training and testing dataset. A validation is required to assess the performance of generated models. The training dataset is split into smaller training set (75%) and validation set (25%).

Data Type	Entries
Smaller Training	6472
Validation	2158
Testing	3700

4. Generate and Test Prediction Models

4.1 Scoring Rule – Brier Skill Score (BSS)

Since the outcome variable only has two possible events, the BSS is chosen to evaluate each model. This scoring rule is especially suitable for problems with binary outcomes as BSS assess the relative skill of the probability forecast over all generated models. BSS = 1 denotes a perfect model, whereas BSS = 0 indicates the skill of the reference forecast [4].

```
# Function to calculate the average Brier score.
brier.score <- function(predictions, realizations) {
     return(mean((predictions - realizations)^2))
}
```

```
# Function to calculate the Brier skill score
skill.score <- function(predictions, realizations, brier.ref) {
    # calculate the Brier score for your predictions.
    brier.score <- brier.score(predictions, realizations)
    return(1 - brier.score/ brier.ref)
}
```

4.2 Predictive Base Model Selection

Due to the nature of classification, I chose **Logistic Regression**, **Regression Tree**, and **XGBoost** as my base models.

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. Since it is mostly used in modeling binary outcomes [5], the model is suitable for online jewelry store conversion forecast.

We discovered that adding interaction between variables will increase its predictive power due to certain correlations between variables[6].

[34]:
```
# Logistic Regression

# Train logistic regression model
logit.reg <- glm(Revenue~., data = smalltrainSet1.select, family = "binomial")
summary(logit.reg)
```

[35]:
```
# Generate predictions in the validation set
# Remember to set type = "response", which gives you the probability rather than the outcome variable
pred.logit.reg <- predict(logit.reg, validSet1.select, type = "response")
#pred.logit.reg
```

[36]:
```
# Calculate the Brier Skill Score
skill.score(pred.logit.reg, validSet1.select$Revenue, brier.ref.valid)
```

```
0.344259177347891
```

```
[37]: # Train logistic regression model with interactions
      logit.reg <- glm(Revenue ~ . + .^2, data = smalltrainSet1.select, family = "binomial")
      summary(logit.reg)

[39]: # Generate predictions in the validation set
      pred.logit.reg <- predict(logit.reg, validSet1.select, type = 'response')

[40]: # Calculate the Brier Skill Score based on the interaction logistic regression model
      skill.score(pred.logit.reg, validSet1.select$Revenue, brier.ref.valid)

      # The score is higher than the model without interactions, which means adding interactions can improve the model

      0.375868973506968
```

Compared to logistic regression, **Regression Tree** is most effective if the relationship between variables is non-linear [7]. However, as is shown in Figure 11, regression tree generally yields worse results than logistic regression. This suggests that there may be certain linear relationship between predictors that the logistic regression takes better account for [8].

XGBoost (Extreme Gradient Boosting) falls under the category of boosted trees that combine many weak and inaccurate models into highly accurate model. Key features include:

1) Decision trees in gradient boosting are built additively – every new tree is built to improve on the deficiencies of the previous trees so that the overall model performance is improved.

2) A set of parameters can be adjusted to avoid overfitting [9].

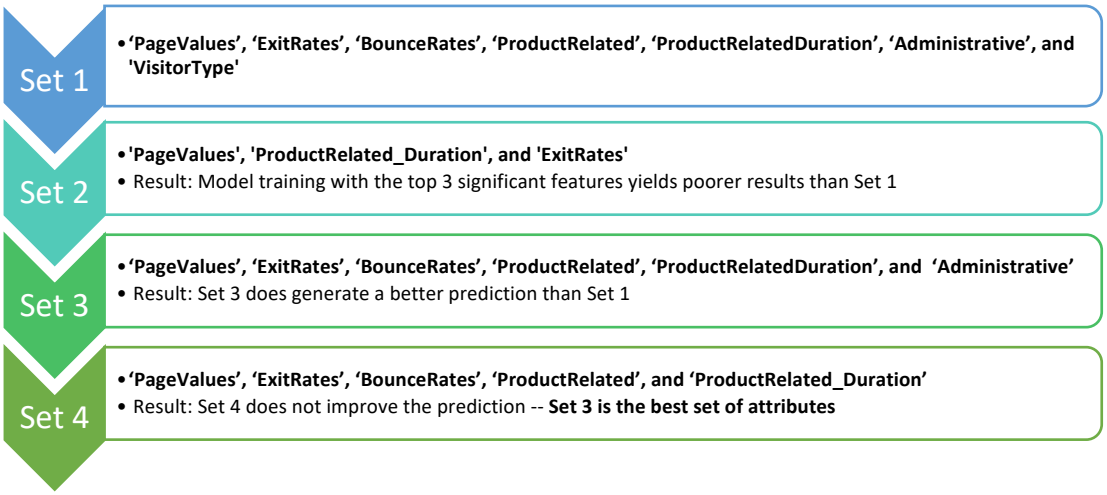
Therefore, we use XGBoost as our final base model to see if we can further improve model results.

Brier Skill Score	Logistic Regression (Interaction)	Regression Tree	XGBoost
Set 1	0.375869	**0.310860**	0.425493
Set 2	0.348037	0.294424	0.383946
Set 3	**0.376077**	0.298002	**0.426273**
Set 4	0.351022	0.275808	0.407745

Figure 11: BSS of base models from Set 1 to Set 4

4.3 Model Configuration

Each set of models (1 to 4) configuration will be fit into the forecast models to assess performance sequentially. Content of each set is listed as follow, and we will justify reasons for such selection:



Set 1 – is composed of the top 6 attributes that have the highest correlation with 'Revenue', plus 'Visitor Type' (from Section 2). These are selected based on individual variable's meaningful sense to the problem.

**Attributes, except from the ones included in Set 1 (correlation <0.1), are irrelevant and thus we do not take them into account during model training.

Set 2 – We want to know if a more concise parameter set can empower the prediction. Hence, Set 2 is constituted of 3 attributes with the most significant features. In Section 2, we discovered two sets of attributes are complementary, so one out of both sets are selected to be part of Set 2 based on feature importance. 'Administrative' and 'VisitorTypes' are removed due to non-significant reason. This can be verified by Figure 12 and 13, where the 'ProductRelated', 'ExitRates', and 'PageValues' have 1) the lowest P-Values, suggesting that we can reject Null hypothesis (the variable has no correlation with the dependent variable) and 2) the highest feature importance [10].

