ML Project
Group Members

Saravana Prabhu Ramasamy
Koushik Chandrasekaran
Nikhil Krishna Balamurali

We consider the problem of customer segmentation using a hybrid clustering and classification system for segmenting customers into different groups.

Context

An automobile manufacturer intends to use its current products (P1, P2, etc.) to explore newer markets. The conclusion after conducting extensive market research is that the new market's pattern or behavior is comparable to that of their current market.

The sales team has classified their customers into 4 different divisions (A, B, C, D). After carrying out segmented outreach and communication for several of their clients, they found out that they had great success with this strategy. The plan is to use this strategy on new markets with 2627 new potential clients.

Link to the dataset: https://www.kaggle.com/datasets/vetrirah/customer

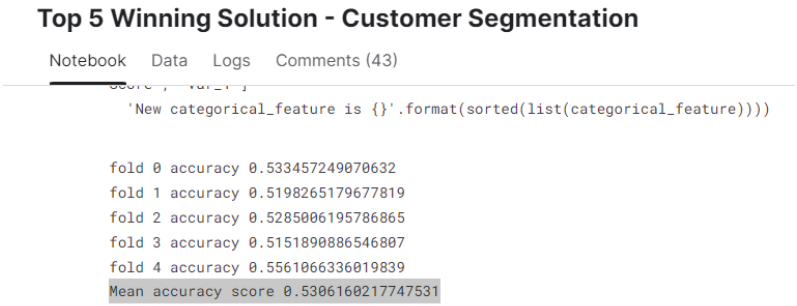
Introduction and Problem Statement

Customer segmentation is the practice of dividing a customer base into groups of individuals that are similar in specific ways relevant to marketing, such as age, gender, interests and spending habits. Companies employing customer segmentation operate under the fact that every customer is different and that their marketing efforts would be better served if they target specific, smaller groups with messages that those consumers would find relevant and lead them to buy something

Existing Approach (From Milestone 2)

Looking at some of the works on this dataset, there were many solutions that used traditional classification models (Decision Trees, Random Forests, K-NN, Naïve Bayes), and clustering models (Hierarchical clustering, and K-Means clustering to this problem on customer segmentation). Both clustering and classification models can be applied on this problem statement. One particular notebook that really stood out was Vetrivel's and his approach used the Light GBM

classification model. His work is intriguing and presented the best results with a score of 0.53 which was amongst the top 5 solutions to this data-science hackathon by Analytics Vidhya. Further insights show that the best approach to this problem is to use boosting methods.



Even though his work had a systematic approach, there is some room for improvement. Pre-processing methods such as feature selection could be a good addition to retain the most important features in the data. It increases the prediction power of the algorithms by selecting the most critical variables and eliminating the redundant and irrelevant ones. The following snippet performs feature selection.

How can our approach be different?

- We can explore encoding methods such as Label Encoding.
- To further improve the accuracy of the ML model, hyperparameter tuning can be employed. This concept was introduced in the notebook, however, the implementation was missing. Hyperparameter tuning is an essential part of controlling the behavior of a machine learning model. If we don't correctly tune our hyperparameters, our estimated model parameters produce suboptimal results, as they don't minimize the loss function, which makes more errors.
- Another Innovative approach to this problem statement would be to combine both clustering and classification algorithms to form an hybrid-ensemble machine learning system. In short, we could first separate the datasets into different clusters, and fit a classifier on each of those clusters.

Proposed System

The proposed hybrid ML model combines supervised (classification) and unsupervised learning (clustering) to further improve and explore the prediction modeling system. This approach divides the original dataset into clusters using K-Prototype clustering and then builds in base prediction, on each of the clusters. The final accuracy is then calculated by taking the mean of accuracy values in all clusters.

Pre-learning by clustering keeps similar cases in the same group. This improves the on-hand classifier's performance. Clustering before classification provides an added description to the data and improves the effectiveness of the classification task. This model can be deployed with any classification algorithms to improve its performance.

Our entire work has been divided into different modules as follows

1. Data Preprocessing
2. Sampling Data Distribution to Increase Size
3. Feature Scaling - Normalization
4. Unsupervised Learning - Clustering
5. Addressing Class Imbalance Problem
6. Train-Test Split
7. Multi-Class Classification on each cluster
8. Decision-Tree Classifier
9. Hyperparameter Tuning
10. Final Results

Data Preprocessing

The dataset contains almost 10 variables. It also has several missing values or NA values. In order to improve the performance of the model, we have to convert these missing values into their counterparts. The dataset contains the following variables.

