National College of Ireland

Customer Behaviour Prediction Using Recommender Systems

MSc Research Project
Data Analytics

Ifeoma Delphine Onyeka
Student ID: x20189231

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ifeoma Delphine Onyeka
Student ID:	x20189231
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	15/08/2022
Project Title:	Customer Behaviour Prediction Using Recommender Systems
Word Count:	XXX
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ifeoma Delphine O
Date:	13th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Customer Behaviour Prediction Using Recommender Systems

Ifeoma Delphine Onyeka
x20189231

1 Introduction

Our configuration handbook acts as a roadmap for this research. It provides a summary of the hardware and software requirements needed to run the programs from the step of preparing the data to the step of implementation. It is essential that you have jupyter installed, a Windows system with 8 GB of RAM, and an i5 processor with GPU in order to implement this project. Considering that this project was developed in Python 3.6, a version that is compatible with it is required.

Research Study: Prediction of customer behaviour using recommender systems on a historical data. This study's major objective is to evaluate how well various machine learning models are being used.

2 System Configuration

2.1 Hardware

The settings utilized for this research are displayed below.:

- Model : HP
- OS : Windows 10 Operating System
- Processor : 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
- Memory : 8.00 GB
- Number of Core : 4

2.2 Software

In this case, Python was used as the programming language.We use Google Chrome as our web browser. In addition, the report documentation was done with Overleaf software. After openning the jupyter, all required libraries are needed to be imported that will be used for the implementation. which are:

- Numpy
- datetime

- pandas
- matplotlib
- seaborn

Imported libraries are shown below:

```
# Importing the the libraries

import datetime
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# import the models
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

Figure 1: Importing the libraries

```
#import classification_report
from sklearn.metrics import classification_report
```

Figure 2: Importing the libraries

```
#import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Figure 3: Importing the libraries

3 Data Preparation

An overview of the process of uploading the dataset to a JUPYTER notebook is provided in this section. Immediately after reading in the data, we begin the data preprocessing, which includes cleaning and transforming the data. There were three datasets used for this study and they were obtained from Kaggle ; the behaviour data(events) , also property dataset and finally category dataset. The dataset provides information about recommending products to customers. The data here did not contain any missing values at this point. After mounting the dataset on my hard drive, the python code was used to read the dataset into the environment. Furthermore, the figure shows that there were no missing values in the data when it was inspected for missing values, so the data didn't require much cleaning.

```python
# Taking a look at the head and the tail first five and last five rows in event dataframe
events_df.head()
```

	timestamp	visitorid	event	itemid	transactionid
0	1433221332117	257597	view	355908	NaN
1	1433224214164	992329	view	248676	NaN
2	1433221999827	111016	view	318965	NaN
3	1433221955914	483717	view	253185	NaN
4	1433221337106	951259	view	367447	NaN

```python
events_df.tail()
```

	timestamp	visitorid	event	itemid	transactionid
2756096	1438398785939	591435	view	261427	NaN
2756097	1438399813142	762376	view	115946	NaN
2756098	1438397820527	1251746	view	78144	NaN
2756099	1438398530703	1184451	view	283392	NaN
2756100	1438400163914	199536	view	152913	NaN

Figure 4: Events Dataset

```python
# Taking a look at the head and the tail first five in category dataframe
category_tree_df.head()
```

	categoryid	parentid
0	1016	213.0
1	809	169.0
2	570	9.0
3	1691	885.0
4	536	1691.0

```python
# Taking a look at the head and the last five rows in category dataframe

category_tree_df.tail()
```

	categoryid	parentid
1664	49	1125.0
1665	1112	630.0
1666	1336	745.0
1667	689	207.0
1668	761	395.0

Figure 5: Category Datatset

```
# Taking a look at the head first five rows in property dataframe

item_properties_1_df.head()
```

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566
3	1431226800000	59481	790	n15360.000
4	1431831600000	156781	917	828513

```
item_properties_2_df.head()
```

	timestamp	itemid	property	value
0	1433041200000	183478	561	769062
1	1439694000000	132256	976	n26.400 1135780
2	1435460400000	420307	921	1149317 1257525
3	1431831600000	403324	917	1204143
4	1435460400000	230701	521	769062

Figure 6: Items_Property

As part of the data preprocessing, we have to convert the epoch time to a readable format as shown in figure 7

```
# Converting the UNIX/ Epoch time to a readabale format

unix_time = int("1433220801")
readable_time = datetime.datetime.fromtimestamp(unix_time)
readable_time.strftime("%Y-%m-%d %H:%M:%S")
```

```
'2015-06-02 05:53:21'
```

Figure 7: Converting the Unix

3.1 Exploaratory Data Analysis

After retrieving the data from the source, exploratory data analysis is done to identify the relationships between the variables. Figure 8 below displays a pie chart of the target variable in the dataset. There are three classes, View, Add to Cart and Transaction. From the figure, 96 percent of the customers viewed the product, 2.5 percent added the product to cart and finally, 0.8 percent purchased the product.

