Wine Recommender & Wine Point Regressor

Dec 2019

The **goal** of this project is to analyze the Wine Review dataset from Kaggle.

- Original dataset location - https://www.kaggle.com/zynicide/wine-reviews .
- The winemag-data-130k-v2.csv file (50.46 MB) from this location that contains 14 columns is what will be used.
- First goal is to predict the points rated by WineEnthusiast on a scale of 1-100 using 2 separate regression models.
- Second goal is to suggest top 5 good wineries to a customer interested in purchasing a fruity pinot noir, with a price less than 20 dollars.

Approach would be to use Python's ScikitLearn and NLTK packages do the necessary data pre-processing,cleansing, regression based predictions and text based nearest neighbor detection.

- Duplicate rows and columns that are less relevant will be removed. Wine review text present in the 'description' field will be lemmatized , tokenized and encoded using tfidf vectorization. Encoding will be used to convert categorical variables as well, before splitting the dataset into test and train and then applying 2 various regression techniques to predict the points rated by WineEnthusiast on a scale of 1-100.
- Filtering on Wine 'variety' and 'price' columns along with text matching of the 'description' field to a 'fruity' taste using cosine distance using 'NearestNeighbors' from ScikitLearn package will be used to identify and suggest the best wineries that offer fruity pinot noir, with a price less than 20 dollars.

Result of the above approach would be the following:

- Predictions of wine points by 2 different regression models along with the corresponding metrics like RMSE values
- 5 best wineries that offer fruity pinot noir priced less than $20

Data Processing

Describe how you process the data so that it can be analyzed by a regression model. This includes, but not limited to, processing text data.

- Wine data from CSV file is **loaded** into pandas dataframe using 'read_csv()' function as the initial step. Pandas was chosen to ease the data pre-processing steps expecially those related to the wine review field 'description'.
- **Duplicate rows** from the dataframe are removed using 'drop_duplicates()' function which removes rows that are exact duplicates with respect to all columns.
- Columns 'taster_name','taster_twitter_handle' that do not seem to be much relevant to the usecases being considered . These are considered **irrelevent columns** and are removed using 'drop()' function.
- Columns 'designation','region_2' are also dropped as large number of rows had **null values** in these columns (based on results from 'dataframe.isnull().sum()' function) and also due to the fact that there are other **correlated columns** like 'variety' and 'region_1' respectively that hold almost the same information held in these fields.

- Categorical variables 'country', 'province', 'region_1', 'variety','winery' are **encoded to create dummy variables** by leveraging the pandas inbuilt function and combined with the 'price' variable.
- Review text in the 'description' field is cleaned by **removing stopwords, special characters, numbers and punctuations**.
- Review text in the 'description' field is also cleaned by performing **pos tagging, tokenization and also word lemmatization** using 'WordNetLemmatizer' from NLTK package in python.
- Cleaned up 'description' text is **converted in to a text vector** using NLTK's 'TfidfVectorizer' which performs the **tf-idf transformation** which considers term frequency and inverse document frequency while performing the vectorization.
- Encoded categorical variables along with 'price' and text vectors from tf-idf transformation are **combined to create an input vector array** and passed through ScikitLearn's 'Preprocessing.StandardScaler()' so that an array of scaled vectors can be created.
- This cleaned, encoded, vectorized and scaled input array can be **split into Train data and Test data** (75 - 25 proportion) and will be fed to the Python's scikit learn **regression models** that will then perform the wine point predictions based on input data.

Descriptive statistics of Data

- Below results from hidden code shows some of the statistics about the data during each and every step of data pre-processing, until the data is ready for the regression models.
- Examining the **duplicate rows, categorical variables and their distinct values** along with **variable importance** lead to dropping duplicate rows as well as columns 'taster_name','taster_twitter_handle' , 'region_2', 'designation'
- Examining the **numerical variables** - points and price - indicate that points varies between 80 and 100 where as the variance in proce is very very huge ranging from a 4 dollars to thousands of dollars up to even 3300.
- The above finding about points also lead to the decision to not encode 'price' variable into **dummy variables** as that would increase the size of the input vecor considerably.
- The size of the **final vector** useed as input to the test-train split function is also displayed below
- For all the details mentioned above as well as data processing refer the **hidden code** where as output for these steps is displayed below

Hidden Code Here

```
Total number of rows in the original dataset:  129971
Total number of columns in the original dataset:  13

No: of exact duplicate rows dropped from dataset:  9983
No: of rows remaining after removing duplicate rows :  119988

Columns remaining after removing less important columns are:  9
Index(['country', 'description', 'points', 'price', 'province', 'region_1',
       'title', 'variety', 'winery'],
      dtype='object')

Inspecting all the numeric variables:
              points          price
count  119988.000000  111593.000000
```

```
mean       88.442236      35.620747
std         3.092915      42.103728
min        80.000000       4.000000
25%        86.000000      17.000000
50%        88.000000      25.000000
75%        91.000000      42.000000
max       100.000000    3300.000000

Shape of encoded Dataframe:  (93580, 15533)
```

