Wine quality forecasts

Edoardo Fantolino
Politecnico di Torino
Student id: s286008
s286008@studenti.polito.it

***Abstract*—This document contains my idea about a regression problem. I will build a model that will approximate wine's quality given the content of the review.**

I. Problem overview

This problem simulate a real and practical issue and nowadays this type of analysis involve more and more companies. We have a considerable amount of data and we want to use them to make predictions. Our data are made by reviews expressed by an expert on a specific wine. There are textual and categorical attributes. The metric used to understand the precision of the forecast will be the R^2 score.

Our dataset is divided into two main groups: development and test set. As the names suggest, the development set allows to create the model while the test will be used to evaluate the precision of the forecasts. First, we need to explore the content of our dataset. The attributes are: country, designation, description, province, region_1, region_2, variety, winery and quality (quality only for the development set). All this characteristic are in string format.

If we observe the data, we can see that some attributes embed the same concept. In fact, half of the total features that we have, describe a location (country, province, region_1, region_2). In order to escape from the classical type of encoding, I wanted to be more specific and I tried to convert these string information into a numerical one in the following way: we know that a location is uniquely identified by its latitude and longitude. So, I searched a package that allowed me to convert the name of an address into its geographical coordinates.Some of the problem related to this approach are: for example, if we have the state of Georgia, the system get confused between Georgia in the US and Georgia in Europe-Asia. Another issue was that the algorithm recognize the string 'NaN' as the city of Nan in Thailand. Moreover, sometimes this package didn't has the right longitude and latitude for the parameters "region_1" and "region_1". Then, the cardinality of some of this attributes was too large and the algorithm too slow and it make it unfeasible for the overall encoding of the dataset. So, I found useful information only for the "country" attribute.

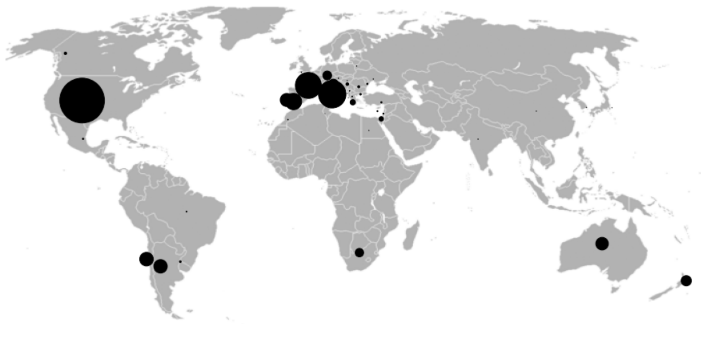
Fig. 1. Distribution of the wines around the world.

TABLE I
TOP 10 COUNTRIES FOR WINE PRODUCTION

country	number of wines
US	62397
Italy	23478
France	21098
Spain	8268
Chile	5816
Argentina	5631
Portugal	5322
Australia	4957
New Zealand	3320
Austria	3057

With the previous results, I realized a picture that describe the distribution around the globe for the wines in the different countries. You can see in Fig.1 the result obtained. We understand from this picture that the most of the wines came from the US and Europeans countries like Italy, France and Spain. Globally, the northern hemisphere has more production with respect to the southern. The total number of country is around 50, but most of them have a low productions and in the picture are represented by few pixels. In Tab.1 are shown the top 10 countries for production of wine. In the podium we have in order: US, Italy and France. The production of this first 10 states represent almost the 95% of the overall production in the world.

$$Top10\% = \frac{Top\ 10\ total\ production}{Global\ production} \approx 94.98\%$$

This means that $\frac{1}{5}$ of the countries produce 95% of the total global production.

