Documentation of Cross sell/ Up sell Model

Definition and Purpose of Moodle

Cross-selling and upselling are marketing techniques aimed at increasing sales by encouraging customers to buy additional or higher-priced products or services.
A cross-sell model is used to identify products or services that are often purchased together with the customer's current purchase. For example, a retailer might suggest a customer buy a case for their newly purchased smartphone.
An upsell model, on the other hand, is used to identify opportunities to upgrade the customer's current purchase to a more expensive option. For example, a hotel might offer a customer an upgrade to a higher-priced room.
These models can be implemented using various data-driven techniques, such as association rule mining, collaborative filtering, or machine learning algorithms. The goal is to identify the products or services that are most likely to result in a sale, based on data such as past purchase history, customer behavior, and demographic information.
Cross-sell and upsell models can help businesses increase revenue and improve customer satisfaction by offering relevant products or services that meet their needs.

Which ML Algorithms That Can Be Used To Create The Model

1. Association Rule Mining: This technique is based on finding associations between items in a large dataset. Association rule mining can be used to identify frequently purchased items and make recommendations based on this information.
2. Collaborative Filtering: This technique is based on the idea that people who have purchased similar items in the past are likely to purchase similar items in the future. Collaborative filtering algorithms can be used to make personalized recommendations based on a customer's purchase history.
3. Decision Trees: Decision trees are a type of supervised machine learning algorithm that can be used to make predictions based on a set of input

features. In the context of cross-sell or upsell, decision trees can be used to identify the factors that are most important in determining whether a customer is likely to make an additional purchase.

4. Random Forest: Random forests are an extension of decision trees that build multiple trees and combine their predictions to make a final prediction. Random forests can be used to handle complex data and non-linear relationships, making them well suited for cross-sell and upsell applications.
5. Neural Networks: Neural networks are a type of machine learning algorithm that are inspired by the structure and function of the human brain. They can be used to model complex relationships between inputs and outputs, making them well suited for cross-sell and upsell applications where there are many factors that influence a customer's purchasing behavior.

Definitions Of Some Terms

SUPPORT

This measure gives an idea of how frequent an *itemset* is in all the transactions. Consider *itemset1* = {bread} and *itemset2* = {shampoo}. There will be far more transactions containing bread than those containing shampoo. So as you rightly guessed, *itemset1* will generally have a higher support than *itemset2*. Now consider *itemset1* = {bread, butter} and *itemset2* = {bread, shampoo}. Many transactions will have both bread and butter on the cart but bread and shampoo? Not so much. So in this case, *itemset1* will generally have a higher support than *itemset2*. Mathematically, support is the fraction of the total number of transactions in which the itemset occurs.

Value of support helps us identify the rules worth considering for further analysis. For example, one might want to consider only the itemsets which occur at least 50 times out of a total of 10,000 transactions i.e. support = 0.005. If an *itemset* happens to have a very low support, we do not have enough information on the relationship between its items and hence no conclusions can be drawn from such a rule

CONFİDENCE

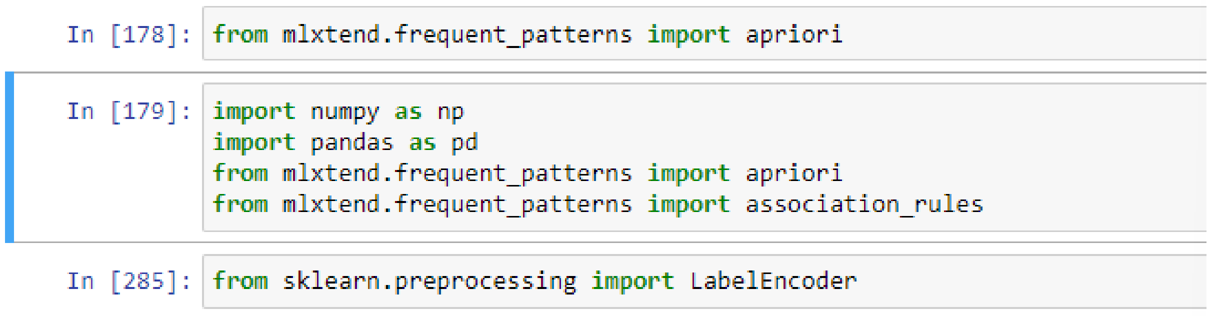
This measure defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents. That is to answer the question — of all the transactions containing say, {Captain Crunch}, how many also had {Milk} on them? We can say by common knowledge that {Captain Crunch} → {Milk} should be a high confidence rule. Technically, confidence is the conditional probability of occurrence of consequent given the antecedent.

