Prediction of the product category

```
1) EDA - Exploratory Data Analysis
2) Data Pre-Processing
3) Using NLP librarires for getting the product transmission
4) Decision of ML models
5) Accuracy and Performance metrics to understand the performance of the model
```

In [17]:

```
#Importing the required libraries

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
import math

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [72]:

```
product_data = pd.read_csv("E:/Projects/MachineLearning/Machine-Learning/Hacker Earth/Great
```

In [73]:

```
product_data.columns
```

Out[73]:

```
Index(['Inv_Id', 'Vendor_Code', 'GL_Code', 'Inv_Amt', 'Item_Description',
       'Product_Category'],
      dtype='object')
```

In [74]:

```
product_data.shape
```

Out[74]:

```
(5566, 6)
```

In [75]:

```
product_data['Inv_Amt'][:2]
```

Out[75]:

```
0    83.24
1    51.18
Name: Inv_Amt, dtype: float64
```

In [76]:

```
product_data.Inv_Id[:2]
```

Out[76]:

```
0    15001
1    15002
Name: Inv_Id, dtype: int64
```

In [77]:

```
product_data.Inv_Amt[:2]
```

Out[77]:

```
0    83.24
1    51.18
Name: Inv_Amt, dtype: float64
```

In [78]:

```
product_data.Product_Category.values
```

Out[78]:

```
array(['CLASS-1963', 'CLASS-1250', 'CLASS-1274', ..., 'CLASS-1721',
       'CLASS-1652', 'CLASS-1758'], dtype=object)
```

In [79]:

```
onvert the string data to numberical data. Since it is going to be the categorical data, we
```

In [80]:

```
# Cleaning the data

product_desc = product_data.Item_Description
target = product_data.Product_Category
```

In [81]:

```
month = ['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
data = []
print('Data cleaning and preprocessing is started')
for index,item in product_desc.items():
    item = item.replace('\\',' ')
    item = item.replace('/',' ')
    item = item.replace('''(Field Only)''','')
    item = re.sub(r"[^a-zA-Z0-9]+", ' ',item)
    item = re.sub("\d+","",item)
    item = item.replace('.',' ')
    item = item.lower()
    data.append(item)

print('')
print('Data cleaning and preprocessing is completed')
```

```
Data cleaning and preprocessing is started

Data cleaning and preprocessing is completed
```

In [82]:

```
#Let us go ahead with the stop words of English. Since the product description does not nee
# like to go ahead with the stop words implementation.. We will add the months in the stop
# Month looks not useful in the prediction of the product category.

stop_words = stopwords.words('english')
stop_words = stop_words + month


#We will go ahead with the word tokenizer. Not the sentence tokenizer.
final_data = []
for item in data:

    tokenize = word_tokenize(text=item)
    words = [ w for w in tokenize if not w in stop_words]
    text = ""
    for w in words:
        text = text + w + " "

    text = text.strip()

    final_data.append(text)
```

In [83]:

```
#Let us check the balanced vs imbalanced data... bu using the different selection methods.
product_data['Pre_Processed_Data'] = final_data
```

In [84]:

```
#Let`s break up the target variable as well...
target = []
for pc in product_data['Product_Category'].values:
    target.append(pc.replace("CLASS-",""))
product_data['Target'] = target
```

In [85]:

```
product_data.head(2)
```

Out[85]:

	Inv_Id	Vendor_Code	GL_Code	Inv_Amt	Item_Description	Product_Category	Pre_Proce
0	15001	VENDOR-1676	GL-6100410	83.24	Artworking/Typesetting Production Jun 2009 Champion Parts Inc SMAP Prototype and Comp Production/Packaging Design	CLASS-1963	artworking productio par prot productior
1	15002	VENDOR-1883	GL-2182000	51.18	Auto Leasing Corporate Services Corning Inc /Ny 2013-Mar Auto Leasing and Maintenance Other Corporate Services	CLASS-1250	auto leasin services ny a m corpora

In [86]:

```
# The final data contains the words without stop words and removed months.
```

Bag Of words..!

In [87]:

```
X_tr, Y_te, X_sc, Y_sc = train_test_split(final_data,target,test_size=0.3,random_state=42)
```

In [88]:

```
def bagOfWordsWithTrainTestData(train_data, test_data):
    count_vect = CountVectorizer(stop_words='english',max_features=500,min_df=10) #in sciki

    train_data_vectorized = count_vect.fit_transform(train_data)
    print("some feature names after transforming the TRAIN data", count_vect.get_feature_na

    test_data_vectorized = count_vect.transform(test_data)
    print("some feature names after transforming the TEST data", count_vect.get_feature_nam

    print("the type of count vectorizer ",type(train_data_vectorized))
    print("the shape of out text BOW vectorizer ",train_data_vectorized.get_shape())

    print("the type of count vectorizer ",type(test_data_vectorized))
    print("the shape of out text BOW vectorizer ",test_data_vectorized.get_shape())

    return count_vect, train_data_vectorized, test_data_vectorized
```

In [89]:

```
count_vect,train_data_vect, test_data_vect =  bagOfWordsWithTrainTestData(train_data=X_tr,t
```

```
some feature names after transforming the TRAIN data ['account', 'adr', 'ad
v', 'advertising', 'agency', 'air', 'akorn', 'akzo', 'alabama', 'alco']
some feature names after transforming the TEST data ['account', 'adr', 'ad
v', 'advertising', 'agency', 'air', 'akorn', 'akzo', 'alabama', 'alco']
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (3896, 250)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (1670, 250)
```

Logistic Regression Models

In [108]:

```
def performSimpleCV_On_Log_Regression(penalty, train_data, test_data, train_score, test_scc

    regularization_coeff = [10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2, 10**3, 10*

    score_m1 = []
    score_m2 = []

    for co_eff in regularization_coeff:

        logistic_model = LogisticRegression(penalty=penalty,C=co_eff,multi_class='ovr',clas
        logistic_model.fit(X=train_data,y=train_score)

        predicted_data_m1 = logistic_model.predict(X=test_data)

        model_accuracy = 0
        count=0;
        for i in range(len(predicted_data_m1)):
            if(predicted_data_m1[i]==test_score[i]):
                count+=1
        print('The accuracy of the model is %.2f for the co_eff  %.4f and the number of cor

        score_m1.append(predicted_data_m1)

    return score_m1
```

In [126]:

```
def performOptimalLogisticRegression(test_data,train_data,train_score,coeff,penalty):

    logistic_model = LogisticRegression(penalty=penalty,C=coeff,multi_class='ovr',class_wei
    logistic_model.fit(X=train_data,y=train_score)
    predicted_output = logistic_model.predict(X=test_data)
    return predicted_output
```

In [109]:

```
score_m1 = performSimpleCV_On_Log_Regression(penalty='l1',
    train_data=train_data_vect, test_data=test_data_vect,
    train_score=X_sc, test_score=Y_sc)
```

```
The accuracy of the model is 0.00 for the co_eff  0.0001 and the number of c
orrect values is 1
The accuracy of the model is 0.27 for the co_eff  0.0010 and the number of c
orrect values is 449
The accuracy of the model is 0.98 for the co_eff  0.0100 and the number of c
orrect values is 1638
The accuracy of the model is 1.00 for the co_eff  0.1000 and the number of c
orrect values is 1664
The accuracy of the model is 1.00 for the co_eff  1.0000 and the number of c
orrect values is 1665
The accuracy of the model is 1.00 for the co_eff  10.0000 and the number of
correct values is 1665
The accuracy of the model is 1.00 for the co_eff  100.0000 and the number of
correct values is 1665
The accuracy of the model is 1.00 for the co_eff  1000.0000 and the number o
f correct values is 1665
The accuracy of the model is 1.00 for the co_eff  10000.0000 and the number
of correct values is 1665
```

In [110]:

```
score_m1 = performSimpleCV_On_Log_Regression(penalty='l2',
    train_data=train_data_vect, test_data=test_data_vect,
    train_score=X_sc, test_score=Y_sc)
```

