Beverage PH Analysis - PY Script

Coffy Andrews-Guo

2022-07-02

## PYSCRIPT - Beverage Manufacturing Company

### 1. Required R Libraries / Import Python Packages

The ETL process will incorporate two programming languages, R and Python, and used to model predictions.

# R packages  
suppressPackageStartupMessages(library(tidyverse)) #data transformations  
suppressPackageStartupMessages(library(reticulate)) #bridges Python and R

# Python libraries/modules  
import warnings  
warnings.simplefilter(action = 'ignore', category = FutureWarning)  
warnings.filterwarnings('ignore')  
def ignore\_warn(\*args, \*\*kwargs):  
 pass  
  
warnings.warn = ignore\_warn #ignore annoying warning (from sklearn and seaborn)  
  
import pandas as pd #data processing  
import numpy as np #linear algebra  
import openpyxl  
import pyreadr #read and write R RData and Rds files into/from pandas dataframes  
import matplotlib.pyplot as plt #create graphs  
from IPython import get\_ipython #to see the graphs in the output  
import seaborn as sns

### 3. Load Data and Save Original Data Files

pydf = pd.read\_excel("StudentData - TO MODEL.xlsx")  
pydf.head(3)  
  
#remove the space from column name

## Brand Code Carb Volume Fill Ounces ... Alch Rel Carb Rel Balling Lvl  
## 0 B 5.340000 23.966667 ... 6.58 5.32 1.48  
## 1 A 5.426667 24.006667 ... 6.56 5.30 1.56  
## 2 B 5.286667 24.060000 ... 7.66 5.84 3.28  
##   
## [3 rows x 33 columns]

pydf.columns = pydf.columns.str.replace(' ', '')  
print("Removed Space between Header Strings", pydf.head(3))

## Removed Space between Header Strings BrandCode CarbVolume FillOunces ... AlchRel CarbRel BallingLvl  
## 0 B 5.340000 23.966667 ... 6.58 5.32 1.48  
## 1 A 5.426667 24.006667 ... 6.56 5.30 1.56  
## 2 B 5.286667 24.060000 ... 7.66 5.84 3.28  
##   
## [3 rows x 33 columns]

pyde = pd.read\_excel("StudentEvaluation- TO PREDICT.xlsx")  
pyde.head(3)  
  
#remove the space from column name

## Brand Code Carb Volume Fill Ounces ... Alch Rel Carb Rel Balling Lvl  
## 0 D 5.480000 24.033333 ... 6.56 5.34 1.48  
## 1 A 5.393333 23.953333 ... 7.14 5.58 3.04  
## 2 B 5.293333 23.920000 ... 6.52 5.34 1.46  
##   
## [3 rows x 33 columns]

pyde.columns = pyde.columns.str.replace(' ', '')  
print("Removed Space between Header Strings", pyde.head(3))

## Removed Space between Header Strings BrandCode CarbVolume FillOunces ... AlchRel CarbRel BallingLvl  
## 0 D 5.480000 24.033333 ... 6.56 5.34 1.48  
## 1 A 5.393333 23.953333 ... 7.14 5.58 3.04  
## 2 B 5.293333 23.920000 ... 6.52 5.34 1.46  
##   
## [3 rows x 33 columns]

**View the dataset size:**

print("\n Model dataframe shape\n", pydf.shape, "\n\nPredict dataframe shape\n", pyde.shape)

##   
## Model dataframe shape  
## (2571, 33)   
##   
## Predict dataframe shape  
## (267, 33)

### 4. Exploratory Data Analysis (EDA)

#### 4.1 Descriptive Statistics

**View the Data frame rows:**

#check data distribution on first 4 rows and 5 columns  
print("Train dataframe:\n", pydf.iloc[:4, :5].round(2), "\n\nPredict dataframe:\n", pyde.iloc[:4, :5].round(2))

## Train dataframe:  
## BrandCode CarbVolume FillOunces PCVolume CarbPressure  
## 0 B 5.34 23.97 0.26 68.2  
## 1 A 5.43 24.01 0.24 68.4  
## 2 B 5.29 24.06 0.26 70.8  
## 3 A 5.44 24.01 0.29 63.0   
##   
## Predict dataframe:  
## BrandCode CarbVolume FillOunces PCVolume CarbPressure  
## 0 D 5.48 24.03 0.27 65.4  
## 1 A 5.39 23.95 0.23 63.2  
## 2 B 5.29 23.92 0.30 66.4  
## 3 B 5.27 23.94 0.19 64.8