LİFT

In the context of association rule mining, "lift" is a measure of the strength of the association between two items. It measures the ratio of the observed support for the items in the transaction data to the expected support if the items were independent.
The lift value for a given association rule "A -> B" can be calculated as:
lift(A -> B) = support(A U B) / (support(A) * support(B))
where "A" and "B" are the items in the rule, "support(A U B)" is the support of the combined itemset "A U B", and "support(A)" and "support(B)" are the supports of items "A" and "B", respectively.
A lift value of 1 indicates that the items in the rule are independent, while a lift value greater than 1 indicates that there is a positive association between the items, meaning that they are more likely to occur together than would be expected if they were independent. A lift value less than 1 indicates a negative association between the items, meaning that they are less likely to occur together than would be expected if they were independent.

How To Create The Model

First import neccesary liberaries:

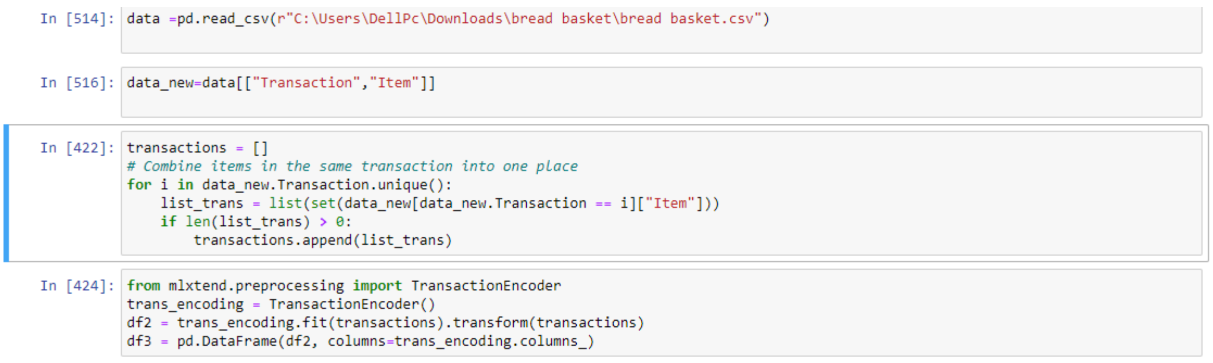


```
In [178]: from mlxtend.frequent_patterns import apriori
```

```
In [179]: import numpy as np
          import pandas as pd
          from mlxtend.frequent_patterns import apriori
          from mlxtend.frequent_patterns import association_rules
```

```
In [285]: from sklearn.preprocessing import LabelEncoder
```

Now let’s explore and prepare your data



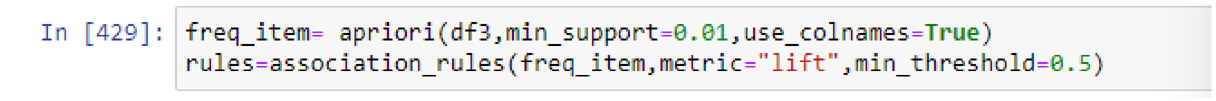
```
In [514]: data =pd.read_csv(r"C:\Users\DellPc\Downloads\bread basket\bread basket.csv")
```

```
In [516]: data_new=data[["Transaction","Item"]]
```

```
In [422]: transactions = []
          # Combine items in the same transaction into one place
          for i in data_new.Transaction.unique():
              list_trans = list(set(data_new[data_new.Transaction == i]["Item"]))
              if len(list_trans) > 0:
                  transactions.append(list_trans)
```

```
In [424]: from mlxtend.preprocessing import TransactionEncoder
          trans_encoding = TransactionEncoder()
          df2 = trans_encoding.fit(transactions).transform(transactions)
          df3 = pd.DataFrame(df2, columns=trans_encoding.columns_)
```

