NLP Program

Course-End Project - Solution



**Help Zomato Predict Rating from the Review**

**Objective:** Using NLP and machine learning, make a model to predict the rating in a review based on the contents of the text review. This will help identify cases where there is a mismatch.

**Problem Statement:**

Zomato is India’s largest platform for discovering restaurants and ordering food. It operates in India as well as a few cities internationally. Bangalore is one of the biggest customer and restaurant bases for Zomato with 4 to 5 million users using the platform each month.

Users on the platform can also post reviews of restaurants, and provide a rating accompanying the review. The content in the reviews should ideally reflect the rating provided by the customer. In many cases, there is a mismatch, owing to multiple reasons where the rating does not match the customer review. The reviews and ratings matching is very important as it builds customer trust on the platform, and helps the user get an accurate picture of the restaurant.

You, as a data scientist, need to enable the identification and cleanup of such cases, to ensure the ratings are reflective of the reviews and that the reviews seem trustworthy to the customer. You will need to use NLP techniques in conjunction with Machine learning models to predict the rating from the review text.

**Domain:** Hospitality and internet

**Analysis to be done:** Perform specific data cleanup, build a rating prediction model using Random Forest technique and NLP.

**Content:**

rating: the rating given by the customer

review\_text: the text in the review

**Steps to perform:**

Perform clean up on the data – we’ll need to tweak the stop words (negative terms are important). Follow up with a Random Forest Regressor to predict the star rating given by the customers.

**Tasks:**

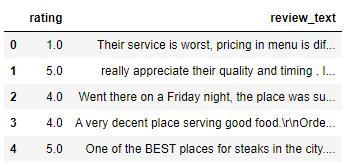
1. Load the data using read\_csv function from pandas package

import pandas as pd, numpy as np

import re

reviews0 = pd.read\_csv("Zomato\_reviews.csv")

reviews0.head()



1. Null values in the review text?
   1. Remove the records where the review text is null.

reviews0.describe(include="all")

#14 records are missing the review text

reviews1 = reviews0[~reviews0.review\_text.isnull()].copy()

reviews1.reset\_index(inplace=True, drop=True)

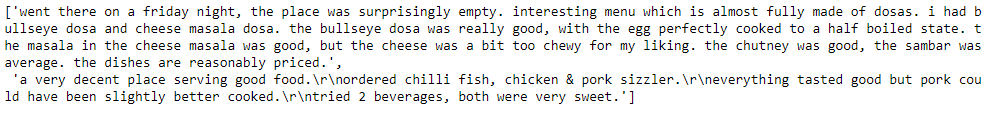
reviews0.shape, reviews1.shape



Converting to list for easy manipulation

reviews\_list = reviews1.review\_text.values

1. Perform cleanup on the data
   1. Normalize the casing

reviews\_lower = [txt.lower() for txt in reviews\_list]

* 1. Remove extra line breaks from the text

reviews\_lower = [" ".join(txt.split()) for txt in reviews\_lower]

* 1. Remove stop words
     1. Note: terms like ‘no’, ‘not’, ‘don’, ‘won’ are important, don’t remove
  2. Remove punctuation

from nltk.tokenize import word\_tokenize

reviews\_tokens = [word\_tokenize(sent) for sent in reviews\_lower]

print(reviews\_tokens[0])



from nltk.corpus import stopwords

from string import punctuation

stop\_nltk = stopwords.words("english")

stop\_punct = list(punctuation)

# Removing the specified terms from the stop words list

stop\_nltk.remove("no")

stop\_nltk.remove("not")

stop\_nltk.remove("don")

stop\_nltk.remove("won")

stop\_final = stop\_nltk + stop\_punct + ["...", "``","''", "====", "must"]

# Defining function to remove the stop words from a tokenized sentence

def del\_stop(sent):

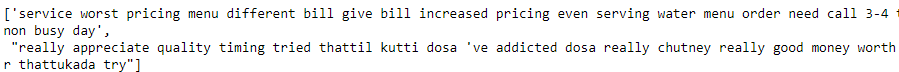
return [term for term in sent if term not in stop\_final]

# Applying on the review texts

reviews\_clean = [del\_stop(sent) for sent in reviews\_tokens]

# Joining back to form strings

reviews\_clean = [" ".join(sent) for sent in reviews\_clean]

reviews\_clean[:2] 

1. Separation into train and test sets
   1. Use train-test method to divide your data into 2 sets: train and test
   2. Use a 70-30 split

X = reviews\_clean

y = reviews1.rating

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state=42)

1. You will use TF-IDF values for the terms as feature to get into a vector space model
   1. Import TF-IDF vectorizer from sklearn
   2. Instantiate with a maximum of 5000 terms in your vocabulary

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max\_features = 5000)

* 1. Fit and apply on the train set
  2. Apply on the test set

len(X\_train), len(X\_test)