#### 4.2 Formatting Data / Filtering Data / Cleaning Data

**Missing Values - Categorical Variable**

#replace NaN values with string in specific column  
pydf[['BrandCode']] = pydf[["BrandCode"]].fillna("Missing")   
pyde[['BrandCode']] = pyde[["BrandCode"]].fillna("Missing")

**Grouped by Brand Code**

#view dataframe first 6 columns based on 'BrandCode` summary grouping  
g = pydf.groupby('BrandCode').sum()  
g.iloc[:, :6].round(3)

## CarbVolume FillOunces PCVolume CarbPressure CarbTemp PSC  
## BrandCode   
## A 1589.727 6834.900 76.871 20160.60 41238.80 22.276  
## B 6555.607 29324.467 345.093 82586.40 172719.60 105.704  
## C 1605.960 7195.460 87.059 20211.28 42598.20 27.110  
## D 3366.880 14589.560 159.138 42631.00 86089.38 48.984  
## Missing 634.903 2783.667 33.504 7885.00 16440.60 10.574

**Removing NaN values in the Numerical Variables**

from sklearn.impute import SimpleImputer

The data set is relatively small and the missing values will remain. The NaN’s will be replaced with a constant: zero.

# 'np.nan' signifies that we are targeting missing values  
# and the strategy we are choosing is replacing it with a 'constant'  
imputer = SimpleImputer(missing\_values=np.nan, strategy='constant', fill\_value = 0)  
  
imputer.fit(pydf.iloc[:, 1:])

## SimpleImputer(fill\_value=0, strategy='constant')

pydf.iloc[:, 1:] = imputer.transform(pydf.iloc[:, 1:])   
  
imputer.fit(pyde.iloc[:, 1:])

## SimpleImputer(fill\_value=0, strategy='constant')

pyde.iloc[:, 1:] = imputer.transform(pyde.iloc[:, 1:])  
  
#print the dataset  
#modeldf

**Creating Dummies**

#created new data frame with `BrandCode` dummy variables  
modeldf2 = pd.get\_dummies(pydf, columns=['BrandCode'])  
predictdf2 = pd.get\_dummies(pyde, columns=['BrandCode'])

**PH value scores and transformation**

#transform func.  
modeldf2['PH'] = modeldf2['PH'].apply(lambda value: 1 if value >= 7.35 else 0)

#### 4.3 PreProcessing

**PreProcessing - Creating Train / Test data**

#train set  
x\_model = modeldf2["PH"] #target variable / dependent variable  
modeldf2.drop("PH", inplace = True, axis = 1) #independent variables  
modeldf2.shape

## (2571, 36)

x\_model.shape

## (2571,)

Removing PH column from Predict data set because the predicted values are not present.

**Perform Train / Test split**

from sklearn.model\_selection import train\_test\_split  
x\_train,x\_test,y\_train,y\_test = train\_test\_split(modeldf2,x\_model,test\_size=0.2,random\_state=40)

View the train/test data frame:

x\_train.shape , x\_test.shape , y\_train.shape , y\_test.shape

## ((2056, 36), (515, 36), (2056,), (515,))

print("\n\nSample view of x-train set:\n", x\_train.iloc[:4, :5].round(2))

##   
##   
## Sample view of x-train set:  
## CarbVolume FillOunces PCVolume CarbPressure CarbTemp  
## 1139 5.31 23.89 0.35 64.0 136.4  
## 1664 5.27 23.97 0.22 65.8 140.2  
## 1574 0.00 24.10 0.00 72.2 138.2  
## 2286 5.26 24.04 0.25 69.4 143.4

**Implemting SMOTE - Imbalance data**

from imblearn.over\_sampling import SMOTE

#imbalance data  
seed = 100  
k = 1  
smote = SMOTE(sampling\_strategy='auto', k\_neighbors = k , random\_state = seed)  
x\_train, y\_train = smote.fit\_resample(x\_train, y\_train)  
y\_train.value\_counts()

## 1 2053  
## 0 2053  
## Name: PH, dtype: int64

### 5. CLASSIFICATION MODELS

import copy as cp  
  
from sklearn.datasets import make\_classification  
  
from sklearn.linear\_model import LogisticRegression  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import RandomForestClassifier  
from xgboost import XGBClassifier  
from sklearn.ensemble import ExtraTreesClassifier  
  
from sklearn.model\_selection import StratifiedKFold, train\_test\_split, PredefinedSplit, GridSearchCV  
from sklearn.ensemble import StackingClassifier  
from sklearn.metrics import accuracy\_score  
  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from lightgbm import LGBMClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neural\_network import MLPClassifier  
  
  
from collections.abc import Iterable  
from more\_itertools import powerset  
  
import warnings  
warnings.simplefilter(action='ignore', category=FutureWarning)  
  