Now we apply the association algorathim to our data



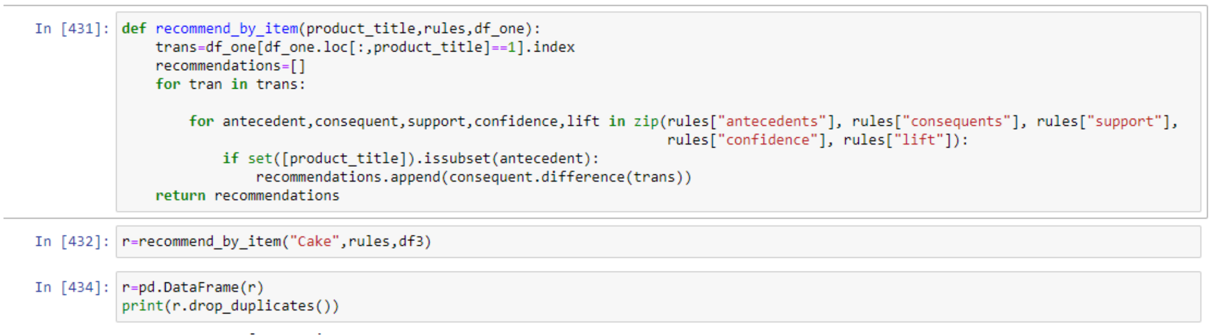
```
In [429]: freq_item= apriori(df3,min_support=0.01,use_colnames=True)
          rules=association_rules(freq_item,metric="lift",min_threshold=0.5)
```

The apriori function: Determine the frequency of items and it takes 3 argument :

First argument is the dataset, Second is min_support which represent the minumum required support value,and the third argument is use_colnames default is False but If we want it to use real column names, we have to make it True

Association_rules function: Is used to generate association rules from a given set of transactions. It uses the Apriori algorithm to identify frequent item sets and then applies a rule generation process to generate association rules. These association rules describe the relationships between items in a transaction set and can be used for market basket analysis, recommendation systems, and other applications.

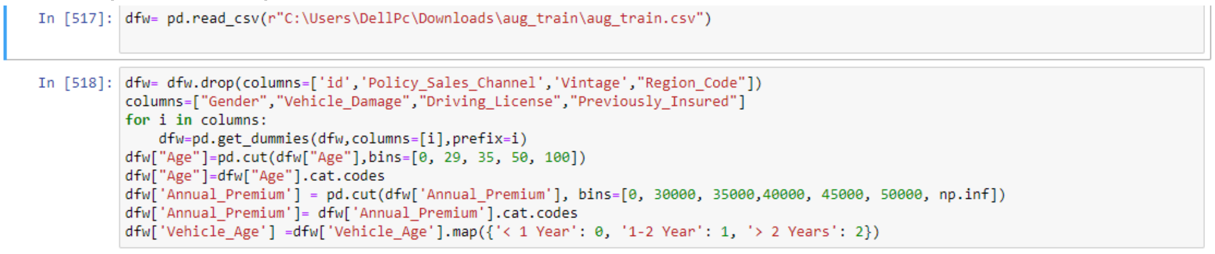
And now we create function to determine the items that appropriate to cross sell model



```
In [431]: def recommend_by_item(product_title,rules,df_one):
              trans=df_one[df_one.loc[:,product_title]==1].index
              recommendations=[]
              for tran in trans:

                  for antecedent,consequent,support,confidence,lift in zip(rules["antecedents"], rules["consequents"], rules["support"],
                                                                           rules["confidence"], rules["lift"]):
                      if set([product_title]).issubset(antecedent):
                          recommendations.append(consequent.difference(trans))
              return recommendations

In [432]: r=recommend_by_item("Cake",rules,df3)

In [434]: r=pd.DataFrame(r)
          print(r.drop_duplicates())
```

