Pharmaceutical Sales Prediction

Analysis of Pharmaceutical dataset

ABSTRACT

01

Creating a powerful model by organizing the data and testing different models.

02

Predicting whether a pharmacy customer will make a purchase in the month of October using data we have from the preceding months

BUSINESS PROBLEM STATEMENT

IMPROVING SALES FOR THE MONTH OF OCTOBER

APPROACH

- Organization of the data
- Selecting a model
- Importing the libraries
- Importing the dataset
- Exploratory data analysis
- Model selection
- Decision and Conclusion

Approach

- We will try to predict whether a pharmacy customer makes a purchase in the month of October using data that we have from previous purchase of past months
- We will make use of the data from April 2019 to September 2019 as the training set and the October data as my test set. Also, we assign our target variable called "flag" a value of 1 if an October purchase is made and 0 if an October purchase is not made. Once the model has been built, we will use the hold out data for final validation
- **Organization of data**: Each observation represents a single purchase at the pharmacy, so an extra time is spent organizing the data so that each observation is rolled up to the customer level
- **Selecting a model:** I tried at least 9 different model on the dataset to visualize the predictive power as well as the level of accuracy

ORGANIZING THE DATA

- We first had to import the libraries to use in our project

```
#Import Libraries
import numpy as np # linear algebra
import pandas as pd # data processing

from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.style as style
from matplotlib import pyplot
from numpy import where
import seaborn as sns
import warnings
```

Importing the training and test set and conducting preliminary EDA

Import Dataset

```
# Training data
data = pd.read_csv('purchase_data_bill_level_sept.csv')
data["created_at_bill"] = pd.to_datetime(data["created_at_bill"]).dt.date
data.head()
```

	bill_ref_id	store_ref_id	customer_ref_id	doctor_ref_id	payment_method	created_at_bill	num_drugs_bill	total_quantity_bill	mrp_bill	total_spend_bill	...
0	7562292	468	6251676	6628790	cash	2019-04-01	3	12	394.60	146.92	...
1	7562335	468	6702206	7298144	cash	2019-04-01	1	1	50.50	26.04	...
2	7562354	468	6689660	6478435	cash	2019-04-01	3	7	708.50	330.19	...
3	7562366	468	8539983	6755214	cash	2019-04-01	1	1	52.32	26.00	...
4	7562370	468	8539984	7298144	cash	2019-04-01	1	1	18.00	14.40	...

5 rows × 21 columns

```
# Test target data
test_target = pd.read_csv('out_of_time_test_oct.csv')
test_target = test_target.rename(columns={"oct_purchase_flag": "flag"})
test_target.head()
```

Eyeballing and trying to fix the missingness in our dataset which will invariably lead us to perform imputation on the columns with missing data

	types	nulls	% nulls	size	uniques
bill_ref_id	int64	0	0.000000	1184025	1184025
store_ref_id	int64	0	0.000000	1184025	43
customer_ref_id	int64	0	0.000000	1184025	278136
doctor_ref_id	int64	0	0.000000	1184025	72099
payment_method	object	3	0.000003	1184025	6
created_at_bill	object	0	0.000000	1184025	183
num_drugs_bill	int64	0	0.000000	1184025	49
total_quantity_bill	int64	0	0.000000	1184025	257
mrp_bill	float64	0	0.000000	1184025	200462
total_spend_bill	float64	0	0.000000	1184025	229685
return_value_bill	float64	0	0.000000	1184025	18147
returned_quantity_bill	int64	0	0.000000	1184025	88
quantity_ethical	int64	0	0.000000	1184025	159
quantity_generic	int64	0	0.000000	1184025	194
quantity_surgical	int64	0	0.000000	1184025	103
quantity_ayurvedic	int64	0	0.000000	1184025	30
quantity_general	int64	0	0.000000	1184025	80
quantity_otc	int64	0	0.000000	1184025	30
quantity_chronic	int64	0	0.000000	1184025	155
quantity_acute	int64	0	0.000000	1184025	224
quantity_h1	int64	0	0.000000	1184025	50

Missing value instruction

- #based on the above we can impute the missing values as following:
- # 1. for customer 7898076, most payments are done with card so we'll impute that value
- # 2. for customer 6754310, the more expensive purchases were with phonepe and since this payment is above average, we will impute phonepe
- # 3. for customer 6076893, we will impute cash since most purchases under 100 are paid with cash