```
The accuracy of the model is 0.94 for the co_eff  0.0001 and the number of c
orrect values is 1563
The accuracy of the model is 0.97 for the co_eff  0.0010 and the number of c
orrect values is 1628
The accuracy of the model is 0.99 for the co_eff  0.0100 and the number of c
orrect values is 1660
The accuracy of the model is 1.00 for the co_eff  0.1000 and the number of c
orrect values is 1663
The accuracy of the model is 1.00 for the co_eff  1.0000 and the number of c
orrect values is 1664
The accuracy of the model is 1.00 for the co_eff  10.0000 and the number of
correct values is 1665
The accuracy of the model is 1.00 for the co_eff  100.0000 and the number of
correct values is 1665
The accuracy of the model is 1.00 for the co_eff  1000.0000 and the number o
f correct values is 1665
The accuracy of the model is 1.00 for the co_eff  10000.0000 and the number
of correct values is 1665
```

Observation...!

From the above model, we could conclude that the model - Logistics Regression could be used very well for the given data set.

The model works fine with both the regularizer. Hence, I would like to choose "L1" regularizer with 10 co-efficients.

Now, use the same model configuration for predicting the test data....

Test data will undergo the same pre-processing technique....!

In [113]:

```
test_df = pd.read_csv("E:/Projects/MachineLearning/Machine-Learning/Hacker Earth/Great Indi
```

In [114]:

```
month = ['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec']
test_data = []
print('Data cleaning and preprocessing is started')
for index,item in test_df.Item_Description.items():
    item = item.replace('\\',' ')
    item = item.replace('/',' ')
    item = item.replace('''(Field Only)''','')
    item = re.sub(r"[^a-zA-Z0-9]+", ' ',item)
    item = re.sub("\d+","",item)
    item = item.replace('.',' ')
    item = item.lower()
    test_data.append(item)

print('')
print('Data cleaning and preprocessing is completed')
```

```
Data cleaning and preprocessing is started

Data cleaning and preprocessing is completed
```

In [115]:

```
#Let us go ahead with the stop words of English. Since the product description does not nee
# like to go ahead with the stop words implementation.. We will add the months in the stop
# Month looks not useful in the prediction of the product category.

stop_words = stopwords.words('english')
stop_words = stop_words + month


#We will go ahead with the word tokenizer. Not the sentence tokenizer.
final_test_data = []
for item in test_data:
    tokenize = word_tokenize(text=item)
    words = [ w for w in tokenize if not w in stop_words]
    text = ""
    for w in words:
        text = text + w + " "
    text = text.strip()
    final_test_data.append(text)
```

In [116]:

```
test_df['Final_Pre_Processed_Data'] = final_test_data
```

In [124]:

```
count_vect,train_data_vect, test_data_vect =  bagOfWordsWithTrainTestData(train_data=final_
```

```
some feature names after transforming the TRAIN data ['account', 'acme', 'ad
r', 'adv', 'advertising', 'agency', 'air', 'airtex', 'akorn', 'akzo']
some feature names after transforming the TEST data ['account', 'acme', 'ad
r', 'adv', 'advertising', 'agency', 'air', 'airtex', 'akorn', 'akzo']
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (5566, 301)
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (2446, 301)
```

In [127]:

```
predicted_output = performOptimalLogisticRegression(test_data=test_data_vect,train_data=tra
```

In [129]:

```
class_predicted = ["CLASS-"+i for i in predicted_output]
```

In [131]:

```
test_df['Product_Category'] = class_predicted
```

In [133]:

```
test_df.head(2)
```

Out[133]:

	Inv_Id	Vendor_Code	GL_Code	Inv_Amt	Item_Description	Final_Pre_Processed_Data	Produ
0	15003	VENDOR-2513	GL-6050310	56.13	Travel and Entertainment Miscellaneous Company Car (Field Only) Ground Transportation Miscellaneous Company Car (Field Only) Oct2011 Fortune National Corp	travel entertainment miscellaneous company car ground transportation miscellaneous company car fortune national corp	
1	15008	VENDOR-1044	GL-6101400	96.56	Final Site Clean Up Store Construction Advanced Micro Devices Inc Oct2011 General Requirements General Contractor	final site clean store construction advanced micro devices inc general requirements general contractor	

In [135]:

```
submission_data = pd.DataFrame({'Inv_Id':test_df['Inv_Id'],'Product_Category':test_df['Proc
```

In [141]:

```
n_data.to_csv('E:/Projects/MachineLearning/Machine-Learning/Hacker Earth/Great Indian Scient
```

In [ ]:

```

```