Figure 9 shows a bar chart of the most viewed items. .

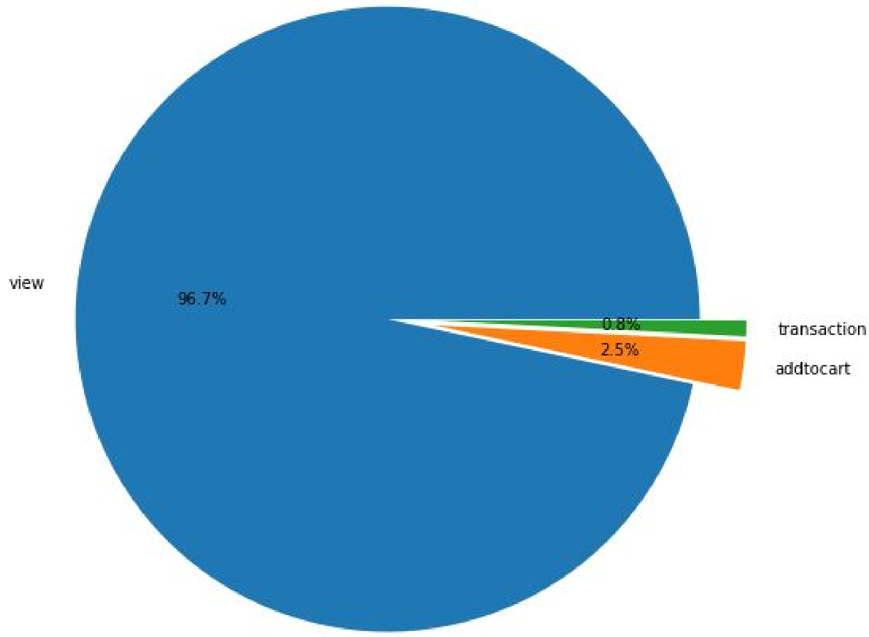

Figure 8: Target Variable

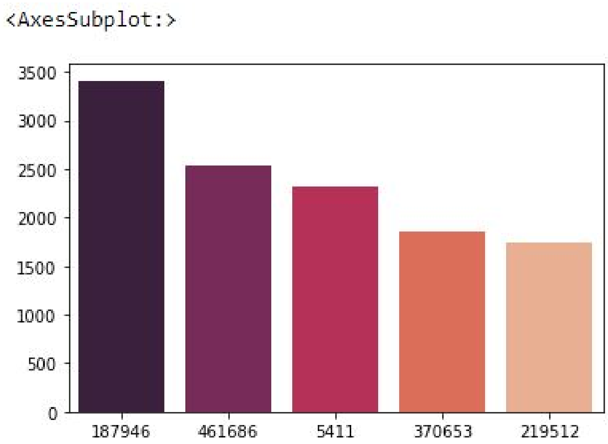

Figure 9: Top 5 Most VIEWED items

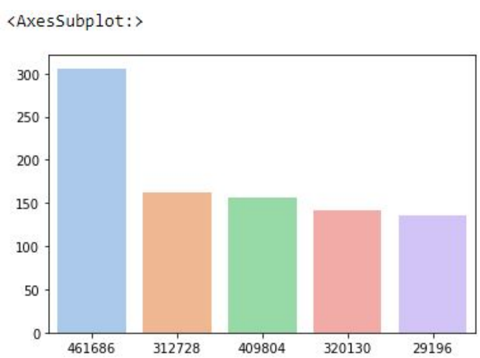

Figure 10: Top 5 items that were added to cart

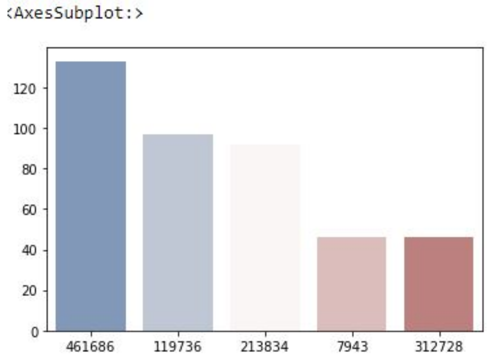

Figure 11: Top 5 Items with most TRANSACTION

4 Implementation

4.1 Model Building

A description of the models that were used in the research can be found in this section of the report. The codes for the models are shown here, along with evaluations of the results of each model. The models used in this research include

- Random Forest
- K-Nearest Neighbour
- Logistic Regression
- Support Vector Machine

In order to improve the performance of the K-Nearest Neighbour, hyperparameter tuning was done after evaluating the models. The dataset was split into 70 for training and 30 for test. The plot below clearly indicates that the higher the view count and the higher the chances of a visitor buying something.