Apply the decision tree ,random forest and logistic regression to kaggle's data

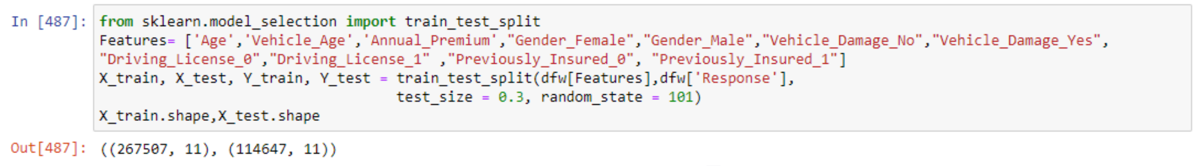
First explor and repair the data:



```
In [517]: dfw= pd.read_csv(r"C:\Users\DellPc\Downloads\aug_train\aug_train.csv")

In [518]: dfw= dfw.drop(columns=['id','Policy_Sales_Channel','Vintage',"Region_Code"])
          columns=["Gender","Vehicle_Damage","Driving_License","Previously_Insured"]
          for i in columns:
              dfw=pd.get_dummies(dfw,columns=[i],prefix=i)
          dfw["Age"]=pd.cut(dfw["Age"],bins=[0, 29, 35, 50, 100])
          dfw["Age"]=dfw["Age"].cat.codes
          dfw['Annual_Premium'] = pd.cut(dfw['Annual_Premium'], bins=[0, 30000, 35000,40000, 45000, 50000, np.inf])
          dfw['Annual_Premium']= dfw['Annual_Premium'].cat.codes
          dfw['Vehicle_Age'] =dfw['Vehicle_Age'].map({'< 1 Year': 0, '1-2 Year': 1, '> 2 Years': 2})
```

The **pd.cut** is a function in the pandas library used to bin continuous or numeric data into discrete intervals. The function takes in the data to be binned and the desired number of bins or the specific bin edges, and returns an object indicating which bin each data point belongs to. The function is used to discretize continuous variables, making them suitable for use in categorical data analysis.
The cat.code Function :The code is transforming the 'Age' column of the dataframe 'df' into categorical codes. 'cat.codes' is a pandas method that converts categorical data into numerical data, where each unique category is assigned a numerical code. In this code, the 'Age' column is transformed into numerical codes, which can then be used in further analysis. This method can be useful when working with machine learning algorithms that only accept numerical inputs.

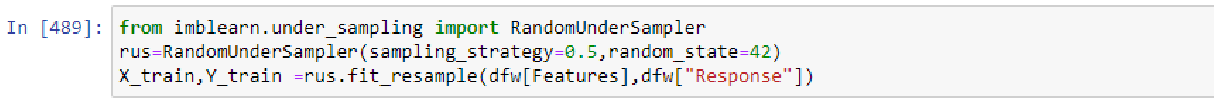
We split the data



```
In [487]: from sklearn.model_selection import train_test_split
          Features= ['Age','Vehicle_Age','Annual_Premium',"Gender_Female","Gender_Male","Vehicle_Damage_No","Vehicle_Damage_Yes",
          "Driving_License_0","Driving_License_1" ,"Previously_Insured_0", "Previously_Insured_1"]
          X_train, X_test, Y_train, Y_test = train_test_split(dfw[Features],dfw['Response'],
                                                              test_size = 0.3, random_state = 101)
          X_train.shape,X_test.shape

Out[487]: ((267507, 11), (114647, 11))
```

