

```
In [24]: import zipfile
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import kaggle
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from datetime import datetime
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelBinarizer,
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.under_sampling import NearMiss
from numpy import where
import pickle

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix, classification_rep
from sklearn.model_selection import GridSearchCV, validation_curve
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns
from dask.distributed import Client
```

Business Understanding

In the last decade, e-commerce has fundamentally changed how we live our lives through how we shop. Companies such as Sears have gone bankrupt over the years, making the transition from brick and mortar to an online e-commerce marketplace, however other companies such as Chewy, have been able to exploit e-commerce to become a market leader in their category.

A study by emarketer.com found that the pandemic has had beneficial effects on US e-commerce. Sales will reach \$794.50 billion this year, up 32.4% year-over-year. That's a much higher growth rate than the 18.0% predicted in our Q2 forecast, as consumers continue to avoid stores and opt for online shopping amid the pandemic. By the end of the year e-commerce sales will reach 14.4% of all US retail spending for the year and 19.2% by 2024. If you further dig into the data and exclude gas and auto sales (categories sold almost exclusively offline), ecommerce penetration jumps to 20.6%.(1)

With e-commerce growing at such an unprecedented rate, many companies are to capitalise on this change in consumer behaviour. It comes as no surprise to many that purchasing items online is a different process from buying an item in a store. While in a store an employee can help guide a customer to items they are both looking for and items they may want to consider purchasing, many e-commerce marketplaces dont have the same leverage; it's much easier to close out of a "You should buy" popup, rather than to ignore the advice of an instore expert.

This has presented a serious challenge to e-commerce stores in the form of cart abandonment. Cart abandonment is when a customer leaves without buying, after adding an item to their cart. It is a trend that has remained steady since e-commerce entered the mainstream, as seen in the below chart from statista.



Find more statistics at [Statista](#)

The good news for data scientists is that during an in store purchase a customer can have a large degree of privacy, while almost every move an online customer makes is tracked and stored in a series of databases. This data can be used to produce insights into customer purchasing behaviour, and spending patterns. Using the "eCommerce Events History in Cosmetics Shop" on kaggle (3) <https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop> we plan to analyse the purchasing patterns in the five month history the dataset provides and predict whether a customer is going to remove an item from their cart. With this knowledge a company can then provide incentives, such as free shipping or discounts to turn that removal into a purchase.

Data Understanding

The kaggle e-commerce dataset contains behavior data for five months (Oct 2019 – Feb 2020) from a medium sized unnamed cosmetics online store. Each row in the file represents an online event or action. All events are related to products and users.

The dataset contains the following features:

1. `event_time` : Time when event happened at (in UTC).
2. `event_type` : What action a customer took. For more information please see below.
3. `product_id` : ID of a product
4. `category_id` : Product's category ID
5. `category_code` : Product's category taxonomy (code name) if it was possible to make it. Usually present for meaningful categories and skipped for different kinds of accessories.
6. `brand` : Downcased string of brand name. Can be a nan value, if it was missed.
7. `price` : Float price of a product. Present.
8. `user_id` : Permanent user ID.
9. `user_session` : Temporary user's session ID. Same for each user's session. This value is changed every time user come back to online store from a long pause.

`event_type` is further broken up into four components, these are:

1. view - a user viewed a product
2. cart - a user added a product to shopping cart
3. remove_from_cart - a user removed a product from shopping cart
4. purchase - a user purchased a product

An example of a purchase funnel, may be three chronological rows, with the rows sharing `user_session` and `user_id` values. Where the first row is a view, the second is cart, and the final is a purchase.

Mission

Using a ecommerce dataset from a cosmetic store on Kaggle, I set out to build a machine learning model that can predict whether an item will be purchased or removed from the cart on an e-commerce store. This will be used to help in marketing and sales. On the supply chain side it will also give more insights into how a product is viewed by customers and how much a brand/product should be stocked or not.