- ID
- Gender
- Marital Status
- Age
- Educational status
- Profession
- Work Experience
- Spending score
- Family size
- Category

Here ID is insignificant to our model, so we are nullifying the complete column. Marital status has several missing values which we are replacing with No (not married). Similarly, Educational status has some missing values which we are replacing with No(not graduated).

```
# Filling nan values with "No"
df.Graduated.fillna("No", inplace = True)
```

In a similar way we cleanse the complete data replacing all missing values. For the last variable - Category, since we do not have information on how these categories are assigned we remove the rows with missing values.

```
# drop rows with na values in Var_1
df = df[df['Var_1'].notna()]
```

Sampling Data Distribution to Increase Size

In order to increase the performance of model, we increase the size of it by using the Random sampling method. Random sampling is a part of the sampling technique in which each sample has an equal probability of being chosen. A sample chosen randomly is meant to be an unbiased representation of the total population. Here we increase the size for the data by thirty percent. We randomly select 30 percent of the data and then append it at the bottom of the table.

```
sample = df.sample(frac=0.30)
df = df.append(sample)
```

```
df.size
```

```
103900
```

Feature Scaling - Normalization

Before fitting any machine learning model, we must ensure that the numerical values follow a common distribution as the model might struggle to interpret varying distributions of different numerical data. Therefore, we need to either normalize or standardize our distribution across all numerical features. For this purpose, we adopt the MinMaxScaler to normalize all numerical features in the range [0,1].

```
In [28]: from sklearn.preprocessing import MinMaxScaler

         # Extract numerical column positions
         numeric_cols = ['Age', 'Work_Experience', 'Family_Size', 'Var_1']
         scaler = MinMaxScaler().fit(df[numeric_cols])

In [29]: df[numeric_cols] = scaler.transform(df[numeric_cols])

In [30]: df.head()
```

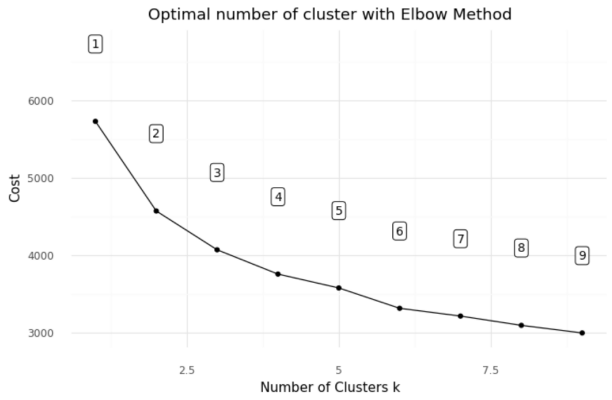
Out[30]:

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	Male	No	0.056338	No	Healthcare	0.071429	Low	0.375	0.500000	D
1	Female	Yes	0.281690	Yes	Engineer	0.000000	Average	0.250	0.500000	A
2	Female	Yes	0.690141	Yes	Engineer	0.071429	Low	0.000	0.833333	B
3	Male	Yes	0.690141	Yes	Lawyer	0.000000	High	0.125	0.833333	B
4	Female	Yes	0.309859	Yes	Entertainment	0.000000	High	0.625	0.833333	A

Unsupervised Learning - Clustering

The general clustering methods used for different datasets are K-Means, and K-Modes based on the type of data. However, there are a few shortcomings in both of these methods. While the K-means is used on numerical distributions where the cost function is based on the Euclidean distance between the numerical features, K-modes is used for categorical data where the cost function is calculated based on matching categorical features across the dataset.

Since our dataset is mixed, in the sense, contains both categorical and numerical features, we employ a method known as K-Prototypes. K-Prototype is a combination of the K-Means and K-Modes where the cost function is the sum of both the individual costs from K-Means and K-Modes where each of the individual methods deals with their respective data types. We also determine the number of clusters using the elbow method where the graph denotes the number of clusters based on the elbow point in the graph. We do not take the extremities to avoid overfitting and therefore, we choose the optimal value which is k = 3 from the graph below.



Out[41]: <ggplot: (163717001974)>

Based on the above plot, lets assume k = 3

In [42]:
```
# Fitting the clusters
kprototype = KPrototypes(n_jobs = -1, n_clusters = 3, init = 'Huang', random_state = 0)
kprototype.fit_predict(df_matrix, categorical = category_columns_position)
```

Out[42]: array([0, 1, 1, ..., 1, 0, 1], dtype=uint16)

Finally, we assign the cluster labels from the output to the rows in the dataset and create subsets of the data based on these labels, in the sense, we will have 3 subsets or clusters of the data.