Then I checked if the distribution of the development set and evaluation set was similar. I compared the numbers of countries and provinces present in the two sets and they follow the same distribution. I found the type of wine that were most common and you can see the results in Fig.2. Chardonnay, Pinot Noir and Cabernet Sauvignon lead the ranking. Another

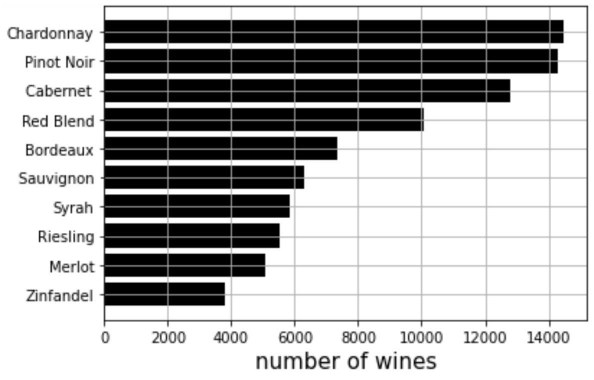

Fig. 2. Most popular wines.

check that I did was over the distribution of the quality. As we can see from Fig.3 it seems to follow a normal distribution. Wines with quality less than 20 and more than 80 are rare and represent exceptions or outliers. The majority of our wines are between this two threshold.

II. Proposed approach

In this section you will find my approach to solve the regression problem. The main steps are: Preprocessing, Model selection, Hyperparameters tuning.

A. Preprocessing

First, we need to clean and prepare our data.

We can see that the columns ”designation”, ”region_1” and ”region_2” contain ”NaN” values. Given that those are text fields, I replaced this value with a blank character. Next, we can observe that in the development set there are duplicated data. Basically, we can see that there are multiples rows with the exactly same entries. I decided to delete them and we reduce the size from a total of more that 120K down to 85K rows.

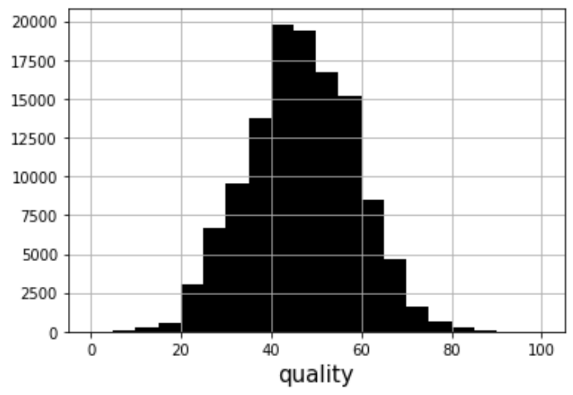

Fig. 3. Quality distribution of development set.

Then we have to face one of the most relevant issue: the encoding of the categorical data into numerical ones. We want to perform a transformation as follow:

$$string \longrightarrow number$$

For example, we can assign to different categories a progressive number. We will end up with one column and a cardinality equal to the total number of category. But in this way:

$$category_1 < category_2 < \cdots < category_n$$

We are assigning an order to the categories and this is not the best idea because we add a new concept that is not present in the original data.

So I searched more suitable techniques to encode this values. We can exploit different strategies:

- One-Hot Encoding
- Dummy Coding
- Effect Coding

Those type of encoding allow us two transform categorical data into numerical ones without introducing some fictitious information. The drawback of this techniques is that the number of columns generated are directly proportional to the number of categories. So, they are really useful when the cardinality of the categorical data are in the order of 10, otherwise the number of columns become unsuitable.

We have: 48 different countries, 455 provinces, 632 types of wine and 14809 wineries. The techniques mentioned above are unsuitable for such cardinalities. I made a research and I founded another techniques that allow to manage such high values of cardinalities, it is called ”Binary encoding”. I used this techniques to translate the ”country”, ”province”, ”region_1” and ”region_2” attributes. I used this techniques because allow us to significantly reduce the number of columns generated.

Then, to extract information to other categorical data like ”winery”, ”variety” and ”designation” I decided not to interpret them like category but like textual data. This approach allowed me not to fall in the **curse of dimensionality**. The use of TfidfVectorizer is made. Basically, we try to understand what are the most recurrent word and we create a matrix that contains the weight of this words for a given wine.

A similar approach was used fot the ”description” field. For this attribute I also used the nltk packages to analyze the text. I used the stopwords and even the word_tokenizer.

B. Model selection

The matrix obtained with the preprocessing described above is large and sparse. The number of nonzeros entry is low with respect to the dimensions of the DataFrame.