EXPLORATORY DATA ANALYSIS

- Imputing payment method
- Finding and replacing missing values
- Creating a target variable for the test set
- Performing one hot encoder

```
for i in [7898076, 6754310, 6076893]:
    print(data[data["customer_ref_id"]==i ][["customer_ref_id","payment_

         customer_ref_id payment_method  total_spend_bill  total_quantity_bill
50493            7898076            NaN           925.225                    5
52855            7898076           cash          1133.475                    3
53795            7898076           card           124.100                    1
54096            7898076           card           363.800                    2
55262            7898076           card           584.800                    2
119497           7898076           card           224.350                    2
138202           7898076           card           326.700                    2
598525           7898076           card           501.250                    5
721139           7898076           card           748.000                    2
          customer_ref_id payment_method  total_spend_bill  total_quantity_bill
53120             6754310           cash             55.02                    4
53268             6754310        phonepe            209.70                    1
73757             6754310        phonepe            839.65                    5
82850             6754310            NaN            508.30                    1
261373            6754310          paytm            173.70                    1
734541            6754310        phonepe            421.20                    2
885788            6754310           cash             67.95                    1
1129372           6754310        phonepe             89.64                    1
          customer_ref_id payment_method  total_spend_bill  total_quantity_bill
6790              6076893           cash            80.000                    7
11883             6076893           cash            93.480                    3
14449             6076893           cash            35.360                    4
62172             6076893           cash            76.700                    1
```

h for anything

Run Code

Organize the data for the training and test set

```
#First, for the training set, we will take out all instances of september
data["flag"] = [1 if x.month == 9 else 0 for x in data["created_at_bill"]]
print(data.flag.value_counts())

data = data[data["flag"] == 0]
print(data.flag.value_counts())

data = data.drop('flag', axis=1)
```

```
0    963561
1    220464
Name: flag, dtype: int64
0    963561
Name: flag, dtype: int64
```

Grouped data mean of each numeric value

```
#training data
summary_reg = data.groupby('customer_ref_id')[[col for col in data if col not in ["doctor_ref_id", "customer_ref_id", "bill_ref
summary_reg.columns = ['_'.join(col).strip() for col in summary_reg.columns.values]

#test data
```

One hot encoding

- Performing one hot encoding on the payment method variable to convert it from categorical variable to binary variable
- After which we split the already prepped data into train and test set then we can start modelling

One hot encoding to fix the payment_method varible¶

```
#training set
merged2 = pd.get_dummies(merged)

#test set
test = pd.get_dummies(merged_test)
```

Append target variable to the datasets

```
#training set
train = merged2.merge(target, how="left", on="customer_ref_id")

#test set
test = test.merge(test_target, how="left", on="customer_ref_id")
```

Correlation

```
corr = train[train.columns[1:]].corr()['flag'][:].sort_values(ascending=True)
# print(corr)

corr[:-1].plot(kind='barh', figsize = (20,10), fontsize = 17)
```

MODELLING

- Importing the libraries for modelling
- Importing the libraries for performance metrics
- Importing the libraries to learn classifiers

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate

# Import required libraries for performance metrics
from sklearn import metrics
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
```

- *These group of libraires are divided into three.*
- *1. Libraries for modelling*
- *2. Libraries to test performance metrics of our model*
- *3. Libraries to test our machine learning classifiers*

Defining a function to hold the accuracies and F1 Scores of our model

Below are a few functions to streamline modeling

```python
accuracies = {}
kappaScores= {}
f1scores={}

def Models(models, X_train, X_test, y_train, y_test, title):
    model = models
    model.fit(X_train,y_train)

    test_matrix = pd.crosstab(y_test, model.predict(X_test), rownames=['Actual'], colnames=['Predicted'])

    g2 = sns.heatmap(test_matrix, annot=True, fmt=".1f",cbar=False)
    g2.set_title(title)
    g2.set_ylabel('Total Observed October Sales = {}'.format(y_test.sum()), rotation=90)
    g2.set_xlabel('Accuracy score for Testing Dataset = {}'.format(accuracy_score(model.predict(X_test), y_test)))

    #record metrics
    accuracies[title]=accuracy_score(model.predict(X_test), y_test)*100
    f1scores[title]=f1_score(model.predict(X_test), y_test)*100
    kappaScores[title]=cohen_kappa_score(model.predict(X_test), y_test)*100

    plt.show()
```

Model output!

- We will define dictionaries with performance metrics
- Instantiate the machine learning classifiers
- Perform a cross fold validation on each machine learning model

```
# Define dictionary with performance metrics
scoring = {'accuracy':make_scorer(accuracy_score),
           'precision':make_scorer(precision_score),
           'recall':make_scorer(recall_score),
           'f1_score':make_scorer(f1_score)}
```

Instantiate the machine learning classifiers

- Perform a cross fold validation on each machine learning model

