Wine Variety Prediction based on Description

WITH SCIKIT-LEARN

Georgiy Voropayev

COMP 3710 ARTIFICIAL INTELLIGENCE

Introduction:

Over the years, as wine became more and more popular, different countries around the world have started the production of their own wine. By the time when winemaking spread beyond Europe, Italian and French governments started implementing strict laws that helped them authenticate traditional wines that were made in specific areas of their countries. The purpose of my study is to find out more about wine; what country produces which type of wine, what techniques are used for winemaking etc., and to build a machine learning program that would do classification of wine based on its description. In order to program such application, first, I had to answer following questions:

- How this alcohol beverage is categorized?
- What grape is used for which wine variety?
- What descriptions of taste of wine are used and what they really mean?

After answering these questions, I decided that I want to create an application that would take description of a specific wine, given by a professional sommelier, country and province of origin, and would give the wine variety as the output. For that I looked up and found a dataset that includes a wine variety along with the its country, sommelier's description, designation, province, and winery. In my program, I use details of wine such as country, description, and province, and try to predict the wine variety.

Background:

There are two major studies that were conducted regarding wine classification and machine learning. One of them is "Classifying wines by quality using machine learning". This project's goal is to try to discover the quality of a wine considering only its chemical properties taking a dataset that consists of column names such as **fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality,** and 1599 observations. Another study is called "Wine Ratings Prediction using Machine Learning"**.** In this study, the person tried to answer the question "Is it possible, through machine learning, to predict a wine rating (in point) based on its description?" He used the same dataset that I used in my project, and the results were impressive as the program predicted the quality of a wine with 97% of accuracy.

Project details:

To implement my idea of prediction of wine variety based on description I used Scikit-Learn machine learning library for Python. First, I had to find an appropriate dataset that would include different features of wine such country, province etc. along with its variety and how a sommelier describes it. It wasn't hard to do since Kaggle website provides many wine datasets, and the one I was looking for was at the top of the list. Next is the code. I had to create a separate class, called Wine, which has attributes information, consisting of country, description and province, and variety. These attributes I am using in order to add Wine objects to a list and split it later on for training and testing sets. I reduced my dataset from 150000 to 50000 samples for easier testing and also removed duplicated data for better performance of my classification models. Because my goal is to predict a wine variety based on country, description, and province, I had to look up an appropriate "Bag-of-Words". It is a simple and effective model that throws away all of the order information in the words and focuses

on the occurrence of words in a document. This is done by assigning each word a unique number. For my project I had to test both to see which one works best for my dataset. I used Count Vectorizer and TF_IDF Vectorizer. The difference between the two is that Count Vectorizer simply counts the word frequencies, as opposed to TF-IDF Vectorizer, in which the value increases proportionally to count, but is offset by the frequency of the word in the corpus; this helps to adjust for the fact that some words appear more frequently. After that, classification takes place where, once again, I had to decide on the most effective and appropriate model to use for training and predicting a variety of wine. Several models, including Linear SVM model, Stochastic Gradient Descent model along as well as with Radial Basis Function Kernel, Decision Tree model, Naïve Bayes model, and Logistic Regression model were tested to see, which one performs the best. As soon as the scores of each model with different vectorizers were obtained and the best ones were chosen, I decided to test these two models on whole 150000 samples and display the dataset with the predictions for visual comparison. At the end, I saved my best performing model using pickle.