Importing and preparing the data

First we need to read the CSVs from our data folder and turn them into Pandas DataFrames for functionality. For a list of instructions on how to download the data form kaggle please see `"../data/data_download.ipynb"`

```
In [25]: #Convert csv to DataFrame
oct19 = pd.read_csv("../data/2019-Oct.csv")
nov19 = pd.read_csv("../data/2019-Nov.csv")
dec19 = pd.read_csv("../data/2019-Dec.csv")
jan20 = pd.read_csv("../data/2020-Jan.csv")
feb20 = pd.read_csv("../data/2020-Feb.csv")
```

```
In [26]: oct19.head()
```

```
Out[26]:
```

	event_time	event_type	product_id	category_id	category_code	brand	price	u
--	------------	------------	------------	-------------	---------------	-------	-------	---

	event_time	event_type	product_id	category_id	category_code	brand	price	u
0	2019-10-01 00:00:00 UTC	cart	5773203	1487580005134238553	NaN	runail	2.62	4632
1	2019-10-01 00:00:03 UTC	cart	5773353	1487580005134238553	NaN	runail	2.62	4632
2	2019-10-01 00:00:07 UTC	cart	5881589	2151191071051219817	NaN	lovely	13.48	4296
3	2019-10-01 00:00:07 UTC	cart	5723490	1487580005134238553	NaN	runail	2.62	4632
4	2019-10-01 00:00:15 UTC	cart	5881449	1487580013522845895	NaN	lovely	0.56	4296

Now we have to combine each DataFrame that represents a month into one DataFrame that holds all the months data.

```
In [27]: ecom_df = pd.concat([oct19, nov19, dec19, jan20, feb20], axis=0)
ecom_df.reset_index(inplace=True)
print(ecom_df.shape)
```

```
(20692840, 10)
```

Dropping columns here to both save memory later on, also they are irrelevant to our data as we are only looking at purchased, or removed from cart event_types.

THINK WHAT KIND OF DATA WE WOULD GET IN THE FUTURE?!

```
In [28]: ecom_df = ecom_df[ecom_df.event_type != 'view']
ecom_df = ecom_df[ecom_df.event_type != 'cart']
print(ecom_df.shape)
```

```
(5266686, 10)
```

Data Cleaning

Before we start working with the data we need to make sure the dataset is in a workable state.

NAs

Renaming unknown values in the brand column with the string 'Unknown'.

```
In [29]: ecom_df.brand.fillna(value="unknown", axis=0, inplace=True)
```

Dropping the remaining NA's in the dataset.

```
In [30]: ecom_df.dropna(inplace=True)
```

Tidying the DataFrame

create empty env open notebooks or start installing things you need. (conda install matplotlib lib example) start creating dependencies, and just make sure the notebook runs on that environment.

Dropping category_id, as we will use category code instead. It is redundant.

```
In [31]: ecom_df.drop("category_id", axis=1, inplace = True)
```

Removing any negative prices that appear in the dataset.

```
In [32]: ecom_df = ecom_df[ecom_df['price'] >= 0]
```

Setting our target variable, event_type to binary.

```
In [33]: ecom_df.event_type = ecom_df["event_type"].replace({'purchase': 0, 'remove_from_
```

Splitting our data

Now that data has been cleaned up, we now need to establish what our target column is and set it to y, with our predictors being set to X.

As our event_type column has already been converted to binary we can set it to y.

```
In [34]: X = ecom_df.drop(["event_type"], axis=1)
y = ecom_df.event_type
```

We will also check out y classes to see if there is a class imbalance.

```
In [35]: y.value_counts()
```

```
Out[35]: 1    38449
0     17214
Name: event_type, dtype: int64
```

We can see there is a class imbalance. We will have to account for this before we model.

Now that our data has been separated we need to split the data into a train and test set.

Feature Engineering

As our DataFrame doesn't have many features, we will have to build out our own features to help our models perform better. As we have a timestamp for every event, we will start with Feature Engineering using time.