[82]:
```
# Logistic Regression

# Train logistic regression model
logit.reg <- glm(Revenue~., data = smalltrainSet1.select, family = "binomial")
summary(logit.reg)
```

```
Call:
glm(formula = Revenue ~ ., family = "binomial", data = smalltrainSet1.select)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-6.3786  -0.4622  -0.3785  -0.2038   3.0994

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -2.258e+00  1.078e-01 -20.945  < 2e-16 ***
Administrative           6.208e-03  1.219e-02   0.509  0.61051
ProductRelated           5.878e-03  1.385e-03   4.245 2.19e-05 ***
ProductRelated_Duration  1.564e-05  2.968e-05   0.527  0.59828
BounceRates             -8.051e+00  4.632e+00  -1.738  0.08220 .
ExitRates               -1.312e+01  3.235e+00  -4.056 5.00e-05 ***
PageValues               8.845e-02  3.525e-03  25.091  < 2e-16 ***
VisitorTypeNew_Visitor   3.306e-01  1.185e-01   2.789  0.00529 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5598.9  on 6471  degrees of freedom
Residual deviance: 3857.2  on 6464  degrees of freedom
AIC: 3873.2

Number of Fisher Scoring iterations: 7
```

Figure 12: Summary of logistic regression & P-Values based on Set 1 predictors

[48]:
```
# Print out feature importance
# Gain indicates the contribution of each feature. Higher gain means higher importance.
xgb.importance(colnames(xsmalltrain.xgb), model = xgb.trees)
```

A data.table: 7 × 4

Feature	Gain	Cover	Frequency
<chr>	<dbl>	<dbl>	<dbl>
PageValues	0.593188895	0.17892289	0.15420928
ProductRelated_Duration	0.116085321	0.25859646	0.24940991
ExitRates	0.107043650	0.26939277	0.25019670
BounceRates	0.065197056	0.07620810	0.09520063
ProductRelated	0.064316375	0.13800161	0.15342250
Administrative	0.044676609	0.05015857	0.08261212
VisitorTypeNew_Visitor	0.009492094	0.02871960	0.01494886

Figure 13: Feature importance of XGBoost based on Set 1 predictors

However, Set 2 yields a poorer BSS than Set 1. This explains a training model including predictors with larger effect may not result in a better prediction, potentially because the model overlooks some features and jeopardize the outcome.

Set 3 – From the experience of Set 2, we realize that it is not sensible to form a smaller set of configurations, as dropping less significant features does not ensure better performance.

A better way to decide which features to include is removing attributes with the lowest importance one by one and assess if each step improves the model result [11]. Consequently, Set 3 is composed of all the predictors from baseline (Set 1), except for 'VisitorTypes'.

In return, Set 3 does generate a better BSS in terms of Logistic Regression and XGBoost.

Set 4 – We do the same thing for Set 4 to see if we improve the model further by removing the second lowest feature (Administrative). Set 4 yields poorer results from all three of the prediction models, suggesting that the attribute 'Administrative' does influence the models to some extent.

After feeding the aforementioned four sets of attributes into model training, Set 3 is proved to be the most effective configuration set, and will be the predictors for ensemble models and final prediction on testing data. These six elements are most likely to be the key determinators of the online store's successes.

4.4 Ensemble Model – Stacking (regression tree and XGBoost)

Ensemble techniques are the methods that use multiple training models to produce better prediction than base learners taken alone. Stacking learns heterogeneous weak learners in parallel and combines them by training a meta-learner to output a prediction based on the weak learners' predictions [12]. Hence, we train two stacking models that takes all three of the base models into consideration, in hope of improving our BSS.

Step 1. Prepare a stacker data frame with the base models

[78]:
```
# 1. Train the base models (M1, M2 and M3) by using the smaller training set,
#    and generate predictions from these three models in the validation set.
# 2. Fit the stacker model (regression tree or XGBoost) to the predictions generated in Step 1.
#    The dependent variable is Revenue in the training set and the independent variables are the predictions obtained in Step 1.
```

[79]:
```
# Logistic regression
M1 <- glm(Revenue ~  . + .^2, data = smalltrainSet3.select, family = "binomial")
M1.predict <- predict(M1, valid.df, type = "response")
```

```
Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"
```

[80]:
```
# Regression tree
M2 <- rpart(Revenue ~ .,
            data = smalltrainSet3.select,
            control = rpart.control(cp = 0.0001))
M2.predict <-  predict(M2, valid.df)
```

[87]:
```
# XGBoost
xtrain.xgb <- model.matrix(~ 0 + ., data = smalltrainSet3.select[,-7])
ytrain.xgb <- as.vector(smalltrainSet3.select$Revenue)

xvalid.xgb <- model.matrix(~ 0 + ., data = validSet3.select[,-7])
yvalid.xgb <- as.vector(validSet3.select$Revenue)

M3 <- xgboost(xtrain.xgb, ytrain.xgb,
              max.depth = 3,
              #nthread = workers,
              nround = 200,
              objective = "binary:logistic",
              verbose = 0)

M3.predict <- predict(M3, xvalid.xgb)
```

```
[14:21:57] WARNING: ../..//amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

[88]:
```
# 2. Fit a stacker model to the predictions generated in last step.
```

[89]:
```
# Construct the stacker dataframe in validation set
stacker.df <- data.frame(Revenue = valid.df$Revenue,
                         M1.predict = M1.predict,
                         M2.predict = M2.predict,
                         M3.predict = M3.predict)

head(stacker.df)
```

A data.frame: 6 × 4

	Revenue	M1.predict	M2.predict	M3.predict
	<dbl>	<dbl>	<dbl>	<dbl>
8	0	0.06074425	0.0000000	0.0272882022
9	0	0.97814309	0.7272727	0.4445826113
19	0	0.02868217	0.0000000	0.0005039875
31	0	0.27164831	0.7500000	0.2154290676
32	1	0.67949219	0.9565217	0.9745755196
37	0	0.06577577	0.0000000	0.0438112579

Step 2. Training two meta-learners (Regression Tree & XGBoost) to output a prediction

[90]:
```
# Stacking model 1 - Regression Tree
```

[91]:
```
# Train Stacker model 1: regression tree
# Dependent variable is the Revenue, independent variables are M1, M2, M3 predictions
stackerModel_1 <- rpart(Revenue ~ .,
                        data = stacker.df,
                        control = rpart.control(cp = 0.0004))
```

[92]:
```
# Predict from stacker model 1 --- regression tree
stacker.predict.rt <- predict(stackerModel_1, stacker.df[, -1])

# Score the stacker model's prediction
skill.score(stacker.predict.rt, validSet3.select$Revenue, brier.ref.valid)
```

```
0.604421940748509
```

[93]:
```
# Stacking model 2 - XG Boost
```

[94]:
```
# Convert stacker.df to matrix format
stacker.x.xgb <- model.matrix(~ 0 + ., data = stacker.df[,-1])
stacker.y.xgb <- as.vector(stacker.df[,1])

stackerModel_2 <- xgboost(stacker.x.xgb, stacker.y.xgb,
                          max.depth = 3,
                          #nthread = workers,
                          nround = 20,
                          objective = "binary:logistic",
                          verbose = 0)
# Different settings of max.depth and nround have been examined.
# max.depth = 3 and nround = 20 provide the highest Brier Skill Score
```

```
[14:22:31] WARNING: ../..//amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

[96]:
```
# Predict from stacker model 2 --- XGBoost, and calculate the Brier Skill Score
predict.variables.xgb <- model.matrix(~ 0 + ., data = stacker.df[, -1])
stacker.predict.xgb <- predict(stackerModel_2, predict.variables.xgb)

# Score the stacker model's prediction
skill.score(stacker.predict.xgb, validSet3.select$Revenue, brier.ref.valid)
```

```
0.527222168279419
```

To conclude the prediction section, we compare the BSS of three base models and two stacking models as a table below. With Set 3 predictor, we realise that **Stacking model – Regression Tree** is the best model.

