E-Commerce Visitor Buying Intention

Summary of Work

The task was to train 3 machine-learning models to perform binary classification on a dataset containing features describing the activities of visitors to an e-commerce website. The final goal was to be able to predict, using the data given, whether a visitor would purchase an item (**Revenue=TRUE**) or not.

Data Exploration

As with any machine-learning project, the first port-of-call was to explore the data. This was simply done by firstly observing the features, ensuring understanding of their meaning and noting the range of values they take as well as their **Dtype**. It was doubled-checked that all features did not have any missing values. Simple histograms of each feature were plotted to visualise the distribution of the data and a correlation matrix created to find any correlations between the features. For features with large ranges such as **ProductRelated_Duration** boxplots were used to identify any points which could be considered outliers.

Data Pre-Processing

Next was to prepare the data for feeding into the machine-learning models. From the earlier

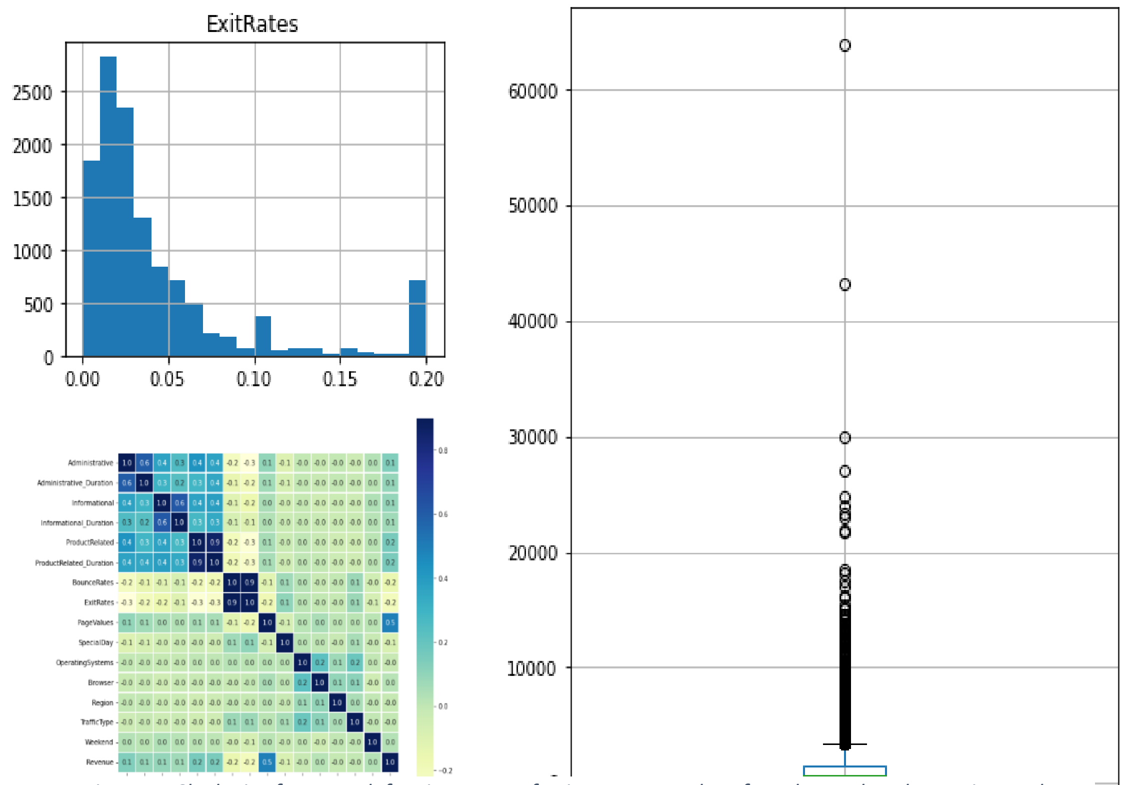

Figure 1: Clockwise from top-left. Histogram of Exit Rates, Boxplot of Product Related Duration and Correlation Matrix

data exploration, it was known that 2 features had **Dtype Boolean** and 2 had **Dtype Object.**

These would have to be changed to numeric, the former simply by multiplying the columns by 1. The latter however would need to be passed through the **OneHotEncoder**.

Data was split into its target and feature data (**Revenue** and **data_x**)

A categorical transformer Pipeline was used to apply the **OneHotEncoder** to the **Dtype object** categories ‘**Month’** and ‘**VisitorType’.**

A numeric transformer pipeline was also used to apply **StandardScaler** to features with large ranges of values. These were chosen to be:

- **“Administrative_Duration"**
- **"Informational_Duration"**
- **"ProductRelated"**
- **"ProductRelated_Duration"**
- **"PageValues"**

This newly processed data was assigned to an object and printed to check the pipeline formatted the data correctly.