```python
# Instantiate the machine learning classifiers
log_model = LogisticRegression(max_iter=10000)
svc_model = LinearSVC(dual=False)
dtr_model = DecisionTreeClassifier()
rfc_model = RandomForestClassifier()
gnb_model = GaussianNB()
xgb_model = XGBClassifier()
lgb_model = LGBMClassifier()
knn_model = KNeighborsClassifier(n_neighbors=3)
gbc_model = GradientBoostingClassifier(n_estimators=500, learning_rate=1,
lda_model = LinearDiscriminantAnalysis()
```

```python
def models_evaluation(X, y, folds):

    # Perform cross-validation to each machine learning classifier
    log = cross_validate(log_model, X, y, cv=folds, scoring=scoring)
    svc = cross_validate(svc_model, X, y, cv=folds, scoring=scoring)
    dtr = cross_validate(dtr_model, X, y, cv=folds, scoring=scoring)
    rfc = cross_validate(rfc_model, X, y, cv=folds, scoring=scoring)
    gnb = cross_validate(gnb_model, X, y, cv=folds, scoring=scoring)

    xgb = cross_validate(xgb_model, X, y, cv=folds, scoring=scoring)
    lgb = cross_validate(lgb_model, X, y, cv=folds, scoring=scoring)
    knn = cross_validate(knn_model, X, y, cv=folds, scoring=scoring)
    gbc = cross_validate(gbc_model, X, y, cv=folds, scoring=scoring)
    lda = cross_validate(lda_model, X, y, cv=folds, scoring=scoring)
```

DATAFRAME TO CAPTURE OUR MODELS

```
models_scores_table = pd.DataFrame({'Logistic Regression':[log['test_accuracy'].mean(),
                                                           log['test_precision'].mean(),
                                                           log['test_recall'].mean(),
                                                           log['test_f1_score'].mean(),
                                                           log['fit_time'].mean(),
                                                           log['score_time'].mean()],

                                    'Support Vector Classifier':[svc['test_accuracy'].mean(),
                                                                 svc['test_precision'].mean(),
                                                                 svc['test_recall'].mean(),
                                                                 svc['test_f1_score'].mean(),
                                                                 svc['fit_time'].mean(),
                                                                 svc['score_time'].mean()],

                                    'Decision Tree':[dtr['test_accuracy'].mean(),
                                                     dtr['test_precision'].mean(),
                                                     dtr['test_recall'].mean(),
                                                     dtr['test_f1_score'].mean(),
                                                     dtr['fit_time'].mean(),
                                                     dtr['score_time'].mean()],

                                    'Random Forest':[rfc['test_accuracy'].mean(),
                                                     rfc['test_precision'].mean(),
                                                     rfc['test_recall'].mean()
```

MODEL OUTPUT

- We compared multiple models using a 5-fold cross validation. It is important to note that I did not use the test set (or holdout) with the actual October values. These will be used when evaluating the chosen model

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

	Logistic Regression	Support Vector Classifier	Decision Tree	Random Forest	XGB	LGB	KNN	GBC	LDA	Gaussian Naive Bayes	Best Score
Accuracy	0.793655	0.789644	0.681070	0.775673	0.793824	0.798925	0.726057	0.794317	0.724069	0.716356	LGB
Precision	0.770590	0.785166	0.468436	0.654649	0.729953	0.752639	0.550791	0.737563	0.674424	0.546836	Support Vector Classifier
Recall	0.486122	0.449078	0.535526	0.550322	0.548978	0.548758	0.519082	0.539405	0.426771	0.603300	Gaussian Naive Bayes
F1 Score	0.563105	0.536237	0.497816	0.579564	0.594647	0.598629	0.532035	0.590401	0.471595	0.535721	LGB
fit_time	24.186776	2.373043	3.092731	43.887778	16.154774	1.457699	0.056843	31.717781	0.695374	0.159778	Random Forest
score_time	0.060240	0.053259	0.073416	1.796200	0.159972	0.253522	143.137007	0.314563	0.052260	0.083571	KNN

RESULTS

Comparing multiple models using a 5-fold cross validation

DECISION

- Choosing the model that contains the largest accuracy, which is based on the above table is LGBM Classifier.
- The accuracy reflects the amount of correct true and false predictions made and therefore is a good representation of the overall performance of the model.

CONCLUSION

Comparing each model by using a variety of metrics, discussed how to go about choosing the model based on the performance metrics of each model, and considered the business problem at hand.