Time Based Features

First we convert our event_time into a datetime object to allow us to have more functionality with the column.

```
In [36]: X.event_time = pd.to_datetime(X['event_time'])
```

The first feature we will add, is if an event happened on a Holiday. It seems reasonable that people may be more inclined to spend money when they have the day off work.

```
In [37]: dr = pd.date_range(start='2019-10-01', end='2020-02-29')
df = pd.DataFrame()
df['Date'] = dr

cal = calendar()
holidays = cal.holidays(start=dr.min(), end=dr.max())
X['holiday'] = X['event_time'].isin(holidays)
```

We now will pull from the `event_time` column three new columns:

1. What month was the event during?
2. Was the event in 2020 or 2019?
3. What hour of the day is the event taken place during?

```
In [38]: X['month'] = pd.DatetimeIndex(X['event_time']).month
X['2020'] = pd.DatetimeIndex(X['event_time']).year
X['hour'] = pd.DatetimeIndex(X['event_time']).hour

#converting the year to binary
X['2020'].replace({2019: 0, 2020: 1}, inplace=True)

#converting month to string
X['month'].replace({1: 'January', 2: 'February', 10: 'October', 11: 'November', 1
```

Now to add a feature, on what day of the week will a purchase take place.

```
In [39]: X['day'] = X['event_time'].dt.dayofweek
X['day'] = X['day'].replace({0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thur
```

Creating four bins, we will also establish a feature that describes at what period of the day did an event happen:

1. Night: Midnight -> 6am
2. Morning: 6am -> Midday
3. Afternoon: Midday -> 6pm
4. Evening: 6pm ->Midnight

```
In [40]: #creating new when dataframe
when_df = pd.DataFrame({'hour':range(1, 25)})
bins = [0,6,12,18,24]
labels = ['Night', 'Morning', 'Afternoon', 'Evening']
when_df['when'] = pd.cut(when_df['hour'], bins=bins, labels=labels, include_lowe

#joining the new dataframe
X = X.join(when_df['when'], on=X["hour"])
```

Train Test Split

```
In [41]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Labeling

Now we will use LabelBinarizer and OneHotEncoder on several feature in the natives dataset, and several of the features we built above to create many new features

```

In [42]: #Labeling
encoder = LabelBinarizer()
brand_labels = pd.DataFrame(encoder.fit_transform(X_train[['brand']]), columns=encoder.get_feature_names_out())
encoder2 = LabelBinarizer()
productid_labels = pd.DataFrame(encoder2.fit_transform(X_train[["product_id"]]), columns=encoder2.get_feature_names_out())

#OHE
ohe1 = OneHotEncoder(sparse=False, drop='first')
catco_labels = pd.DataFrame(ohe1.fit_transform(X_train[['category_code']]), index=X_train.index, columns=ohe1.get_feature_names_out())
ohe2 = OneHotEncoder(sparse=False, drop='first')
day_labels = pd.DataFrame(ohe2.fit_transform(X_train[['day']]), index=X_train.index, columns=ohe2.get_feature_names_out())
ohe3 = OneHotEncoder(sparse=False, drop='first')
month_labels = pd.DataFrame(ohe3.fit_transform(X_train[['month']]), index=X_train.index, columns=ohe3.get_feature_names_out())
ohe4 = OneHotEncoder(sparse=False, drop='first')
when_labels = pd.DataFrame(ohe4.fit_transform(X_train[['when']]), index=X_train.index, columns=ohe4.get_feature_names_out())

#join the dataframes together
X_train = X_train.join(brand_labels)
X_train = X_train.join(productid_labels)
X_train = X_train.join(catco_labels)
X_train = X_train.join(day_labels)
X_train = X_train.join(month_labels)
X_train = X_train.join(when_labels)
X_train.shape

```

Out[42]: (44530, 551)

Lets make sure everything has executed correctly and our X_train and y_train are the same length.

```

In [43]: print(X_train.shape)
         print(y_train.shape)

```

(44530, 551)

(44530,)

Everything looks good, both have 44530 rows.

Preprocessing Data

Now that data has been cleaned up and the new features we wanted to add have been built, we now need to use SMOTE to preprocess our data to correct the class imbalance in our y.

We will also need to drop our features in X_train that we have encoded or are redundant.

```

In [44]: X_train = X_train.drop(["month", "index", "event_time", "hour", "day", "when", "u

```

SMOTE

As we have a class imbalance of roughly 2:1 in favour of our removed_from_cart category we will smote our training data to help our models perform more accurately.