X\_train\_bow = vectorizer.fit\_transform(X\_train)

X\_test\_bow = vectorizer.transform(X\_test)

X\_train\_bow.shape, X\_test\_bow.shape

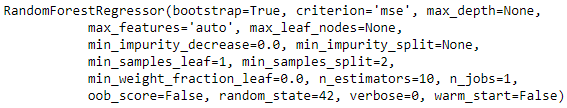


1. Model building: Random Forest Regressor
   1. Instantiate RandomForestRegressor from from sklearn (set random seed)
   2. Fit on the train data

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

learner\_rf = RandomForestRegressor(random\_state=42)

learner\_rf.fit(X\_train\_bow, y\_train)



* 1. Make predictions for the train set

y\_train\_preds = learner\_rf.predict(X\_train\_bow)

1. Model evaluation
   1. Report the root mean square error

from sklearn.metrics import mean\_squared\_error

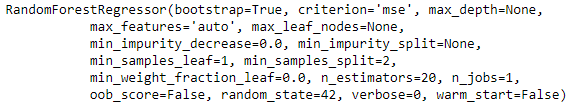
mean\_squared\_error(y\_train, y\_train\_preds)\*\*0.5



Increasing the number of trees.

learner\_rf = RandomForestRegressor(random\_state=42, n\_estimators=20)

learner\_rf.fit(X\_train\_bow, y\_train)



y\_train\_preds = learner\_rf.predict(X\_train\_bow)

mean\_squared\_error(y\_train, y\_train\_preds)\*\*0.5

1. Hyperparameter tuning
   1. Import GridSearch
   2. Provide the parameter grid to choose:
      1. max\_features – ‘auto’, ‘sqrt’, ‘log2’, 500
      2. max\_depth – 10, 15, 20, 25

from sklearn.model\_selection import GridSearchCV

# Instantiating the model with a random state

learner\_rf = RandomForestRegressor(random\_state=42)

# Create the parameter grid based on the results of random search

param\_grid = {

'max\_features': [500, "sqrt", "log2", "auto"],

'max\_depth': [10, 15, 20, 25]

}

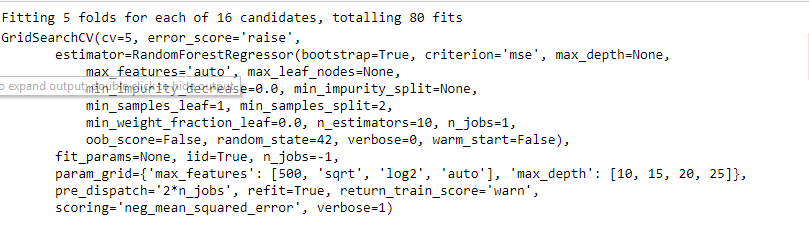
1. Find the parameters with the best mean square error in cross validation
   1. Choose the appropriate scoring as the metric for scoring
   2. Choose stratified 5 fold cross validation scheme

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator = learner\_rf, param\_grid = param\_grid,

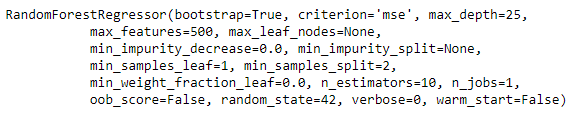
cv = 5, n\_jobs = -1, verbose = 1, scoring = "neg\_mean\_squared\_error" )

* 1. Fit on the train set

grid\_search.fit(X\_train\_bow, y\_train)

1. What are the best parameters?

grid\_search.best\_estimator\_



1. Predict and evaluate using the best estimator
   1. Use best estimator from the grid search to make predictions on the test set
   2. What is the root mean squared error on the test set?

y\_train\_pred = grid\_search.best\_estimator\_.predict(X\_train\_bow)

y\_test\_pred = grid\_search.best\_estimator\_.predict(X\_test\_bow)

mean\_squared\_error(y\_train, y\_train\_pred)\*\*0.5



mean\_squared\_error(y\_test, y\_test\_pred)\*\*0.5



1. Can we identify mismatch cases?
   1. Make a rule based on the predicted value and the actual value that identifies mismatch cases (e.g. difference in actual and predicted being more than a cutoff)

res\_df = pd.DataFrame({'review':X\_test, 'rating':y\_test, 'rating\_pred':y\_test\_pred})

* 1. How many such cases do you see?
  2. Are all these genuine mismatch cases?

res\_df[(res\_df.rating - res\_df.rating\_pred)>=2].shape



res\_df[(res\_df.rating - res\_df.rating\_pred)>=2]



While some of these are correctly labeled, there are some that are actually mismatched cases. These are the cases which can be reviewed and cleaned up. This model can greatly help reduce the consideration set for cleanup and help Zomato identify mismatch cases and clean them.