Source Code:

```
import pandas as pd
import numpy as np

class Wine:
    def __init__(self, country, description, province, variety):
        self.information = str(country) + ', ' + str(description) + ', ' + str(pr
ovince)
        self.variety = variety

initial_data = pd.read_csv('Wine Reviews Project\\train_data.csv', encoding='utf-
8')
# print("Data before duplicates are removed: " , len(initial_data)) #* Data befor
e duplicates are removed:  49975

data = initial_data[initial_data.duplicated('description', keep=False)]
# print("Data after duplicates are removed: " , len(data)) #* Data after duplicat
es are removed:  10102

# Creating a list of Wine objects where attribute "information" consists of count
ry, description
# and province, and attribute variety is the variety
classif = []
for index, c in data.iterrows():
    classif.append(Wine(c['country'], c['description'], c['province'], c['variety
']))

# Splitting the data into training and testing sets
from sklearn.model_selection import train_test_split
training, test = train_test_split(classif, test_size=0.1, random_state=42)
```

```
# Splitting the training set into features and target
train_x = [x.information for x in training]
train_y = [x.variety for x in training]

# Splitting the testing set into features and target
test_x = [x.information for x in test]
test_y = [x.variety for x in test]

""" Count Vectorizer """
from sklearn.feature_extraction.text import CountVectorizer
c_vectorizer = CountVectorizer()
train_x_v = c_vectorizer.fit_transform(train_x)
test_x_v = c_vectorizer.transform(test_x)

""" TF-IDF Vectorizer """
from sklearn.feature_extraction.text import TfidfVectorizer
tf_vectorizer = TfidfVectorizer()
Train_x_v = tf_vectorizer.fit_transform(train_x)
Test_x_v = tf_vectorizer.transform(test_x)

''' CLASSIFICATION '''

""" Linear SVM """ # exited with code=1 in 1213.782 seconds
# !Takes forever to produce an output, so discard it
# from sklearn import svm
# classif_svc = svm.SVC(gamma='auto')
# classif_svc.fit(train_x_v, train_y)
# classif_svc.predict(test_x_v)

# print("\\nMean accuracy of SVC using Count Vectorizer:")
# print(classif_svc.score(test_x_v, test_y))

# classif_svc.fit(Train_x_v, train_y)
# classif_svc.predict(Test_x_v)
# print("\\nMean accuracy of SVC using TF-IDF Vectorizer:")
# print(classif_svc.score(Test_x_v, test_y))

""" Stochastic Gradient Descent """
from sklearn.linear_model import SGDClassifier
# classif_sgd = SGDClassifier(loss="hinge", penalty="l2", shuffle=True)
# classif_sgd.fit(train_x_v, train_y)
# classif_sgd.predict(test_x_v)
# print("Mean accuracy of SGD using Count Vectorizer:")
# print(classif_sgd.score(test_x_v, test_y))
```

```
# classif_sgd.fit(Train_x_v, train_y)
# classif_sgd.predict(Test_x_v)
# print("Mean accuracy of SGD using TF-IDF Vectorizer:")
# print(classif_sgd.score(Test_x_v, test_y))

#? Output before removing duplicates (49975 samples):
# Mean accuracy of SGD using Count Vectorizer:
# 0.7322929171668667
# Mean accuracy of SGD using TF-IDF Vectorizer:
# 0.7030812324929971

#? Output after removing duplicates (10102 samples):
# Mean accuracy of SGD using Count Vectorizer:
# 0.9574678536102869 #? <-- the best (SGD)
# Mean accuracy of SGD using TF-IDF Vectorizer:
# 0.9188921859545005

""" Radial Basis Function Kernel """
#! ValueError: array is too big; `arr.size * arr.dtype.itemsize` is larger than t
he maximum possible size.
# from sklearn.kernel_approximation import RBFSampler
# classif_rbf = RBFSampler(gamma=1, n_components=20990, random_state=1)
# features_train_x_v = classif_rbf.fit_transform(train_x_v)
# clf = SGDClassifier()
# clf.fit(features_train_x_v, train_y)
# clf.predict(test_x_v)

# print("Mean accuracy of SGD with RBF kernel using Count Vectorizer:")
# print(clf.score(test_x_v, test_y))

# clf.fit(Train_x_v, train_y)
# clf.predict(Test_x_v)
# print("Mean accuracy of RBF using TF-IDF Vectorizer:")
# print(clf.score(Test_x_v, test_y))

""" Decision Tree """
from sklearn.tree import DecisionTreeClassifier
# classif_dec = DecisionTreeClassifier()
# classif_dec.fit(train_x_v, train_y)
# classif_dec.predict(test_x_v)

# print("Mean accuracy of Decision Tree Clissifier using Count Vectorizer:")
# print(classif_dec.score(test_x_v, test_y))
```

```
# classif_dec.fit(Train_x_v, train_y)
# classif_dec.predict(Test_x_v)
# print("Mean accuracy of Decision Tree Clissifier using TF-IDF Vectorizer:")
# print(classif_dec.score(Test_x_v, test_y))

#? Output before removing duplicates (49975 samples):