A data.frame: 5 × 2

Model	BSS
<chr>	<dbl>
Logistic Regression	0.376077
Regression Tree	0.298002
XGBoost	0.426273
Stacking Regression Tree	0.604422
Stacking XGBoost	0.527222

Therefore, the final step is to use the **Regression Tree Stacking** to retrain the entire training set and generate predictions on our testing set.

[106]:
```
# Train Stacker model 1: regression tree
# Dependent variable is the Revenue, independent variables are M1, M2, M3 predictions
best.stackerModel <- rpart(Revenue ~ .,
                           data = best.stacker.df,
                           control = rpart.control(cp = 0.0004))
```

[107]:
```
# Predict from our best stacker model --- regression tree
best_stacker.predict.rt <- predict(best.stackerModel, best.stacker.df)
best_stacker.predict.rt
```

1: 0.0666666666666667 **2:** 0.000969367972082202 **3:** 0.000969367972082202 **4:** 0.000969367972082202 **5:** 0.000969367972082202 **6:** 0.000969367972082202 **7:** 0.0666666666666667 **8:** 0.000969367972082202 **9:** 0.0261437908496732 **10:** 0.000969367972082202 **11:** 0.933333333333333 **12:** 0.000969367972082202 **13:** 0.000969367972082202 **14:** 0.962365591397849 **15:** 0.993814432989691 **16:** 1 **17:** 0 **18:** 0.000969367972082202 **19:** 0.000969367972082202 **20:** 0.000969367972082202 **21:** 0.938461538461538 **22:** 0.000969367972082202 **23:** 0.000969367972082202 **24:** 0.000969367972082202 **25:** 0.000969367972082202 **26:** 0.916666666666667 **27:** 0.993814432989691 **28:** 0 **29:** 0.0149253731343284 **30:** 0.000969367972082202 **31:** 0.0149253731343284 **32:** 0.993814432989691 **33:** 0.000969367972082202 **34:** 0 **35:** 0.000969367972082202 **36:** 0.105263157894737 **37:** 0.000969367972082202 **38:** 0.000969367972082202 **39:** 0.000969367972082202 **40:** 0.000969367972082202 **41:** 0.000969367972082202 **42:** 0.993814432989691 **43:** 0.000969367972082202 **44:** 0.000969367972082202 **45:** 0 **46:** 0.993814432989691 **47:** 0.000969367972082202 **48:** 0.0421052631578947 **49:** 0.142857142857143 **50:** 0.000969367972082202 **51:** 0.000969367972082202 **52:** 0.000969367972082202 **53:** 0.993814432989691 **54:** 0.000969367972082202 **55:** 0.000969367972082202 **56:** 0.000969367972082202 **57:** 0 **58:** 0.993814432989691 **59:** 0.000969367972082202 **60:** 0.000969367972082202 **61:** 0.000969367972082202 **62:** 0.0261437908496732 **63:** 0.416666666666667 **64:** 0.0261437908496732 **65:** 0.000969367972082202 **66:** 0.000969367972082202 **67:** 0.000969367972082202 **68:** 0.363636363636364 **69:** 0.0149253731343284 **70:** 0.000969367972082202 **71:** 0 **72:** 0.962365591397849 **73:** 0.307692307692308 **74:** 0 **75:** 0.000969367972082202 **76:** 0.000969367972082202 **77:** 0.428571428571429 **78:** 0.000969367972082202 **79:** 0.000969367972082202 **80:** 0.000969367972082202 **81:** 0.000969367972082202 **82:** 0.5625 **83:** 0.000969367972082202 **84:** 0.000969367972082202 **85:** 0.000969367972082202 **86:** 0 **87:** 0.000969367972082202 **88:** 0.000969367972082202 **89:** 0.000969367972082202 **90:** 0.000969367972082202 **91:** 0.000969367972082202 **92:** 0.0666666666666667 **93:** 0.000969367972082202 **94:** 0.000969367972082202 **95:** 0.000969367972082202 **96:** 0.6 **97:** 0.000969367972082202 **98:** 0.962365591397849 **99:** 0.000969367972082202 **100:** 0.000969367972082202 **101:** 0.28125 **102:** 0.000969367972082202 **103:** 0.000969367972082202 **104:** 0.555555555555556 **105:** 0.000969367972082202 **106:** 0 **107:** 0.938461538461538 **108:** 0 **109:** 0.000969367972082202 **110:** 0.08 **111:** 0 **112:** 0.0666666666666667 **113:** 0.000969367972082202 **114:** 0.0666666666666667 **115:** 0.000969367972082202 **116:** 0 **117:** 0.75 **118:** 0.307692307692308 **119:** 0.938461538461538 **120:** 0.000969367972082202 **121:** 0 **122:** 0.993814432989691 **123:** 0.000969367972082202 **124:** 0.000969367972082202 **125:** 0.962365591397849 **126:** 0.000969367972082202 **127:** 0.444444444444444 **128:** 0.000969367972082202 **129:** 0.0666666666666667 **130:** 0.000969367972082202 **131:** 0.000969367972082202

** Since the testing dataset came without the whole column of target attribute ‘Revenue’ (it was meant to be generated after conducting this project), We cannot give a BSS to the final prediction.

5. Problem Conclusions and Recommendations

Since our best model – Stacking Regression Tree has a high BSS value of 0.604 for all training set, our prediction model is proved to be significant. The result of our prediction – the probability for each customer to be classified under the category 'Revenue = 1', along with the best model can be shared with the online jewelry shop owner. Therefore, the owner can improve their online shopping experience and sales based on our study.

Forcast model indicates that **'PageValues', 'ExitRates', 'BounceRates', 'ProductRelated', 'ProductRelated_Duration'**, and **'Administrative'** are highly relevant to the conversion, so shops should try to prolong the time each visitor spends on their online stores by making the products and website interfaces more intriguing (i.e. front-end/ back-end design, promotion schemes, and purchase online 'traction' to gain access to target customers). Additionally, based on the attribute 'VisitorTypes' in data visualization section, the owners should devise loyalty strategies to retain old customers.

A limitation to increase the overall online sales maybe the products. Ecommerce platforms are just a medium to deliver products. The nature of product may have more weighting power than the delivery channel – we cannot make predictions on success conversion purely based on figures of user interactions.