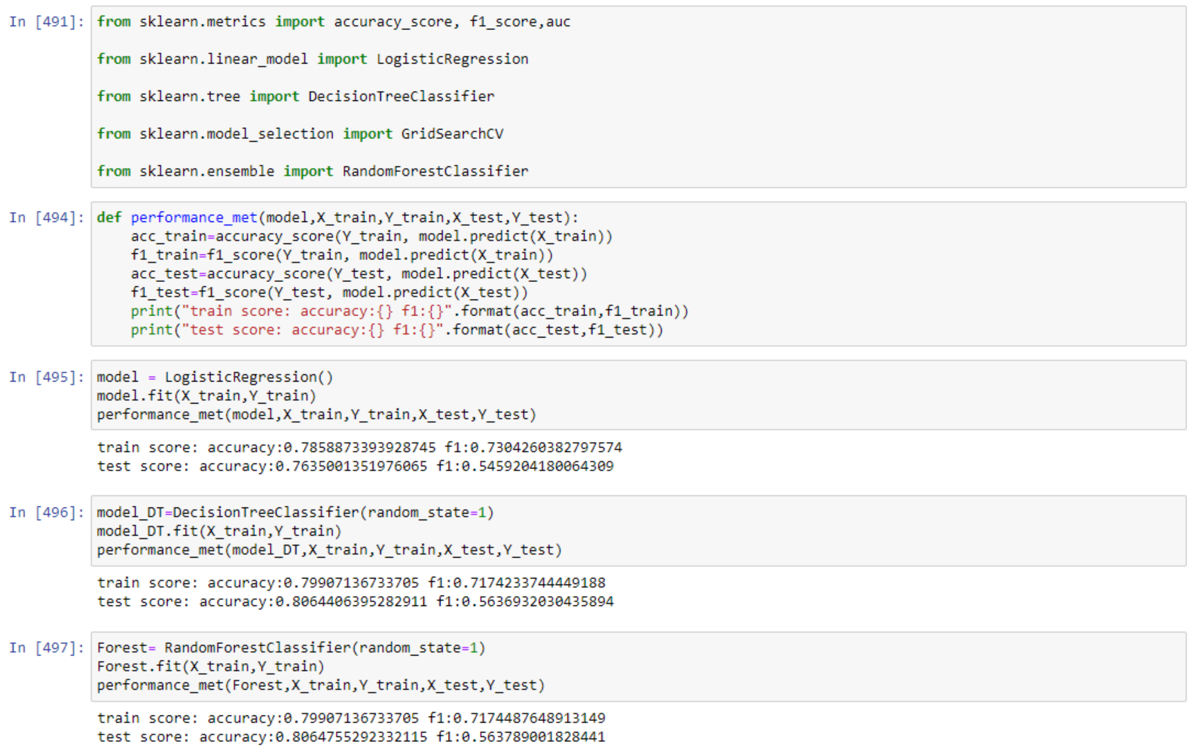
Because the data is imbalanced we have to balance a class distribution by randomly under-sampling the majority class.
So we used RandomUnderSampler

RandomUnderSampler : a method of data resampling in the field of machine learning. It is used to balance a class distribution by randomly under-sampling the majority class. This method can be useful in cases where the data is imbalanced, i.e., when the number of instances of one class far outweigh the number of instances of another class. By randomly under-sampling the majority class, RandomUnderSampler ensures that the balance of the class distribution is maintained in the resampled data. This can help prevent problems such as bias or over-fitting, which can occur when the majority class dominates the dataset. The RandomUnderSampler method is part of the imbalanced-learn library in Python.



```
In [489]: from imblearn.under_sampling import RandomUnderSampler
          rus=RandomUnderSampler(sampling_strategy=0.5,random_state=42)
          X_train,Y_train =rus.fit_resample(dfw[Features],dfw["Response"])
```

Now we apply the three algorithm



```
In [491]: from sklearn.metrics import accuracy_score, f1_score,auc

          from sklearn.linear_model import LogisticRegression

          from sklearn.tree import DecisionTreeClassifier

          from sklearn.model_selection import GridSearchCV

          from sklearn.ensemble import RandomForestClassifier

In [494]: def performance_met(model,X_train,Y_train,X_test,Y_test):
              acc_train=accuracy_score(Y_train, model.predict(X_train))
              f1_train=f1_score(Y_train, model.predict(X_train))
              acc_test=accuracy_score(Y_test, model.predict(X_test))
              f1_test=f1_score(Y_test, model.predict(X_test))
              print("train score: accuracy:{} f1:{}".format(acc_train,f1_train))
              print("test score: accuracy:{} f1:{}".format(acc_test,f1_test))

In [495]: model = LogisticRegression()
          model.fit(X_train,Y_train)
          performance_met(model,X_train,Y_train,X_test,Y_test)

          train score: accuracy:0.7858873393928745 f1:0.7304260382797574
          test score: accuracy:0.7635001351976065 f1:0.5459204180064309

In [496]: model_DT=DecisionTreeClassifier(random_state=1)
          model_DT.fit(X_train,Y_train)
          performance_met(model_DT,X_train,Y_train,X_test,Y_test)

          train score: accuracy:0.79907136733705 f1:0.7174233744449188
          test score: accuracy:0.8064406395282911 f1:0.5636932030435894

In [497]: Forest= RandomForestClassifier(random_state=1)
          Forest.fit(X_train,Y_train)
          performance_met(Forest,X_train,Y_train,X_test,Y_test)

          train score: accuracy:0.79907136733705 f1:0.7174487648913149
          test score: accuracy:0.806475529232115 f1:0.563789001828441
```

Notes

1. The association algorithm take data as bool not int
2. The method we descoverd to transform data to appropriate shape to model is better than the method suggested by AI