# Mean accuracy of Decision Tree Clissifier using Count Vectorizer:
# 0.6818727490996399
# Mean accuracy of Decision Tree Clissifier using TF-IDF Vectorizer:
# 0.6634653861544618

#? Output after removing duplicates (10102 samples):
# Mean accuracy of Decision Tree Clissifier using Count Vectorizer:
# 0.9515331355093967
# Mean accuracy of Decision Tree Clissifier using TF-IDF Vectorizer:
# 0.9475766567754699

""" Naive Bayes """
# !This model requires bigger array size for my data, so skip it
# from sklearn.naive_bayes import GaussianNB
# classif_na = GaussianNB()
# classif_na.fit(train_x_v.toarray(), train_y.toarray())
# print(classif_na.predict(test_x_v[0]))

""" Logistic Regression """
from sklearn.linear_model import LogisticRegression
# classif_log = LogisticRegression()
# classif_log.fit(train_x_v, train_y)
# classif_log.predict(test_x_v)

# print("Mean accuracy of Logistic Regression using Count Vectorizer:")
# print(classif_log.score(test_x_v, test_y))

# classif_log.fit(Train_x_v, train_y)
# classif_log.predict(Test_x_v)
# print("Mean accuracy of Logistic Regression using TF-IDF Vectorizer:")
# print(classif_log.score(Test_x_v, test_y))

#? Output before removing duplicates (49975 samples):
# Mean accuracy of Logistic Regression using Count Vectorizer:
# 0.7486994797919168 #? <-- the best (Logistic Regression)
# Mean accuracy of Logistic Regression using TF-IDF Vectorizer:
# 0.6860744297719088

#? Output after removing duplicates (10102 samples):
```

```
# Mean accuracy of Logistic Regression using Count Vectorizer:
# 0.9525222551928784
# Mean accuracy of Logistic Regression using TF-IDF Vectorizer:
# 0.6983184965380811

''' Saving Model '''
import pickle

with open('./Wine Reviews Project/wine_classifier_sgd.pkl', 'wb') as f:
    pickle.dump(classif_sgd, f)

#TODO:      NOW LET'S TAKE OUR BEST MODELS
#TODO:      AND TEST THEM ON A BIGGER DATASET.
#TODO:      DISPLAY RESULT AND ACTUAL DATA FOR VISUAL COMPARISON

initial_test_data = pd.read_csv('Wine Reviews Project\\train_data - Copy.csv', en
coding='utf-8')
test_data = initial_test_data.dropna(subset=['description', 'country', 'province'
])

# print("initial_test_data: " , len(initial_test_data)) #* 150930
# print("test_data: " , len(test_data)) #* 150925

test_set = []
for index, c in test_data.iterrows():
    test_set.append(Wine(c['country'], c['description'], c['province'], c['variet
y']))

new_training, new_test = train_test_split(test_set, test_size=.8, random_state=42
, shuffle=False)

new_test_x = [x.information for x in new_test]
new_test_y = [x.variety for x in new_test]

c_testing = CountVectorizer()
train_x_v = c_testing.fit_transform(train_x)
new_test_x_v = c_testing.transform(new_test_x)

tf_testing= TfidfVectorizer()
Train_x_v = tf_testing.fit_transform(train_x)
new_Test_x_v = tf_testing.transform(new_test_x)

new_classif_sgd = SGDClassifier(loss="hinge", penalty="l2", shuffle=True)
# new_classif_sgd.fit(train_x_v, train_y)
# new_classif_sgd.predict(new_test_x_v)
```

```
# # print("Mean accuracy of SGD using Count Vectorizer:") #* 0.5213019711777372
# # print(new_classif_sgd.score(new_test_x_v, new_test_y))

new_classif_sgd.fit(Train_x_v, train_y)
new_classif_sgd.predict(new_Test_x_v)
# # print("Mean accuracy of SGD using TF-
IDF Vectorizer:") #* 0.5800066258075203 <-- the best
# # print(new_classif_sgd.score(new_Test_x_v, new_test_y))

new_classif_log = LogisticRegression()
new_classif_log.fit(train_x_v, train_y)
new_classif_log.predict(new_test_x_v)
# # print("Mean accuracy of Logistic Regression using Count Vectorizer:") #* 0.56
02865661752526 <-- the best
# # print(new_classif_log.score(new_test_x_v, new_test_y))

# new_classif_log.fit(Train_x_v, train_y)
# new_classif_log.predict(new_Test_x_v)
# # print("Mean accuracy of Logistic Regression using TF-
IDF Vectorizer:") #* 0.5000745403346033
# # print(new_classif_log.score(new_Test_x_v, new_test_y))

''' Display for compariosn '''

print("PRINTING LAST 10 ROWS OF TESTING SET:")
print(test_data.tail(n=10))
print("PREDICTING VARIETY WITH SGDClassifier (TF-IDF VECTORIZER): ")
print(new_classif_sgd.predict(new_Test_x_v[10:]))
print(" ")
print("PREDICTING VARIETY WITH LogisticRegression (Count Vectorizer): ")
print(new_classif_log.predict(new_test_x_v[10:]))
```