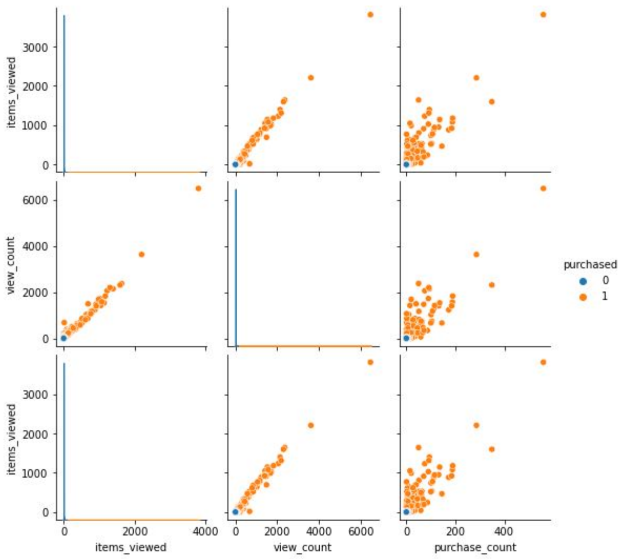

Figure 12: Plot showing the view count and the chances of purchase

4.2 Logistic Regression

SKLearn.linear.model LogisticRegression was used to train the logistic model.

```
X = combine_df.drop(['purchased', 'visitorid', 'purchase_count'], axis = 'columns')
y = combine_df.purchased

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42, train_size = 0.7)

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

LogisticRegression()

# Let's now use the model to predict the test features
y_pred_class = logreg.predict(X_test)

print('accuracy = {:7.4f}'.format(metrics.accuracy_score(y_test, y_pred_class)))

accuracy =  0.8011
```

Figure 13: Logistic Regression Model

```
# Plot the ROC Curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw = lw, label = 'ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc = "lower right")
plt.show()
```

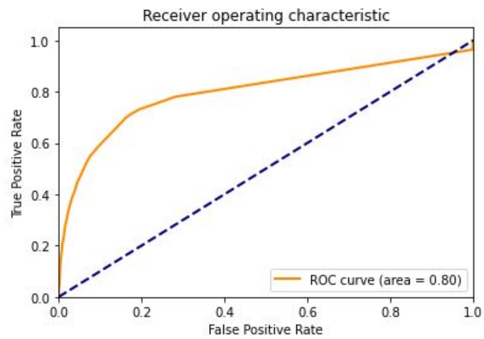

Figure 14: ROC Curve for Logistic Regression

4.3 K-nearest Neigbour

The number of neighbours is tuned for this experiment with the highest number of neighbours yielding the best accuracy.

```
# We can observe above that we get maximum testing accuracy for k=6.
#So lets create a KNeighborsClassifier with number of neighbors as 6.

knn.fit(X_train,y_train)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=6, p=2,
           weights='uniform')

KNeighborsClassifier(n_jobs=1, n_neighbors=6)

#Get accuracy. Note: In case of classification algorithms score method represents accuracy.
knn.score(X_test,y_test)

0.81200472095768
```

Figure 15: KNN Result

4.4 Support Vector Machine

The first experiment is done with default parameters and the second is done tuning the C parameter to 0.1 and the kernel being linear. This was applied to see if a different result will be given.

Running SVM with default hyperparameter

```
from sklearn.svm import SVC
from sklearn import metrics
svc=SVC() #Default hyperparameters
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.8019726858877086
```

Figure 16: default hyperparameter

default Linear Kernel

```
svc=SVC(kernel='linear')
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy Score:')
print(metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy Score:
0.8072837632776935
```

Figure 17: Default Kernel

SVM by taking hyperparameter C=0.1 and kernel as linear

```
from sklearn.svm import SVC
svc= SVC(kernel='linear',C=0.1)
svc.fit(X_train,y_train)
y_predict=svc.predict(X_test)
accuracy_score= metrics.accuracy_score(y_test,y_predict)
print(accuracy_score)
```

```
0.8105715730905412
```

Figure 18: Tuning the C parameter to 0.1

With K-fold cross validation(where K=10)

```
from sklearn.model_selection import cross_val_score
svc=SVC(kernel='linear',C=0.1)
scores = cross_val_score(svc, X, y, cv=10, scoring='accuracy')
print(scores)
```

```
[0.80576631 0.80854831 0.80374305 0.80273141 0.79564997 0.79969651
 0.8113303  0.80146687 0.80576631 0.80925879]
```

Figure 19: K Fold

4.5 Random Forest

Random forest classifer was imported with its criterion assigned as entropy, random_state to zero.To check accuracy of the random forest model,confusion matrix is used and obtained an accuracy of 82%

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
#predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)

accuracy_score(y_test, y_pred)
```

```
[[7860  574]
 [1535 1893]]

0.8222053616590794
```

Figure 20: Random Model