Reference

1. Coppola D. Topic: E-commerce in the United Kingdom (UK) [Internet]. Statista. 2022 [cited 2022 May 9]. Available from: https://www.statista.com/topics/2333/e-commerce-in-the-united-kingdom/#topicHeader__wrapper

2. Sharman BS. Predicting Visitor-to-Customer Conversion for an Online Store via Supervised Machine Learning… [Internet]. Medium. 2020 [cited 2022 May 7]. Available from: https://bharatss.medium.com/predicting-visitor-to-customer-conversion-for-an-online-store-via-supervised-machine-learning-d5943b4a0e04

3. https://www.facebook.com/MachineLearningMastery. Why One-Hot Encode Data in Machine Learning? [Internet]. Machine Learning Mastery. 2017 [cited 2022 May 9]. Available from: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

4. Dash S. Brier Score - How to measure accuracy of probablistic predictions [Internet]. Machine Learning Plus. 2020 [cited 2022 May 8]. Available from: https://www.machinelearningplus.com/statistics/brier-score/

5. W. Edgar T, O. Manz D. Logistic Regression - an overview | ScienceDirect Topics [Internet]. www.sciencedirect.com. 2017 [cited 2022 May 7]. Available from: https://www.sciencedirect.com/topics/computer-science/logistic-regression

6. Frost J. Understanding Interaction Effects in Statistics [Internet]. Statistics By Jim. 2017 [cited 2022 May 9]. Available from: https://statisticsbyjim.com/regression/interaction-effects/

7. Prasad A. Regression Trees | Decision Tree for Regression | Machine Learning [Internet]. Analytics Vidhya. 2021 [cited 2022 May 7]. Available from: https://medium.com/analytics-vidhya/regression-trees-decision-tree-for-regression-machine-learning-e4d7525d8047

8. Bock T. Decision Trees Are Usually Better Than Logistic Regression [Internet]. Displayr. 2018 [cited 2022 May 10]. Available from: https://www.displayr.com/decision-trees-are-usually-better-than-logistic-regression/

9. Chen T, Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

- KDD ’16 [Internet]. 2016 [cited 2022 May 7];2939672.2939785(978-1-4503-4232-2/16/08). Available from: https://arxiv.org/abs/1603.02754

10. Frost J. How to Interpret P-values and Coefficients in Regression Analysis [Internet]. Statistics By Jim. 2017 [cited 2022 May 8]. Available from: https://statisticsbyjim.com/regression/interpret-coefficients-p-values-regression/

11. Hoang N. How to utilize Feature Importance correctly [Internet]. Medium. 2021 [cited 2022 May 6]. Available from: https://towardsdatascience.com/how-to-utilize-feature-importance-correctly-1f196b061192

12. Khandelwal Y. Ensemble Stacking for Machine Learning and Deep Learning [Internet]. Analytics Vidhya. 2021 [cited 2022 May 8]. Available from: https://www.analyticsvidhya.com/blog/2021/08/ensemble-stacking-for-machine-learning-and-deep-learning/

Code Appendix

Section 1. Online Jewelry Case Study

[1]:
```
# load the training data into R
web.df <- read.csv('online_jewellery_shop_train.csv')
```

[2]:
```
str(web.df)
```

```
'data.frame':   8630 obs. of  19 variables:
 $ ID                     : int  3935 7866 3727 4454 6186 6394 6760 6734 5399 9960 ...
 $ Administrative         : int  1 6 0 0 0 0 4 0 2 0 ...
 $ Administrative_Duration: num  50.3 299 0 0 0 ...
 $ Informational          : int  0 1 0 0 0 0 0 0 0 0 ...
 $ Informational_Duration : num  0 41 0 0 0 0 0 0 0 0 ...
 $ ProductRelated         : int  21 36 19 4 24 8 17 20 35 1 ...
 $ ProductRelated_Duration: num  586.2 696.4 620 46.2 671.1 ...
 $ BounceRates            : num  0 0 0 0.05 0.0267 ...
 $ ExitRates              : num  0.00556 0.00263 0.00789 0.06667 0.03657 ...
 $ PageValues             : num  0 0 0 0 0 ...
 $ SpecialDay             : num  0.6 0 0 0 0 0 0 0 0 0 ...
 $ Month                  : chr  "May" "May" "Feb" "Nov" ...
 $ OperatingSystems       : int  1 2 1 3 1 1 2 3 2 1 ...
 $ Browser                : int  1 2 1 2 1 1 2 2 2 1 ...
 $ Region                 : int  1 2 4 6 3 1 1 1 1 3 ...
 $ TrafficType            : int  2 4 2 13 2 1 13 1 10 8 ...
 $ VisitorType            : chr  "Returning_Visitor" "New_Visitor" "Returning_Visitor" "Returning_Visitor" ...
 $ Weekend                : int  0 1 0 0 0 0 1 0 1 0 ...
 $ Revenue                : int  0 0 0 0 0 0 0 0 0 0 ...
```

Section 2. Data Visualization

[3]:
```
# Data visualisation
# We start with categorical values (VisitorType and Month) to see their feature.
```

[4]:
```
# Revenue of different visitor types
library(janitor)
tabyl(web.df,Revenue, VisitorType)
```

```
Attaching package: ‘janitor’


The following objects are masked from ‘package:stats’:

    chisq.test, fisher.test
```

A tabyl: 2 × 4

Revenue	New_Visitor	Other	Returning_Visitor
<int>	<dbl>	<dbl>	<dbl>
0	906	46	6343
1	295	12	1028

2.1 Correlation between numerical attributes

[5]:
```
# Revenue of months across the year
tabyl(web.df,Revenue, Month)
```

A tabyl: 2 × 11

Revenue	Aug	Dec	Feb	Jul	June	Mar	May	Nov	Oct	Sep
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
0	259	1063	121	255	180	1201	2086	1575	300	255
1	47	152	3	50	23	135	253	534	77	61

[6]:
```
# Exploring how numerical attributes are correlated to "revenue" (customer conversion)
```

[7]:
```
# set up web_visualising for data visualisation
# remove all the attributes that are not numerical values (Month,VisitorType)
myvars <- names(web.df) %in% c("Month", "VisitorType")
web_visualising.df <- web.df[!myvars]
```

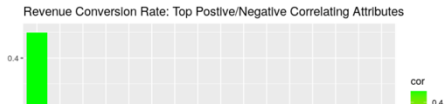
[8]:
```
#list out correlation between features and target attribute as a table
#find the ones most relevant to plot diagrams on tableu and r
```

[9]:
```
targetCol <- which(names(web_visualising.df)=="Revenue")
startCol <- which(names(web_visualising.df)=="ID")
endCol <- which(names(web_visualising.df)=="Weekend")

cor.RevenueConversionRate <- cor(web_visualising.df[,c(targetCol, startCol:endCol)]
                      ,use="complete.obs")[-1, "Revenue"]

cor.RevenueConversionRate.df <- data.frame(cor=cor.RevenueConversionRate, abs.cor=abs(cor.RevenueConversionRate),
                                row.names=names(cor.RevenueConversionRate))

cor.RevenueConversionRate.df <- cor.RevenueConversionRate.df[order(-cor.RevenueConversionRate.df$abs.cor),]

cor.RevenueConversionRate.df[1:10,]
```

A data.frame: 10 × 2

	cor	abs.cor
	<dbl>	<dbl>
PageValues	0.49786063	0.49786063
ExitRates	-0.20783428	0.20783428
BounceRates	-0.15388837	0.15388837
ProductRelated	0.15306104	0.15306104
ProductRelated_Duration	0.14132871	0.14132871
Administrative	0.12871815	0.12871815
Informational	0.08261679	0.08261679
SpecialDay	-0.07893131	0.07893131
Administrative_Duration	0.07624346	0.07624346