RANDOM\_STATE : int = 21  
TARGET : int = 60

#### 5.1 Build Stack Models

level\_0\_classifiers = dict()  
level\_0\_classifiers["logreg"] = LogisticRegression(random\_state=RANDOM\_STATE)  
level\_0\_classifiers["forest"] = RandomForestClassifier(random\_state=RANDOM\_STATE)  
level\_0\_classifiers["xgboost"] = XGBClassifier(use\_label\_encoder=False, eval\_metric='logloss', random\_state=RANDOM\_STATE)  
level\_0\_classifiers["naive\_bayes"] = GaussianNB()  
level\_0\_classifiers["knn"] = KNeighborsClassifier()  
level\_0\_classifiers["gbm"] = GradientBoostingClassifier()  
level\_0\_classifiers["lgbm"] = LGBMClassifier()  
level\_0\_classifiers["clf"] = DecisionTreeClassifier()  
level\_0\_classifiers["mlpc"] = MLPClassifier()  
  
  
level\_1\_classifier = ExtraTreesClassifier(random\_state=RANDOM\_STATE)

kfold = StratifiedKFold(n\_splits=5,   
 shuffle=True,   
 random\_state=RANDOM\_STATE)  
stacking\_model = StackingClassifier(estimators=list(level\_0\_classifiers.items()),   
 final\_estimator=level\_1\_classifier,   
 passthrough=True,   
 cv=kfold,   
 stack\_method="predict\_proba")

#### 5.2 Generate classification predictions on Base Models\*\*

level\_0\_columns = [f"{name}" for name in level\_0\_classifiers.keys()]  
stackfit = pd.DataFrame(stacking\_model.fit\_transform(x\_train, y\_train),   
 columns=level\_0\_columns + list(x\_train.columns))  
   
pd.melt(stackfit.iloc[:1, :9].round(6), var\_name="Model", value\_name="Prediction")

## Model Prediction  
## 0 logreg 1.000000  
## 1 forest 1.000000  
## 2 xgboost 0.999533  
## 3 naive\_bayes 1.000000  
## 4 knn 1.000000  
## 5 gbm 0.999978  
## 6 lgbm 0.999978  
## 7 clf 1.000000  
## 8 mlpc 1.000000

#### 5.3 Generate Secondary Level Model Predictions (Final Predictions)\*\*

transformed = pd.DataFrame(stacking\_model.transform(x\_test), columns=level\_0\_columns + list(x\_train.columns))  
transformed.iloc[:5, :9]

## logreg forest xgboost naive\_bayes knn gbm lgbm clf mlpc  
## 0 1.0 1.0 0.999533 1.0 1.0 0.999978 0.999978 1.0 1.0  
## 1 1.0 1.0 0.999533 1.0 1.0 0.999978 0.999978 1.0 1.0  
## 2 1.0 1.0 0.999533 1.0 1.0 0.999978 0.999978 1.0 1.0  
## 3 1.0 1.0 0.999533 1.0 1.0 0.999978 0.999978 1.0 1.0  
## 4 1.0 1.0 0.999533 1.0 1.0 0.999978 0.999978 1.0 1.0

y\_val\_pred = stacking\_model.predict(x\_test)  
#y\_val\_pred

### 5.4 Evaluation

accuracy = pd.DataFrame([[accuracy\_score(y\_test, cp.deepcopy(classifier).fit(x\_train, y\_train).predict(x\_test)) for name,   
 classifier in level\_0\_classifiers.items()]],  
 columns=level\_0\_columns,  
 index = ["accuracy"])  
accuracy.insert(4, "extra\_tree\_prediction", accuracy\_score(y\_test, y\_val\_pred))  
pd.melt(accuracy.round(6), var\_name="Model", value\_name="Prediction Accuracy")

## Model Prediction Accuracy  
## 0 logreg 0.996117  
## 1 forest 0.998058  
## 2 xgboost 0.994175  
## 3 naive\_bayes 0.998058  
## 4 extra\_tree\_prediction 0.994175  
## 5 knn 0.996117  
## 6 gbm 0.994175  
## 7 lgbm 0.994175  
## 8 clf 0.994175  
## 9 mlpc 0.994175