Results:

Several results must be explained for better understanding of what I was doing in my project. First, when several models were tested with different Vectorizers to see which one performs better, the results are as following:

```
#? Output before removing duplicates (49975 samples):
# Mean accuracy of Decision Tree Clissifier using Count Vectorizer:
# 0.6818727490996399
# Mean accuracy of Decision Tree Clissifier using TF-IDF Vectorizer:
# 0.6634653861544618

#? Output after removing duplicates (10102 samples):
# Mean accuracy of Decision Tree Clissifier using Count Vectorizer:
# 0.9515331355093967
# Mean accuracy of Decision Tree Clissifier using TF-IDF Vectorizer:
# 0.9475766567754699
```

```
#? Output before removing duplicates (49975 samples):
# Mean accuracy of Logistic Regression using Count Vectorizer:
# 0.7486994797919168 #? <-- the best (Logistic Regression)
# Mean accuracy of Logistic Regression using TF-IDF Vectorizer:
# 0.6860744297719088

#? Output after removing duplicates (10102 samples):
# Mean accuracy of Logistic Regression using Count Vectorizer:
# 0.9525222551928784
# Mean accuracy of Logistic Regression using TF-IDF Vectorizer:
# 0.6983184965380811
```

```
#? Output before removing duplicates (49975 samples):
# Mean accuracy of SGD using Count Vectorizer:
# 0.7322929171668667
# Mean accuracy of SGD using TF-IDF Vectorizer:
# 0.7030812324929971

#? Output after removing duplicates (10102 samples):
# Mean accuracy of SGD using Count Vectorizer:
# 0.9574678536102869 #? <-- the best (SGD)
# Mean accuracy of SGD using TF-IDF Vectorizer:
# 0.9188921859545005
```

From the pictures it is clear that Count Vectorizer is a better fit for my dataset as opposed to TF_IDF Vectorizer; moreover, SGDClassifier performed the best after removing duplicates (0.957% of accuracy), and Logistic Regression performed the best before removing duplicates (0.748% of accuracy). When it comes to other models, they failed due to different reasons (see the source code).

Moving forward, I used these two models to see how well they will predict on full 150000 samples. The results are quite surprising. SGDClassifier performed the best using TF_IDF Vectorizer with 0.58% of accuracy, whereas Logistic Regression performed the vest using Count Vectorizer with 0.56% of accuracy.

Finally, it is time to see these models in action. Below is are the results of both models with appropriate Vectorizer:

```
PRINTING LAST 10 ROWS OF TESTING SET:
              0 country  ...          variety                  winery
150920  150921.0   Italy  ...  Champagne Blend                 Letrari
150921  150922.0  France  ...  Champagne Blend                Jacquart
150922  150923.0   Italy  ...            Tocai       Ronchi di Manzano
150923  150924.0  France  ...  Champagne Blend                Jacquart
150924  150925.0  France  ...  Champagne Blend  Heidsieck & Co Monopole
150925  150926.0   Italy  ...       White Blend   Feudi di San Gregorio
150926  150927.0  France  ...  Champagne Blend               H.Germain
150927  150928.0   Italy  ...       White Blend               Terredora
150928  150929.0  France  ...  Champagne Blend                  Gosset
150929       NaN   Italy  ...      Pinot Grigio           Alois Lageder

[10 rows x 11 columns]
PREDICTING VARIETY WITH SGDClassifier (TF-IDF VECTORIZER):
['Cabernet Sauvignon' 'Chardonnay' 'Barbera' ... 'Fiano' 'Champagne Blend'
 'Pinot Grigio']

PREDICTING VARIETY WITH LogisticRegression (Count Vectorizer):
['Cabernet Sauvignon' 'Chardonnay' 'Red Blend' ... 'Gew�rztraminer'
 'Pinot Noir' 'Red Blend']

[Done] exited with code=0 in 52.924 seconds
```

Models displayed an array of last 10 wine varieties that they predicted based on the description that was given to them. To compare results, look at column “variety” and go down, comparing these varieties with the ones predicted by the models. As you can see, the numbers 0.58% and 0.56% don’t lie, and the models predicted the values half of the time.

Discussion and Conclusion:

To begin, I used two different datasets – one with 50000 samples, which I used to train my two models, and the other dataset with 150000 samples, purely for testing. I did that because I wanted to see how well my best models would perform on data they haven’t seen before. As a result, only half of the time the models could predict the true wine variety. I am sure they could have done better if I trained them on greater number of samples. To conclude, if the dataset consists of more than 100k samples, it is a great choice to use SGDClassifier with Count Vectorizer if you have to use sentences of words for prediction.

Future plan:

In the future, I am planning to enhance this application by providing either vintage or variety classification (which describes from what variety of grapes was the selected wine been made) as well as vinification (methods and style).

References:

Goutay, Olivier. “Wine Ratings Prediction Using Machine Learning.” Medium, Towards Data Science, 15 June 2018, https://towardsdatascience.com/wine-ratings-prediction-using-machine-learning-ce259832b321.

Vallantin, Lima. “Classifying Wines by Quality Using Machine Learning.” Medium, Medium, 19 Oct. 2018, https://medium.com/@limavallantin/classifying-wines-by-quality-1d5664df9a81.

“Classification of Wine.” Classification of Wine - Wine Categories, 3 June 2019, http://www.wine-facts.net/facts-about-wine/classification-of-wine/.