[10]:
```
library(ggplot2)
options(repr.plot.height=5)

ggplot(cor.RevenueConversionRate.df, aes(x=reorder(row.names(cor.RevenueConversionRate.df),-abs.cor), y=cor, fill=cor)) +
    geom_col() + ggtitle("Revenue Conversion Rate: Top Postive/Negative Correlating Attributes") + xlab("Attribute") +
    scale_fill_gradient(low="red", high="green") +
    theme(axis.text.x=element_text(angle=-90, hjust=0))
```



[11]:
```
# finds correlation values for the top 6 attributes that have a high correlation with "revenue"

cor(web_visualising.df[,c(2,6,7,8,9,10)],use="complete.obs")
```

A matrix: 6 × 6 of type dbl

	Administrative	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
Administrative	1.0000000	0.41561898	0.34725096	-0.2249286	-0.3172540	0.10272475
ProductRelated	0.4156190	1.00000000	0.85360018	-0.2036301	-0.2879916	0.04640724
ProductRelated_Duration	0.3472510	0.85360018	1.00000000	-0.1798511	-0.2419708	0.04582507
BounceRates	-0.2249286	-0.20363007	-0.17985106	1.0000000	0.9149374	-0.11952249
ExitRates	-0.3172540	-0.28799165	-0.24197081	0.9149374	1.0000000	-0.17425098
PageValues	0.1027247	0.04640724	0.04582507	-0.1195225	-0.1742510	1.00000000

Section 3. Data Preprocessing

3.1 Address missing values

[12]:
```
# Load testing dataset
webtesting.df <- read.csv('online_jewellery_shop_test_final.csv')
```

[13]:
```
# Data preprocessing

# Based on the previous section, we chose the to 6 attributes that have the highest correlation as our target attributes.
# So we will only process column 2,6,7,8,9,10,17
# Processes includes:
#  1. Adressing missing values
#  2. One-hot-encoding on 'VisitorTypes'
#  3. Scaling numerical attributes so that each value is between 0 and 1
```

[14]:
```
# 1.address missing values
# count the number of NAs for each attribute
countNAs <- function(v){
    sum(ifelse(is.na(v)|v == "",1,0))
}

web.countNAs <- sapply(web.df, countNAs)
web.countNAs
```

ID: 0 **Administrative:** 8 **Administrative_Duration:** 8 **Informational:** 8 **Informational_Duration:** 8 **ProductRelated:** 8 **ProductRelated_Duration:** 8 **BounceRates:** 8 **ExitRates:** 8 **PageValues:** 0 **SpecialDay:** 0 **Month:** 0 **OperatingSystems:** 0 **Browser:** 0 **Region:** 0 **TrafficType:** 0 **VisitorType:** 0 **Weekend:** 0 **Revenue:** 0

[15]:
```
# For those variables contain missing values, calculate the proportion of missing values.
round(web.countNAs[web.countNAs != 0]/nrow(web.df),5)
```

Administrative: 0.00093 **Administrative_Duration:** 0.00093 **Informational:** 0.00093 **Informational_Duration:** 0.00093 **ProductRelated:** 0.00093 **ProductRelated_Duration:** 0.00093 **BounceRates:** 0.00093 **ExitRates:** 0.00093

[16]:
```
# We are supposed to remove columns with missing proportions greater than 0.5.
# However, as all the proportions are 0.00093, which is not significant, we keep all of them

summary(web.df$Administrative)

# As shown in the table above, 0.093% of the data in 8 attributes are missing, respectively.
# Thus, we need to impute the missing data in these corresponding attributes.
# We replace the NAs in these 8 columns with their column means.

impute_data <- function(vec, mn){
    ifelse(is.na(vec), mn, vec)
}

web.df$Administrative <- impute_data(web.df$Administrative, mean(web.df$Administrative, na.rm=TRUE))
web.df$Administrative_Duration <- impute_data(web.df$Administrative_Duration, mean(web.df$Administrative_Duration, na.rm=TRUE))
web.df$Informational <- impute_data(web.df$Informational, mean(web.df$Informational, na.rm=TRUE))
web.df$Informational_Duration <- impute_data(web.df$Informational_Duration, mean(web.df$Informational_Duration, na.rm=TRUE))
web.df$ProductRelated <- impute_data(web.df$ProductRelated, mean(web.df$ProductRelated, na.rm=TRUE))
web.df$ProductRelated_Duration <- impute_data(web.df$ProductRelated_Duration, mean(web.df$ProductRelated_Duration, na.rm=TRUE))
web.df$BounceRates <- impute_data(web.df$BounceRates, mean(web.df$BounceRates, na.rm=TRUE))
web.df$ExitRates <- impute_data(web.df$ExitRates, mean(web.df$ExitRates, na.rm=TRUE))

summary(web.df$Administrative)  # Check again.
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.000   1.000   2.343   4.000  26.000       8
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   1.000   2.343   4.000  26.000
```

[17]:
```
# we do the same for testing dataset
webtesting.countNAs <- sapply(webtesting.df, countNAs)
webtesting.countNAs

# For those variables contain missing values, calculate the proportion of missing values.
round(webtesting.countNAs[webtesting.countNAs != 0]/nrow(webtesting.df),5)
```

ID: 0 **Administrative:** 6 **Administrative_Duration:** 6 **Informational:** 6 **Informational_Duration:** 6 **ProductRelated:** 6 **ProductRelated_Duration:** 6 **BounceRates:** 6 **ExitRates:** 6 **PageValues:** 0 **SpecialDay:** 0 **Month:** 0 **OperatingSystems:** 0 **Browser:** 0 **Region:** 0 **TrafficType:** 0 **VisitorType:** 0 **Weekend:** 0

Administrative: 0.00162 **Administrative_Duration:** 0.00162 **Informational:** 0.00162 **Informational_Duration:** 0.00162 **ProductRelated:** 0.00162 **ProductRelated_Duration:** 0.00162 **BounceRates:** 0.00162 **ExitRates:** 0.00162

[18]:
```
# We are supposed to remove columns with missing proportions greater than 0.5.
# However, as all the proportions are 0.00162, which is not significant, we keep all of them
summary(webtesting.df$Administrative)

# As shown in the table above, 0.162% of the data in 8 attributes are missing, respectively.
# Thus, we need to impute the missing data in these corresponding attributes.
# We replace the NAs in these 8 columns with their column means.
impute_data <- function(vec, mn){
    ifelse(is.na(vec), mn, vec)
}
webtesting.df$Administrative <- impute_data(webtesting.df$Administrative, mean(webtesting.df$Administrative, na.rm=TRUE))
webtesting.df$Administrative_Duration <- impute_data(webtesting.df$Administrative_Duration, mean(webtesting.df$Administrative_
webtesting.df$Informational <- impute_data(webtesting.df$Informational, mean(webtesting.df$Informational, na.rm=TRUE))
webtesting.df$Informational_Duration <- impute_data(webtesting.df$Informational_Duration, mean(webtesting.df$Informational_Dur
webtesting.df$ProductRelated <- impute_data(webtesting.df$ProductRelated, mean(webtesting.df$ProductRelated, na.rm=TRUE))
webtesting.df$ProductRelated_Duration <- impute_data(webtesting.df$ProductRelated_Duration, mean(webtesting.df$ProductRelated_
webtesting.df$BounceRates <- impute_data(webtesting.df$BounceRates, mean(webtesting.df$BounceRates, na.rm=TRUE))
webtesting.df$ExitRates <- impute_data(webtesting.df$ExitRates, mean(webtesting.df$ExitRates, na.rm=TRUE))

summary(webtesting.df$Administrative)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.000   1.000   2.259   3.000  27.000       6
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   0.000   1.000   2.259   3.000  27.000
```