Out[44]:

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation	Cluster
4101	Male	Yes	0.661972	Yes	Entertainment	0.071429	Low	0.000	0.833333	B	1
1545	Male	Yes	0.253521	Yes	Artist	0.285714	Average	0.125	0.833333	C	1
5355	Female	Yes	0.380282	No	Engineer	0.000000	Average	0.750	1.000000	A	1
4434	Male	Yes	0.338028	No	Entertainment	0.428571	Low	0.250	0.833333	B	0
3986	Male	Yes	0.690141	Yes	Artist	0.000000	Average	0.625	0.000000	B	1

Addressing Class Imbalance Problem

From these subsets, however, we can notice a large imbalance in distribution of the segment labels. This will cause the model to overly fit towards a particular label causing underfitting with respect to the entire dataset.

```
In [48]: df1['Segmentation'].value_counts()

Out[48]: D    2006
         A     329
         C     290
         B     277
         Name: Segmentation, dtype: int64

In [49]: df2['Segmentation'].value_counts()

Out[49]: C    1947
         B    1409
         A     857
         D     371
         Name: Segmentation, dtype: int64

In [50]: df3['Segmentation'].value_counts()

Out[50]: A    1353
         B     690
         D     543
         C     318
         Name: Segmentation, dtype: int64
```

To overcome this problem, we employ oversampling which will increase the distribution of lesser class data points to match that of the largest. After oversampling, we can expect all the labels to have an equal distribution. This is necessary to reduce underfitting so the model can understand the distribution of all class labels in a better manner.

```
In [56]: # X is all columns except the segmentation, y is segmentation
         X = df1.drop(['Segmentation'], axis = 1)
         y = df1['Segmentation']

         oversample = SMOTE()
         X, y = oversample.fit_resample(X, y)

In [57]: y.value_counts()

Out[57]: 3    2006
         2    2006
         1    2006
         0    2006
         Name: Segmentation, dtype: int64
```

Train-Test split

Before the next step, we split the entire dataset into training and testing sets using the train_test_split from sklearn package.

Multi-Class Classification on each cluster

We have 3 subsets of the data from clustering. We perform classification individually on each subset of data and compute the mean accuracy. For classification we will use the Decision Tree Algorithm.

Decision Tree Classifier

First we analyze the data in each cluster before running the classification algorithm.

Data in cluster 1 is shown below:

	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Score	Family_Size	Var_1	Segmentation
0	Male	No	0.056338	No	Healthcare	0.071429	Low	0.375	0.500000	D
6	Male	No	0.197183	Yes	Healthcare	0.071429	Low	0.250	0.833333	C
7	Female	No	0.211268	Yes	Healthcare	0.071429	Low	0.250	0.833333	D
11	Male	No	0.014085	No	Healthcare	0.285714	Low	0.375	0.500000	D
12	Female	No	0.014085	No	Executive	0.000000	Low	0.000	0.333333	D

Here there are 6 categorical data which has label values. This is converted to numerical data by using **Label Encoding**. It is an important preprocessing step for the structured dataset in supervised learning.

Next, we address the class imbalance problem in each cluster. In first cluster the data distribution of each class is shown below:

```
df1['Segmentation'].value_counts()
```

```
3    2006
0     329
2     290
1     277
Name: Segmentation, dtype: int64
```

The bias in the dataset can influence the algorithm, leading to ignoring the minority class entirely. This is a problem as it is typically the minority class on which predictions are most important. We will perform the **Random Oversampling** in each cluster to make equal distribution of class data which would avoid underfitting in the model.

```
y.value_counts()
```

```
3    2006
2    2006
1    2006
0    2006
Name: Segmentation, dtype: int64
```

Finally, we split the data in each cluster into training and testing dataset respectively.

Hyperparameter Tuning

Decision tree algorithms are generally prone to overfitting problems. Inorder to avoid this problem we have to fine tune hyperparameters that would result in better accuracy.

```
from sklearn.model_selection import GridSearchCV

def dtree_grid_search(X,y,nfolds):
    #create a dictionary of all values we want to test
    param_grid = { 'criterion':['gini','entropy'],'max_depth': np.arange(3, 20)}
    # decision tree model
    dtree_model=DecisionTreeClassifier()
    #use gridsearch to test all values
    dtree_gscv = GridSearchCV(dtree_model, param_grid, cv=nfolds)
    #fit model to data
    dtree_gscv.fit(X, y)
    return dtree_gscv.best_params_
```

The parameters to be used in the decision tree are gini and entropy with depth of the tree in the range of 3 to 19. The GridSearchCV would provide the best parameters for the model with the given dataset as shown below:

```
best = dtree_grid_search(X_train, y_train, 10)
print(best)
```

```
{'criterion': 'gini', 'max_depth': 19}
```

For each cluster we will find the best parameter using this approach.

Training Each Cluster

We can assign the best parameters obtained into the training model and perform the training with DecisionTreeClassifier from scikit-learn. Then for each cluster we calculate the accuracies. For each cluster we get the following accuracies.

1. Cluster 1 - Accuracy 77.11%
2. Cluster 2 - Accuracy 67.60%
3. Cluster 3 - Accuracy 69.09%

We take the mean accuracy from all the 3 clusters to compute the final accuracy which would result in 71.27% accuracy.

Conclusion

We get around 72% accuracy to classify the customers in the right segment which is approximately 20% better than the one we have studied. Hence, our implementation of using a hybrid model with a combination of unsupervised(Clustering) and supervised(Classification) learning yields the best result for the customer segmentation problem.