Following the pipeline, the **RandomUnderSampler** was used the balance our skewed data. It was identified in the Data Exploration phase that for **Revenue** the data was distributed such that:

```
False    10422
True      1908
```

Using **RandomUnderSampler** It was made that the **10442** counts of **FALSE** were randomly reduced to **1908** to match the count of **TRUE.** *1 being TRUE and 0 being FALSE.*

```
1          1908
0          1908
```

Finally before implementing a model, the data was split into training and testing sets in the ratio 7:3 using **train_test_split.** This was to ensure the models were only tested on unseen data to provide a truly accurate evaluation of its performance.

Implementing Machine Learning Models

3 machine-learning models were to be chosen to ensure a good representation of various classification techniques, the models chosen were:

1. **K-Nearest Neighbors**
2. **Decision Tree**
3. **Support Vector Machine**

K-Nearest Neighbors was chosen because it is simple and intuitive. The algorithm responds quickly to changes and is easy to implement.

Decision Tree was chosen because it is a unique classifier and very intuitive. Although data was processed before reaching the model implementation stage, decision trees don't require scaling or normalization of data and handles missing values well.

Support Vector Machine was chosen because it is effective in high dimensional spaces and relatively memory efficient.

Before running any of the above models, a 'dumb' model was tested for a baseline of the scores one would want to better using more sophisticated models. The **BaseEstimator** guessed **FALSE** for every test.

"classification_test" *and* ***"confusion_matrix"*** *were used to evaluate.*

classification_test:

```
              precision    recall  f1-score   support

           0       0.52      1.00      0.68       590
           1       0.00      0.00      0.00       555

    accuracy                           0.52      1145
```

confusion_matrix:

```
[[590   0]
 [555   0]]
```

K-Nearest Neighbors

KNN predicts the correct class of a datapoint by calculating the distance between that point and the other training points. It classifies that point as the class that has the most points closest to the point.

classification_test:

```
              precision    recall  f1-score   support

           0       0.83      0.79      0.81       590
           1       0.79      0.82      0.81       555
    accuracy                           0.81      1145
```

confusion_matrix:

```
[[469 121]
 [98  457]]
```

Cross-validation scores tell us more confidently how accurate our model is as it avoids overfitting. “**cross_val_score**” works by creating a fixed number of folds of the data, running analysis on each fold, and then averaging the overall error estimate.

cross_val_score :

```
array([0.79, 0.76, 0.81, 0.78, 0.77])
```

Decision Tree

The Decision Tree model works by recursively splitting the dataset until we are left with pure leaf nodes (data with only one type of class).

classification_test:

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86       590
           1       0.86      0.82      0.84       555

    accuracy                           0.85      1145
```

confusion_matrix:

```
[[518  72]
 [98 457]]
```

cross_val_score :

```
array([0.83, 0.83, 0.88, 0.86, 0.85])
```

Support Vector Machine

Support Vector Machines work by creating a hyperplane of greatest margin to separate 2 classes, in this case **TRUE** and **FALSE.** The results…

classification_test:

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       590
           1       0.86      0.78      0.82       555

    accuracy                           0.83      1145
```

confusion_matrix:

```
[[521  69]
 [123 432]]
```

cross_val_score :

```
array([0.81, 0.82, 0.83, 0.85, 0.82])
```

Hyperparameter Testing

GridSearchCV was used to find the optimal parameters for each model. It was not found to significantly increase accuracy in any of the models. Hyperparameter tuning could be explored in greater depth, but would require greater computing time

```
KNN best estimator:                KNeighborsClassifier(neighbors=29)
KNN best score:                    0.8068178795197591
KNN best parameters:               {'n_neighbors': 29}


Decision Tree best estimator:      DecisionTreeClassifier(criterion='ent
                                   ropy', max_depth=3, random_state=42)
Decision Tree best score:          0.8536237352563028
                                   {'criterion': 'entropy',
Decision Tree best parameters:     'max_depth': 3}


SVM best estimator:                SVC(C=1, probability=True)
SVM best score:                    0.829655920753264
SVM best parameters:               {'C': 1, 'kernel': 'rbf'}
```

Analysis

An F1 score is a measure of a tests accuracy calculated from the precision and recall. The F1 score therefore favours classifiers with similar precision and recall as it is a harmonic average of the two. Precision and recall have a trade-off, one cannot increase without the other decreasing. One could find the optimum point at which these two are at their greatest without detriment to the other. One could also decide, within the context of their project, whether precision or recall is more important and if one should have a precedence over the other.

Taking F1 scores to be a good evaluator of the models, Decision Tree would be the model to implement permanently as it scored 0.85 and also had the highest recall for instances of **Revenue=TRUE.** In fact, the Decision Tree model was not beaten in **any** score in the classification table. In the confusion matrix, Decision Tree was only beaten by the SVM's ability to predict instances of **Revenue=FALSE** at a slightly greater recall**.**