3.2 One-hot-encoding

[20]:
```
# we do the same for testing dataset
library(caret)
library(lattice)
#define one-hot encoding function
dummy <- dummyVars(" ~ .", data=webtesting.df)

#perform one-hot encoding on data frame
finaltesting.df <- data.frame(predict(dummy, newdata=webtesting.df))
```

3.3 Train/Validation Split

[19]:
```
# 2.implement one-hot-encoding for categorical attribute - "Visitor Types"
library(caret)
library(lattice)
#define one-hot encoding function
dummy <- dummyVars(" ~ .", data=web.df)

#perform one-hot encoding on data frame
finaltraining.df <- data.frame(predict(dummy, newdata=web.df))
```

[21]:
```
# Split the training dataset into smaller trainging set and validation set.
```

[22]:
```
# Get the total numer of rows in the training data set.
nrowTrain <- nrow(finaltraining.df)

# Use 75% of the data to train the model (smaller training set).
nrowSmallTrain <- round(nrowTrain*0.75)

# The remaining 25% of the data is the validation set
nrowvalid <- nrowTrain - nrowSmallTrain

# Suppose we would like to randomly split the data.
set.seed(201)

# generate row index of the smaller training set.
rowIndicesSmallTrain <- sample(1:nrowTrain, size = nrowSmallTrain, replace = FALSE)
smalltrain.df <- finaltraining.df[rowIndicesSmallTrain, ]
valid.df <- finaltraining.df[-rowIndicesSmallTrain, ]
```

[23]:
```
# Number of entries
nrow(smalltrain.df)
nrow(valid.df)
nrow(finaltesting.df)
```

6472

2158

3700

Section 4. Model Evaluation

4.1 Set up a scoring rule to measure the accuracy of the model

[25]:
```
# Function to calculate the average Brier score.
brier_score <- function(predictions, realizations) {
```

[30]:
```
.libPaths("/usr/local/lib/R/site-library")
library(xgboost)
library(randomForest)
library(lubridate)
library(doSNOW)
library(foreach)
library(parallel)
```

as the prediction

[31]:
```
# print out column names
print(colnames(smalltrain.df))
```

```
 [1] "ID"                          "Administrative"
 [3] "Administrative_Duration"     "Informational"
 [5] "Informational_Duration"      "ProductRelated"
 [7] "ProductRelated_Duration"     "BounceRates"
 [9] "ExitRates"                   "PageValues"
[11] "SpecialDay"                  "MonthAug"
[13] "MonthDec"                    "MonthFeb"
[15] "MonthJul"                    "MonthJune"
[17] "MonthMar"                    "MonthMay"
[19] "MonthNov"                    "MonthOct"
[21] "MonthSep"                    "OperatingSystems"
[23] "Browser"                     "Region"
[25] "TrafficType"                 "VisitorTypeNew_Visitor"
[27] "VisitorTypeOther"            "VisitorTypeReturning_Visitor"
[29] "Weekend"                     "Revenue"
```

[32]:
```
# Since Set 1 includes 'PageValues', 'ExitRates', 'BounceRates', 'ProductRelated', 'ProductRelatedDuration', 'Administrative',
# and 'VisitorTypeNew_Visitor', predictors are column 2,6,7,8,9,10,26, and they will be further taken to train models.
# Revenue (column 30) is the response variable


smalltrainSet1.select <- smalltrain.df[,c(2,6,7,8,9,10,26,30)]
validSet1.select <- valid.df[,c(2,6,7,8,9,10,26,30)]
```

4.2 Configurations

4.2.1 Set 1 – Logistic Regression/ Regression Tree/ XGBoost

[34]:
```
# Logistic Regression
```

[48]:
```
# Print out feature importance
# Gain indicates the contribution of each feature. Higher gain means higher importance.
xgb.importance(colnames(xsmalltrain.xgb), model = xgb.trees)
```

4.2.2 Set 2

[51]:
```
# Logistic Regression

# Train logistic regression model with interactions
logit.reg <- glm(Revenue ~ . + .^2, data = smalltrainSet2.select, family = "binomial")
summary(logit.reg)
```

```
Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"

Call:
glm(formula = Revenue ~ . + .^2, family = "binomial", data = smalltrainSet2.select)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.5978  -0.4646  -0.3726  -0.2010   2.9921

Coefficients:
                                   Estimate Std. Error z value Pr(>|z|)
(Intercept)                      -2.116e+00  9.154e-02 -23.115  < 2e-16 ***
ProductRelated_Duration           2.688e-04  4.052e-05   6.634 3.27e-11 ***
ExitRates                        -1.765e+01  2.569e+00  -6.873 6.29e-12 ***
PageValues                        1.022e-01  7.164e-03  14.269  < 2e-16 ***
ProductRelated_Duration:ExitRates -4.472e-03  1.698e-03  -2.634  0.00843 **
ProductRelated_Duration:PageValues -1.263e-05  2.222e-06  -5.685 1.31e-08 ***
ExitRates:PageValues              1.526e-01  2.541e-01   0.601  0.54806
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5598.9  on 6471  degrees of freedom
Residual deviance: 3848.7  on 6465  degrees of freedom
AIC: 3862.7

Number of Fisher Scoring iterations: 7
```

[52]:
```
# Generate predictions in the validation set
pred.logit.reg <- predict(logit.reg, validSet2.select, type = 'response')
# Calculate the Brier Skill Score based on the interaction logistic regression model
skill.score(pred.logit.reg, validSet2.select$Revenue, brier.ref.valid)
```

```
0.348037345756424
```

[53]:
```
# Train regression tree model
reg.tree <- rpart(Revenue ~ ., data = smalltrainSet2.select,
                  control = rpart.control(cp = 0.0001))
```

[54]:
```
# Generate predictions in the validation set
reg.tree.pred <- predict(reg.tree,validSet2.select)
#reg.tree.pred
```

[55]:
```
# Calculate the Brier Skill Score
skill.score(reg.tree.pred, validSet2.select$Revenue, brier.ref.valid)
```