Hidden Code Here

```
Shape of text vector post all text processing and TF-IDF of description field
: (93580, 35928)
Shape of predictor X (Vectorized wine review text field + encoded other field
s) : (93580, 51461)
Shape of predicted y variable(points)  : (93580,)

Shape of text vector post all text processing and TF-IDF of description field
: (93580, 35928)
Shape of predictor X (Vectorized wine review text field + encoded other
fields) : (93580, 51461)

Shape of predicted y variable(points) : (93580,)
```

Regression Model Analysis

- **Models Used** - Two significantly different regression models have been chosen to perform regression analysis and predict the wine review points on the chosen wine data set
  - **Random forest regression model** and **Ridge regression model**
- **Data Processing** - Both the models were trained on Train data which is a 75% of the total data and predictions were done on the remaining 25% of data. Data was pre-processed to vectorize the text review field 'description' and encode the other categorical variables as needed. (details part of the hideen code)
- **Technology** - Random Forest and Ridge Regression libraries from Python's ScikitLearn Machine Learning package have been used to perform the training and predictions, as shown in the code below. NLTK library was used to do the text field processing, TF_IDF and vectorization (details in the hidden code)
- **Tuning** - Random forest model was tuned by defining the parameters maximum tree depth, number of trees or iterators to be used (details in code below). Ridge Regression model was tuned by going over a set of regularization parameters or alpha values and choosing the best among those for further predictions (details in code below).
- **Validation** - To evaluate the models, matrics provided by ScikitLearn's 'metrics' library has been used. Considering that we are building regression models, matrices used for model evaluation are Root Mean Squared Error, R Squared values , Variance Score,Mean Absolute Error, Mean Squared Error. Results of these validations are displayed below
- **Comaparison of two models** - Based on the RMSE values , other model parameters and also the regression plots (see below) of both the regression models, Random forest regressor seems to be doing slightly better and have better regression results while predicting the 'points' related to wine review entries in the dataset.

- **Interpretation of results** - Although ridge regression was used along with tuning of regularization parameter, the Random forest seems to have performed better. Considering the fact that the same preprocessed data was used by both the models, the performance improvement in randomforest might be due to the fact that there was more tuning option available there. The flexibility to set the tree depth , maximum number of individual predictor trees to be used and above all the bootstrapping option seems where samples are drawn with replacement seems to have helped in increasing the prediction accuracy and especially RMSE. The fact that random forest uses an ensemble model that fits a number of defined trees on various sub-samples of the dataset and uses averaging to predict contributes to improving predictive accuracy and reduces over-fitting.

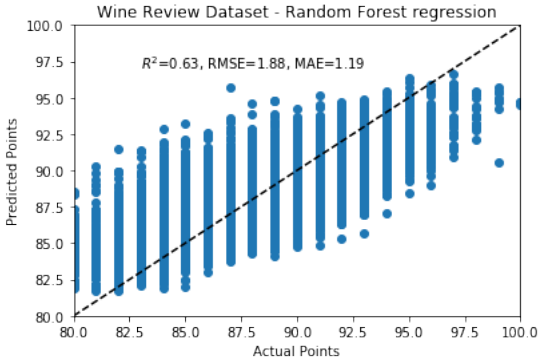
```
##############################################
#QUESTION 1 - REGRESSION MODELS
# Random Forest Regression Model
##############################################

from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics

# fit the random forest model on train data
rfModel = RandomForestRegressor(random_state=1000,n_estimators=70,max_depth = 50)
rfModel.fit(X_train, y_train)

# predict on test data using random forest model
y_pred_rf = rfModel.predict(X_test)

# prediction metrics values - random forest
print('\n Random Forest Regression - Prediction Metrics')
print('Explained Variance Score:', metrics.explained_variance_score(y_test, y_pred_rf))
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_rf))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_rf))
print('R Squared:', metrics.r2_score(y_test, y_pred_rf))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_rf)))

import warnings
warnings.filterwarnings('ignore')
```

```
 Random Forest Regression - Prediction Metrics
Explained Variance Score: 0.6347529842165074
Mean Absolute Error: 1.462358782799907
Mean Squared Error: 3.5480576744716594
R Squared: 0.634708478113786
Root Mean Squared Error: 1.8836288579419407
```