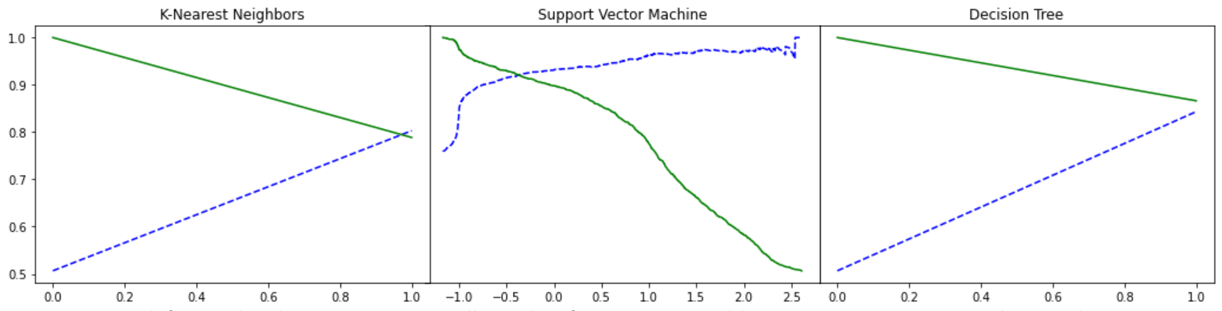

Figure 2: From left to right, the precision vs recall graphs of K-Nearest Neighbours, Support Vector Machine and Decision Tree

The receiver operating characteristic (**ROC**) curve plots recall against the false positive rate (sensitivity vs specificity) across all possible classification thresholds. Visually, models which perform better are those closest to the top left corner of the graph, to compare models more accurately we can measure the area under the curve (**AUC**). Perfect classifier = 1, random classifier = 0.5 (the dotted line).

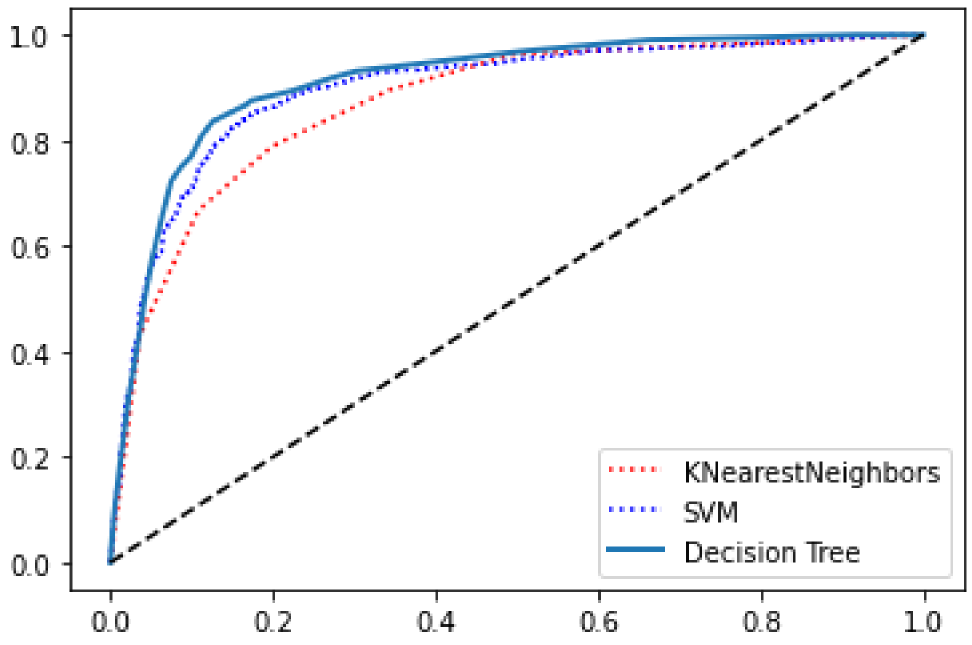

Figure 3: ROC curve for K-Nearest Neighbors, Support Vector Machine and Decision Tree with a black, dashed line to signify 0.5 AUC or a Random Classifier.

```
   K-Nearest Neighbors AUC:  0.8714333460067942
Support Vector Machine AUC:  0.8967056852248754
         Decision Tree AUC:  0.9116143858362297
```

Although graphically very similar, the **Decision Tree** model has the greatest **AUC** with **K-Nearest Neighbors** having the worst score.

<u>Conclusions</u>

Evaluation through **F1 scores** as well as **ROC AUC** would suggest that the **Decision Tree** would be the most accurate of the 3 models to predict whether a website visitor will purchase an item or not. However, a small change in the data can cause a large change in the structure of a decision tree causing instability. Decision Trees are also inadequate for predicting continuous values (regression) should this need to be applied in the future.

Further steps to possibly improve the performance of the models would be to remove outliers from the data and select only features believed to have an impact on the outcome of **Revenue.** Instead of scaling the data down to match that of **Revenue=TRUE,** it may be of interest to use **RandomOverSampler** to gain larger training and test datasets. Perhaps creating new features by combining existing ones would unlock a new, powerful correlation to **Revenue** which could predict outcomes more reliably. Ensemble Methods could also be used.

However, as the models stand now an 85% accuracy in prediction is far more sophisticated than the ‘dumb’ classifier we started with and the precision of all of the models could be increased immediately with sacrifice to the recall. If this was a real-world project, that would be the next question for the client; ‘would you prefer greater precision or recall?’.