0.294423815616179

[56]:
```
# Train XGBoost Model
# XGBoost Model can only deal with numeric numbers/matrix, so we convert all variables to matrix first
xsmalltrain.xgb <- model.matrix(~ 0 + ., data = smalltrainSet2.select[,-4]) #IMPORTANT --> REMOVE TARGET VARIABLE, stores all pred
ysmalltrain.xgb <- as.vector(smalltrainSet2.select$Revenue) #target variables

xvalid.xgb <- model.matrix(~ 0 + ., data = validSet2.select[,-4])
yvalid.xgb <- as.vector(validSet2.select$Revenue)
```

[57]:
```
t1 <- proc.time()
xgb.trees <- xgboost(data=xsmalltrain.xgb,
                     label=ysmalltrain.xgb,
                     max.depth = 3,
                     #nthread = workers,
                     nround = 200,
                     objective = "binary:logistic",
                     verbose = 0)
proc.time() - t1
# Tune the model above by increasing/decreasing the depth of each tree and/or the number of trees (nround).
```

```
[14:12:21] WARNING: ../..//amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
   user  system elapsed
 44.845   0.085  13.367
```

[58]:
```
# Generate predictions in the validation set
xgb.trees.pred <- predict(xgb.trees, xvalid.xgb)
# Calculate the Brier Skill Score
skill.score(xgb.trees.pred, yvalid.xgb, brier.ref.valid)
```

0.383945573117775

Logistic Regression/ Regression Tree/ XGBoost

4.2.3 Set 3 – Logistic Regression/ Regression Tree/ XGBoost

[59]:
```
# Set 2 does not yield a better result, potentiall due to overlooking certain crucial features.
# Hence, we go back to feature importance, and try to improve the model by removing less significant feature one at a time
# to see if there is a significant improvement.

#First we remove VisitorTypeNew_Visitor (Column 26) as Set 3

smalltrainSet3.select <- smalltrain.df[,c(2,6,7,8,9,10,30)]
validSet3.select <- valid.df[,c(2,6,7,8,9,10,30)]
```

[60]:
```
head(smalltrainSet3.select)
```

A data.frame: 6 × 7

	Administrative	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	Revenue
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
7379	7	43	3283.722	0.009302326	0.019379845	24.901442	0
7551	4	30	1021.838	0.000000000	0.005882353	40.435790	0
2602	0	6	140.000	0.000000000	0.033333333	0.000000	0
8369	0	10	693.500	0.000000000	0.022222222	40.433778	1
2942	0	36	1388.200	0.005555556	0.027777778	0.000000	0
7669	17	31	1021.619	0.004545455	0.026479076	2.080534	1

[61]:
```
# Logistic Regression

# Train logistic regression model with interactions
logit.reg <- glm(Revenue ~ . + .^2, data = smalltrainSet3.select, family = "binomial")
#summary(logit.reg)
```

[62]:
```
# Generate predictions in the validation set
pred.logit.reg <- predict(logit.reg, validSet3.select, type = 'response')
# Calculate the Brier Skill Score based on the interaction logistic regression model
```

[69]:
```
# Set 3 (removing a attribute with the lowest feature importance) does improve BSS.
# We try to remove the attribute with the second lowest feature importance to see if BSS is improved further.
# So we remove Administrative (Column 2) as Set 4

smalltrainSet4.select <- smalltrain.df[,c(6,7,8,9,10,30)]
validSet4.select <- valid.df[,c(6,7,8,9,10,30)]
```

[70]:
```
# Logistic Regression

# Train logistic regression model with interactions
logit.reg <- glm(Revenue ~ . + .^2, data = smalltrainSet4.select, family = "binomial")
summary(logit.reg)
```

[71]:
```
# Generate predictions in the validation set
pred.logit.reg <- predict(logit.reg, validSet4.select, type = 'response')
# Calculate the Brier Skill Score based on the interaction logistic regression model
skill.score(pred.logit.reg, validSet4.select$Revenue, brier.ref.valid)
```

0.351022221791949

[72]:
```
# Train regression tree model
reg.tree <- rpart(Revenue ~ ., data = smalltrainSet4.select,
                  control = rpart.control(cp = 0.0001))
```

[73]:
```
# Generate predictions in the validation set
reg.tree.pred <- predict(reg.tree,validSet4.select)

# Calculate the Brier Skill Score
skill.score(reg.tree.pred, validSet4.select$Revenue, brier.ref.valid)
```

0.275808350100779

n the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

[68]:
```
# Generate predictions in the validation set
xgb.trees.pred <- predict(xgb.trees, xvalid.xgb)
# Calculate the Brier Skill Score
skill.score(xgb.trees.pred, yvalid.xgb, brier.ref.valid)
```

0.42627335833578

4.2.4 Set 4 – Logistic Regression/ Regression Tree/ XGBoost

4.3

[74]:
```
Train XGBoost Model
```

[77]:
```
# Ensemble model - Stacking
```

[78]:
```
# 1. Train the base models (M1, M2 and M3) by using the smaller training set,
#    and generate predictions from these three models in the validation set.
# 2. Fit the stacker model (regression tree or XGBoost) to the predictions generated in Step 1.
#    The dependent variable is Revenue in the training set and the independent variables are the predictions obtained in Step 1.
```

[79]:
```
# Logistic regression
M1 <- glm(Revenue ~  . + .^2, data = smalltrainSet3.select, family = "binomial")
M1.predict <- predict(M1, valid.df, type = "response")
```

```
Warning message:
“glm.fit: fitted probabilities numerically 0 or 1 occurred”
```

[80]:
```
# Regression tree
M2 <- rpart(Revenue ~ .,
            data = smalltrainSet3.select,
            control = rpart.control(cp = 0.0001))
M2.predict <-  predict(M2, valid.df)
```

[87]:
```
# XGBoost
xtrain.xgb <- model.matrix(~ 0 + ., data = smalltrainSet3.select[,-7])
ytrain.xgb <- as.vector(smalltrainSet3.select$Revenue)

xvalid.xgb <- model.matrix(~ 0 + ., data = validSet3.select[,-7])
yvalid.xgb <- as.vector(validSet3.select$Revenue)

M3 <- xgboost(xtrain.xgb, ytrain.xgb,
              max.depth = 3,
              #nthread = workers,
              nround = 200,
              objective = "binary:logistic",
              verbose = 0)

M3.predict <- predict(M3, xvalid.xgb)
```

```
[14:21:57] WARNING: ../..//amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

Ensemble Model

[88]:
```
# 2. Fit a stacker model to the predictions generated in last step.
```

[93]:
```
# Stacking model 2 - XG Boost
```

[94]:
```
# Convert stacker.df to matrix format
stacker.x.xgb <- model.matrix(~ 0 + ., data = stacker.df[,-1])
stacker.y.xgb <- as.vector(stacker.df[,1])

stackerModel_2 <- xgboost(stacker.x.xgb, stacker.y.xgb,
                          max.depth = 3,
                          #nthread = workers,
                          nround = 20,
                          objective = "binary:logistic",
                          verbose = 0)
# Different settings of max.depth and nround have been examined.
# max.depth = 3 and nround = 20 provide the highest Brier Skill Score
```

```
[14:22:31] WARNING: ../..//amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
```

[96]:
```
# Predict from stacker model 2 --- XGBoost, and calculate the Brier Skill Score
predict.variables.xgb <- model.matrix(~ 0 + ., data = stacker.df[, -1])
stacker.predict.xgb <- predict(stackerModel_2, predict.variables.xgb)