```

In [45]: sm = SMOTE(random_state=42)
         X_tr_sm, y_tr_sm = sm.fit_resample(X_train, y_train)

```

Again lets check to make sure our features and our target are the same length before we start to model.

```
In [46]: print(X_tr_sm.shape)
         print(y_tr_sm.shape)
```

```
(61562, 540)
(61562,)
```

Everything looks good, now moving on to modeling.

Models

For our models, we will train each model on our transformed training data before we cross validate it to ensure accuracy.

LGBMClassifier

LGBM is one of the strongest, if not the strongest, model on the market currently. I have had success with it before and wanted to start with a strong model to see what kind of results it can give as a benchmark.

```
In [47]: import lightgbm as lgb
```

```
In [48]: # Instantiate XGBClassifier
         lgbm = lgb.LGBMClassifier()

         # Fit XGBClassifier
         lgbm.fit(X_tr_sm, y_tr_sm)

         # Predict on training and test sets
         training_preds_lgbm = lgbm.predict(X_tr_sm)
```

```
In [49]: cross_val_score (lgbm, X_train, y_train, cv=5, scoring='f1')
```

```
Out[49]: array([0.81471023, 0.81610738, 0.81491452, 0.81404123, 0.81486459])
```

While LGBM gave great results with default hyperparameters, I wanted to try some different models first.

DecisionTreeClassifier

Our next model was a Decision Tree. We wanted something to compare to XGBoost that was a little bit more simple but still could deal with complex models. We also wanted a model that could execute faster than XGBoost.

```
In [50]: # Instantitate DecisionTreeClassifier
         tree_clf = DecisionTreeClassifier()

         # Fit DecisionTreeClassifier
         tree_clf.fit(X_tr_sm, y_tr_sm)

         # Predict on training and test sets
         training_preds_tclf = tree_clf.predict(X_tr_sm)
```

```
In [51]: cross_val_score (tree_clf, X_train, y_train, cv=5, scoring='f1')
```

```
Out[51]: array([0.73945409, 0.73815223, 0.73207364, 0.73483508, 0.74064043])
```


RandomForestClassifier

While RandomForest usually performs worse than LGBM, I wanted to do my due diligence and test the models performance first.

```
In [52]: # Instantiate RandomForestClassifier
rf = RandomForestClassifier()

# Fit RandomForestClassifier
rf.fit(X_tr_sm, y_tr_sm)

# Predict on training and test sets
training_preds_rf = rf.predict(X_tr_sm)
```

```
In [53]: cross_val_score (rf, X_train, y_train, cv=5, scoring='f1')
```

```
Out[53]: array([0.77293888, 0.77092709, 0.77232075, 0.76868273, 0.77613951])
```

Hyperparamter Tuning

We have established that our RandomForestClassifier appears to be the best performing model at baseline. The next step is to adjust certain parameters to help improve the quality of our model.

Warning: this grid will take a while to run.

```
In [ ]: param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 95, 110, 115, 120],
    'max_features': [2, 3, 7, 10, 30, 50, 100, 150],
    'criterion': ['gini', 'entropy'],
    'n_estimators': [100, 200, 300, 400]}
```

```
In [ ]: # Instantiate the grid search model
clf = GridSearchCV(estimator = rf, param_grid = param_grid,
                  cv = 3, n_jobs = -1, verbose = 2)
clf.fit(X_tr_sm, y_tr_sm)
GridSearchCV(estimator=rf,
             param_grid=param_grid)
```

```
In [ ]: def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:.2f}%.'.format(accuracy))

    return accuracy
```

```
In [ ]: # Fit the grid search to the data
print(clf.best_params_)
best_grid = clf.best_estimator_
```

```
grid_accuracy = evaluate(best_grid, X_test, y_test)
print(grid_accuracy)
```

Best Model

After running the param_grid in our HyperParameter step, we established that the best model is as follows: RandomForestClassifier with:

1. criterion: gini
2. max_depth: 115
3. max_features: 100
4. n_estimators: 400
5. bootstrap: True