I tried to use was the Principal Component Analysis (PCA) in order to make easier the next computations, in fact, The PCA is a tool that allow us to reduce the number of features. It involve concepts of statistic like the covariance or correlation matrix and linear algebra concepts like eigenvalues and eigenvectors. The graph in Fig.4 tell us how many columns we

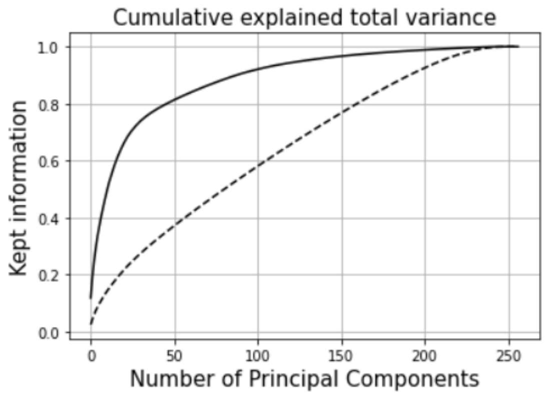

Fig. 4. Cumulative explained total variance in percentage. Dashed line represent the case of normalized data, the continuous line is for the non normalized data.

should keep if we want to maintain a certain amount of original information. I tried the normalized and non normalized case (using and not using the StandardScaler). As we can see, to obtain a suitable explained variance ratio I should kept a lot of columns and moreover it seems that the transformation of the data slowed down the next algorithm. I decided to discarded the idea.

For the model, the choice is to the Random Forest Regressor for multiple reasons:

1) we studied it during the course and I know how it should behave.
2) well know capacity to obtain valuable results in many different situations.
3) flexibility and possibility to perform hyperparameters tuning.
4) good performance in feasible times.

C. Hyperparameters tuning

The hyperparameters tuning is divided into two main steps.

The first one was concentrated on the optimization of the TfidfVectorizer function for the translation of: ”variety”, ”winery” and ”designation”. I tried different values for the parameters called ”min_df” and ”max_df”. Basically, we ignore the terms that have a document frequency strictly higher than max_df and strictly lower than min_df.

The second hyperparameters optimization that I performed is over the Random Forest Classifier. I build a gridsearch to optimaze the following elements: n_estimators, criterion, max_depth and max_features. For the last three the best choice is the default one. For the n_estimator is a trade off in terms of time, precision and computational cost. For example during the development phase of the pipeline I used n_estimators = 50 to make the whole process lighter.

parameter	values
max_depth	{None, 5, 10, 50}
n_estimator	{25, 50, 100}
criterion	{mse, mae}
max_features	{auto, sqrt, log2}

TABLE II
HYPERPARAMETERS CONSIDERED FOR THE RANDOM FOREST REGRESSOR

III. RESULTS

As stated before the model selected is the Random Forest Regressor. The optimal hyperparameter found are the default values. As regards to tfidfvectorizer every attribute has its own optimal values for max_df and min_df. This is linked to the fact that all this feature has a different content and cardinality.

The R^2 score is around 0.5160 and it takes 219 seconds.

This is possible due to a simple fact. If we give a look to our data in the development and evaluation set, we can observe that some rows (except the quality fields) are equals. We can fairly assume that for the corresponding rows the quality will be the same.

Let’s call our dataset X and our qualities y. We make the following assumption:

$$X_i = X_j \Longrightarrow y_i = y_j, \quad i \neq j$$

I am saying that if we have a row in the evaluation set equal to a row in the development set, we can deduce the quality without error. In fact, the R^2 score obtained in the platform is around 0.78. The check of the existence of this rows is quite costly, but the algorithm can be further optimized.

IV. THE SECOND MODEL

I have also build another model obtaining an overall result of 0.844, but that model was born from a mistake. The peculiar fact is that I used a wrong encoder for the features of our dataset, but the result is surprisingly good. There is not much to say about this model because it was all done by encoding the categories in the same way with the ”Label encoder” and dropping the attribute ”description”. An easy application of the Random Forest Regressor allows to get a R^2 value greater that 0.84. This label encoder should be used for labels only and not to encode our data. I propose the solution because of the good score and because it behave in this way.

V. DISCUSSION

In both cases, the algorithms go beyond the baseline. There are possible ways to further improve this models. For example, it will probably pay of give more importance to the ”description” field.

Moreover, if we could add some other feature like the price of the wine, I think that we could create a more precise algorithm because studies have found that the price associated to an item change the overall satisfaction of the consumer and this is directly linked to the presumed wine’s quality.