# Score the stacker model's prediction
skill.score(stacker.predict.xgb, validSet3.select$Revenue, brier.ref.valid)
```

0.527222168279419

[98]:
```
bss.df <- data.frame("Model" = c("Logistic Regression", "Regression Tree", "XGBoost", "Stacking Regression Tree", "Stacking XGBoo
                     "BSS" = c(0.376077, 0.298002, 0.426273, 0.604422, 0.527222))
bss.df
```

A data.frame: 5 × 2

Model	BSS
<chr>	<dbl>
Logistic Regression	0.376077
Regression Tree	0.298002
XGBoost	0.426273
Stacking Regression Tree	0.604422
Stacking XGBoost	0.527222

[106]:
```
# Train Stacker model 1: regression tree
# Dependent variable is the Revenue, independent variables are M1, M2, M3 predictions
best.stackerModel <- rpart(Revenue ~ .,
                           data = best.stacker.df,
                           control = rpart.control(cp = 0.0004))
```

[107]:
```
# Predict from our best stacker model --- regression tree
best_stacker.predict.rt <- predict(best.stackerModel, best.stacker.df)
best_stacker.predict.rt
```

1: 0.0666666666666667 **2:** 0.000969367972082202 **3:** 0.000969367972082202 **4:** 0.000969367972082202 **5:** 0.000969367972082202 **6:** 0.000969367972082202 **7:** 0.0666666666666667 **8:** 0.000969367972082202 **9:** 0.0261437908496732 **10:** 0.000969367972082202 **11:** 0.933333333333333 **12:** 0.000969367972082202 **13:** 0.000969367972082202 **14:** 0.962365591397849 **15:** 0.993814432989691 **16:** 1 **17:** 0 **18:** 0.000969367972082202 **19:** 0.000969367972082202 **20:** 0.000969367972082202 **21:** 0.938461538461538 **22:** 0.000969367972082202 **23:** 0.000969367972082202 **24:** 0.000969367972082202 **25:** 0.000969367972082202 **26:** 0.916666666666667 **27:** 0.993814432989691 **28:** 0 **29:** 0.0149253731343284 **30:** 0.000969367972082202 **31:** 0.0149253731343284 **32:** 0.993814432989691 **33:** 0.000969367972082202 **34:** 0 **35:** 0.000969367972082202 **36:** 0.105263157894737 **37:** 0.000969367972082202 **38:** 0.000969367972082202 **39:** 0.000969367972082202 **40:** 0.000969367972082202 **41:** 0.000969367972082202 **42:** 0.993814432989691 **43:** 0.000969367972082202 **44:** 0.000969367972082202 **45:** 0 **46:** 0.993814432989691 **47:** 0.000969367972082202 **48:** 0.0421052631578947 **49:** 0.142857142857143 **50:** 0.000969367972082202 **51:** 0.000969367972082202 **52:** 0.000969367972082202 **53:** 0.993814432989691 **54:** 0.000969367972082202 **55:** 0.000969367972082202 **56:** 0.000969367972082202 **57:** 0 **58:** 0.993814432989691 **59:** 0.000969367972082202 **60:** 0.000969367972082202 **61:** 0.000969367972082202 **62:** 0.0261437908496732 **63:** 0.416666666666667 **64:** 0.0261437908496732 **65:** 0.000969367972082202 **66:** 0.000969367972082202 **67:** 0.000969367972082202 **68:** 0.363636363636364 **69:** 0.0149253731343284 **70:** 0.000969367972082202 **71:** 0 **72:** 0.962365591397849 **73:** 0.307692307692308 **74:** 0 **75:** 0.000969367972082202 **76:** 0.000969367972082202 **77:** 0.428571428571429 **78:** 0.000969367972082202 **79:** 0.000969367972082202 **80:** 0.000969367972082202 **81:** 0.000969367972082202 **82:** 0.5625 **83:** 0.000969367972082202 **84:** 0.000969367972082202 **85:** 0.000969367972082202 **86:** 0 **87:** 0.000969367972082202 **88:** 0.000969367972082202 **89:** 0.000969367972082202 **90:** 0.000969367972082202 **91:** 0.000969367972082202 **92:** 0.0666666666666667 **93:** 0.000969367972082202 **94:** 0.000969367972082202 **95:** 0.000969367972082202 **96:** 0.6 **97:** 0.000969367972082202 **98:** 0.962365591397849 **99:** 0.000969367972082202 **100:** 0.000969367972082202 **101:** 0.28125 **102:** 0.000969367972082202 **103:** 0.000969367972082202 **104:** 0.555555555555556 **105:** 0.000969367972082202 **106:** 0 **107:** 0.938461538461538 **108:** 0 **109:** 0.000969367972082202 **110:** 0.08 **111:** 0 **112:** 0.0666666666666667 **113:** 0.000969367972082202 **114:** 0.0666666666666667 **115:** 0.000969367972082202 **116:** 0 **117:** 0.75 **118:** 0.307692307692308 **119:** 0.938461538461538 **120:** 0.000969367972082202 **121:** 0 **122:** 0.993814432989691 **123:** 0.000969367972082202 **124:** 0.000969367972082202 **125:** 0.962365591397849 **126:** 0.000969367972082202 **127:** 0.444444444444444 **128:** 0.000969367972082202 **129:** 0.0666666666666667 **130:** 0.000969367972082202 **131:** 0.000969367972082202

4.4 Final prediction on testing set

[105]:
```
# Construct the stacker dataframe in testing set
best.stacker.df <- data.frame(Revenue = finaltraining.df$Revenue,
                              M1.predict.best = M1.predict.best,
                              M2.predict.best = M2.predict.best,
                              M3.predict.best = M3.predict.best)

best.stacker.df
```

A data.frame: 8630 × 4

	Revenue	M1.predict.best	M2.predict.best	M3.predict.best
	<dbl>	<dbl>	<dbl>	<dbl>
1	0	0.090045141	0.0092592593	0.0655927360
2	0	0.110794331	0.0046082949	0.0071820375
3	0	0.084179326	0.0092592593	0.0152244316
4	0	0.025479730	0.0000000000	0.0068338574
5	0	0.055174251	0.0071174377	0.0150805227
6	0	0.022730503	0.0000000000	0.0053112828
7	0	0.082200987	0.0093457944	0.0604529828
8	0	0.063296378	0.0000000000	0.0438194275
9	0	0.974622353	0.3571428571	0.2206926346
10	0	0.000626521	0.0009225092	0.0002591671
11	1	0.779801378	1.0000000000	0.6092999578
12	0	0.045333984	0.0107526882	0.0143068926
13	0	0.000626521	0.0009225092	0.0002591671
14	1	0.110186380	0.8181818182	0.6735302806
15	1	0.993649426	0.8235294118	0.9487466216
16	1	0.129954105	0.8000000000	0.5346331596
17	0	0.099140895	0.0000000000	0.0657188147
18	0	0.050127300	0.0000000000	0.0049783071