```
In [56]: Best_model = RandomForestClassifier(bootstrap=True, criterion='gini', max_dept
Best_model.fit(X_tr_sm, y_tr_sm)

# Predict on training and test sets
training_preds_bm = Best_model.predict(X_tr_sm)

# Printing a classification report for our best model
print(classification_report(y_tr_sm, training_preds_bm))
```

	precision	recall	f1-score	support
0	0.86	0.83	0.85	30781
1	0.84	0.87	0.85	30781
accuracy			0.85	61562
macro avg	0.85	0.85	0.85	61562
weighted avg	0.85	0.85	0.85	61562

```
In [57]: cross_val_score (Best_model, X_train, y_train, cv=5, scoring='f1')
```

```
Out[57]: array([0.77481804, 0.77332334, 0.77027948, 0.7700309 , 0.77848338])
```

Best Model on Test

Now that we have established what our best model is, and what are the best hyperparameters for the model we will need to transform our test data before we run the test data through our trained model.

Transforming the test set

```
In [58]: #Labeling
brand_labels = pd.DataFrame(encoder.transform(X_test[['brand']]), columns=encode
productid_labels = pd.DataFrame(encoder2.transform(X_test[["product_id"]]), colu

#OHE
catco_labels = pd.DataFrame(ohe1.transform(X_test[['category_code']]), index=X_t
day_labels = pd.DataFrame(ohe2.transform(X_test[['day']]), index=X_test.index, c
month_labels = pd.DataFrame(ohe3.transform(X_test[['month']]), index=X_test.inde
```

```

when_labels = pd.DataFrame(ohe4.transform(X_test[['when']]), index=X_test.index,

#join the dataframes together
X_test = X_test.join(brand_labels)
X_test = X_test.join(productid_labels)
X_test = X_test.join(catco_labels)
X_test = X_test.join(day_labels)
X_test = X_test.join(month_labels)
X_test = X_test.join(when_labels)

```

Now we have to drop the columns we transformed.

```
In [59]: X_test = X_test.drop(["month", "index", "event_time", "hour", "day", "when", "us
```

Testing our best model

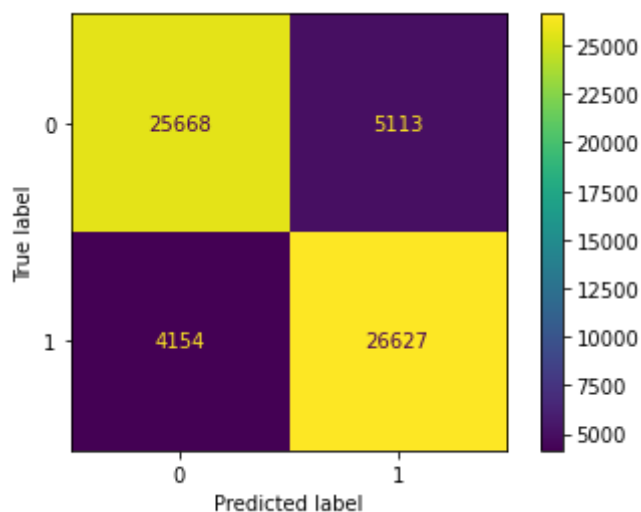
```
In [61]: # Predicting our test scores on our Best_model
test_preds_bm = Best_model.predict(X_test)

# Printing a classification report for our best model
print(classification_report(y_test, test_preds_bm))
```

	precision	recall	f1-score	support
0	0.42	0.38	0.40	3465
1	0.73	0.76	0.74	7668
accuracy			0.64	11133
macro avg	0.57	0.57	0.57	11133
weighted avg	0.63	0.64	0.64	11133

Visuals

```
In [64]: #confusion matrix
plot_confusion_matrix(estimator=Best_model, y_true=y_tr_sm, X=X_tr_sm)
plt.savefig("../figs/conufsion_matrix_train")
```



```
In [65]: #confusion matrix
plot_confusion_matrix(estimator=Best_model, y_true=y_test, X=X_test)
plt.savefig("../figs/conufsion_matrix_test")
```