Hidden Code Here

Out[14]:

```
(80, 100)
```




```
##############################################
#QUESTION 1 - REGRESSION MODELS
# Ridge Regression Model
##############################################

from sklearn.linear_model import Ridge
from sklearn import metrics

# fit the ridge regression model on train data using cross validation
rrModel = Ridge(alpha=0.0001)
rrModel.fit(X_train, y_train)

print ('\n Ideal ridge regression regularization parameter selected by CV is: .0001')

# predict on test data using ridge regression model
y_pred_rr = rrModel.predict(X_test)

coeffUsed = np.sum(rrModel.coef_!=0)
print ("\n Number of features used by chosen ridge model:", coeffUsed)

# prediction metrics values - ridge regression
print('\n Ridge Regression - Prediction Metrics, alpha = .0001')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_rr))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_rr))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_rr)))
```

```
Number of features used by chosen ridge model: 51138

 Ridge Regression - Prediction Metrics, alpha = .0001
Mean Absolute Error: 1.4982960603756292
Mean Squared Error: 3.9418522261371476
Root Mean Squared Error: 1.985409838329897
```

Best Wineries for Fruity Pinot Noir Priced Less than 20

- The **goal** of this exercise is to suggest 5 different wineries to a customer who is interested in purchasing a fruity pinot noir, with a price less than 20 dollars.

Steps Followed

- To achieve this only those records from the dataset with 'variety' = 'pinot noir' and 'price' < $20 were filtered out
- The wine review text field 'description' of the records identified above was subjected to text analytics using NLTK package. After cleanup (stop word, punctuation removal, pos tagging , lemmatization etc) each review was matched to a query string ' fruity fruit' post TF_IDF vectorization and based on how good the match is , a score was assigned. This was done using ScikitLearn's TF-IDF vectorizer and Nearest Neighbor libraries and metrics used was cosine distance.

- Most matching reviews were segregated and the related wineries and associated points were extracted
- Wineries were grouped and total points for each winery were calculated
- Wineries with top 5 summed up points were assumed as those that offer fruity pinor noir process less than $20 and were returned as suggestions to the customer

Additional Assumptions for targeted recommendation

- It was assumed that all the wines were priced using the same dollar scale and the user was not particular about the country from which the wine came. It was also assumed that the user gave more preference to wineries that had multiple options for fruity pinot noir and was not looking for wineries that sold only the top rated fruits pinot noirs. This means in this exercise a winery A that sells 2 different kinds of fruity pinot noirs with points 90 and 91 would be considered as a better suggestion than a winery B that would offer only a single variety with point 95.

Hidden Code Here

```
Total number of wine reviews that will be analyzed:  129971
```

```
##################################################
#QUESTION 2 - Winery Recommendation
# Recommend 5 good wineries for fruity pinot noir wine
# having price < $20
##################################################
# what needs to be matched in the reviews?
#having both fruit and fruity in the query text seems to return better results
queryToMatch = "fruity fruit"
#identifying reviews that mention fruity taste using nearest neighbors  and cosine distance
#using tf-idf to fit, transform and identify nearest neighbors to query string
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
vectorizer = TfidfVectorizer(stop_words = stop_words,max_df = 1.0, min_df = 1)
vctrizedTFIDFRvwArr = vectorizer.fit_transform(rvwTextArrX)
queryTFIDF = vectorizer.fit(rvwTextArrX)
queryTFIDFVctr = queryTFIDF.transform([queryToMatch])
neigh = NearestNeighbors(n_neighbors=leng,metric = 'cosine')
neigh.fit(vctrizedTFIDFRvwArr)
allNieghbors = neigh.kneighbors(queryTFIDFVctr, return_distance=True)
#Find number of most matching reviews based on cosine distance
#Threshold > 0.9
for i,j in zip(np.arange(leng),allNieghbors[0][0]):
    if j > .95:
        break
identifiedNumofRws = i
#prints 'i' most matching reviews based on cosine distance
print("\nAround top {} reviews from those identified, can be good matches to the search \
text(fruit flavour)\n based on Nearest Neighbor Algorithm (metric = Cosine Distance)".format(i))
```

```
Subset of wine reviews for pinot noir < $20 to be analyzed for fruity flavour
: 1356
Around top 249 reviews from those identified, can be good matches to the sear
ch text(fruit flavour)based on Nearest Neighbor Algorithm (metric = Cosine Di
stance)
```

Hidden Code Here

```
Grouping reviews by winery name ,sum the total points for each winery and sor
t to find top 5 wineries
                winery  points
0      Nuiton-Beaunoy     432
1  Vignerons de Buxy      426
```

```
2          Eola Hills     347
3             McManis     341
4               Willm     266

Top 5 wineries for fruity pinot noirs priced < $20 are:
Nuiton-Beaunoy
Vignerons de Buxy
Eola Hills
McManis
Willm
```

Cross checking the findings - The details available in the below URLs indicate that the top two wineries identified by this code offer fruity Pinot Noirs among their top specialties and these are offered at these wineries at affordable prices (around 20). https://www.vivino.com/wineries/buxy , https://www.vivino.com/wineries/nuiton-beaunoy