# *Wine Reviews Dataset* Regression Problem

Andrea Ghiglione
*Politecnico di Torino*
Student id: s287845
s287845@studenti.polito.it

***Abstract*—In this report we discuss one of the possible approaches to the *Wine Reviews Dataset* regression problem. In particular, the approach we selected consists in encoding categorical features in order to perform a regression and to extract knowledge from the row data. The proposed approach provided satisfactory results overcoming a naive baseline.**

## I. Problem overview

The problem consists in a regression task on the *Wine Reviews Dataset*, which is a collection of records regarding different wines' reviews, their productions areas and quality. The objective of this analysis is to correctly identify the wines' qualities, expressed by a floating point number. The dataset is divided into two parts:

- a *development* set, containing 120744 records for which it is present a quality measure.
- a *evaluation* set, composed by 30186 records.

In addition to the numerical target variable of the development set, expressed as the wine quality, each record of the sets has 8 categorical and textual features, which are:

- *country*: the country where the wine is from.
- *description*: review of the wine based on its flavors.
- *designation*: the grape variety of the wine.
- *province*: the province where the wine is from.
- *variety*: the type of grapes used to produce the wine.
- *winery*: the company which made the wine.
- *region_1*: the area in a country where the wine is grown.
- *region_2*: more specific geographical information about wine growing area

We can make some considerations on both sets. The wine quality in the development set feature is normally distributed with values comprised in a range from 0.0 to 100.0 as shown in Figure 1.

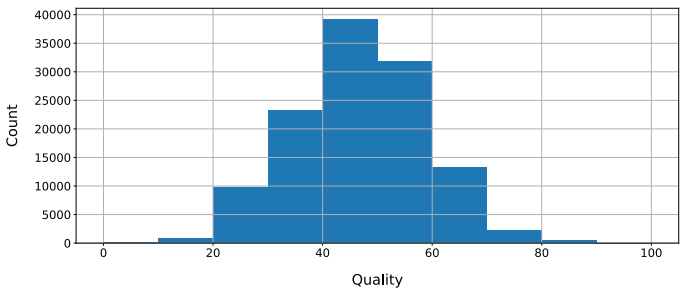


Fig. 1: Distributions of quality of the wines

Considering both sets it is evident how unbalanced they are; almost the half of the records is about US, while the cardinality of the country attribute is 48. However, Italy, France and Spain have an high number of records as well, as we can visualize in Figure 2. In both sets, many null values are present, affecting in particular *designation*, *region_1* and *region_2* features. In addition, there is a big number of duplicates. We will address these problems in the next section.

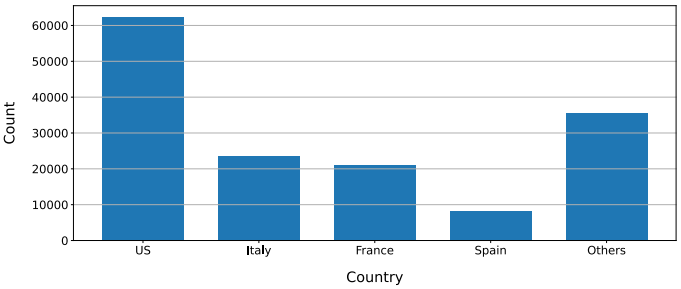


Fig. 2: Wine reviews per country considering both sets

## II. Proposed approach

### A. Preprocessing

First of all, we have to deal with *duplicates* in the development set. This set contains 120744 records and 35716 are duplicates; it is more than a quarter of the records. There are some cases in which we should keep the duplicates in the dataset, because the same information appearing twice might be itself an information. However, in this case there is no reason for which we should keep duplicates; the same wine review is a clear example of useless information. In addition to that, by removing duplicates our model will be able to better generalize the dataset. For the reasons that we mentioned we have decided to remove the duplicates.

We handled missing values as following: first, we removed the 5 records in which the attribute *country* or *province* was missing. Then, we noticed that, considering both datasets, *region_2* feature had 67800 null values. Since this feature provides a more specific geographical information we decided to replace, for each row, *region_1* with *region_2* record, when it is present[1]. After that, we dropped *region_2*, while we kept null values in *designation* and *region_1* features, which will be addressed later.

Analyzing the datasets we have seen that all the features are non-numerical; in particular, in order to perform a regression

[1]This operation affected records about US only, since region_2 attribute is null for remaining countries

we have to convert the categorical and textual attributes into numerical features. Working on the textual attributes *description* we noticed how *tf-idf*, a well-known way to process textual data, wasn't particularly effective. In addition to that, we preferred not enlarging more than necessary the dataset[2]. Indeed, we found out a way to encode some information about the description in one column only. As shown in Figure 3, we noticed a relationship between the wine quality and the length of the description. We encoded this information into a new column and then we dropped *description* feature.

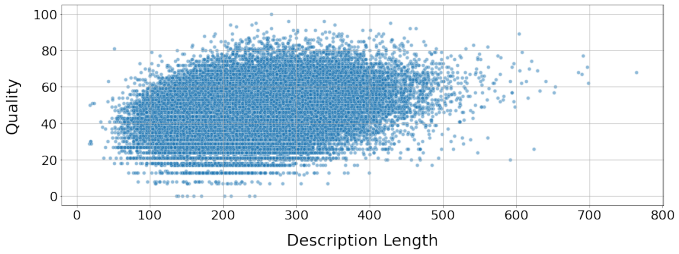


Fig. 3: Quality by description length

Next, we worked on *encoding* categorical features. We have considered several encoding techniques before choosing one, analyzing advantages and disadvantages. The first we took into account was *label encoding*, which consists in mapping each category to a different integer, usually in a ascending order. This method was discarded because, even if we don't enlarge the number of columns, we are introducing some relationships between features which may not be present, and this would affect our result. The second encoding technique that we analyzed was *one hot encoding*, which maps each category to a vector, which contains one '1' value and *n-1* '0' values, where *n* is the cardinality of the specific feature in which the category is included. Again, we discarded this method because, despite the fact that it doesn't introduce new wrong information since it uses binary vectors, this encoding technique would increase the number of columns of the dataset in a huge way. In fact, as shown in Table I, one hot encoding would create 44224 columns, introducing *sparsity* and causing the *curse of dimensionality* problem [1].

| Feature | Unique values |
|---|---|
| country | 48 |
| designation | 27800 |
| province | 444 |
| region_1 | 1206 |
| region_2 | 18 |
| variety | 603 |
| winery | 14105 |
| **Total** | **44224** |

Table I: Cardinality of categorical features in development set

The approach that has been used consists in grouping and encoding the categorical attributes by statistics computed on the target variable. In particular, for every feature, we have grouped by categories and we have calculated statistics, for every group, based on the wine quality of the group. By grouping and transforming data with *pandas* library [2] it has been possible to perform this encoding and, in addition to that, we have been able to avoid the *data leakage* problem[3]. The advantages of this approach are that the features are encoded in a coherent way, since the value is based on the target variable and, in addition to that, the number of features is relatively low. In fact, for every categorical feature, we introduced only *n* columns, where *n* is the number of statistics that we computed. We started by grouping features categories and performing an encoding based on measures of central tendency and measures of variability. At the beginning, we used *mean, median* and *standard deviation* measures. However, as shown in Figure 4, we noticed how *mean* and *median* were highly correlated, so we dropped the features regarding the median. In addition, we dropped *region_1_std*, since it was highly correlated with *region_1_mean*.

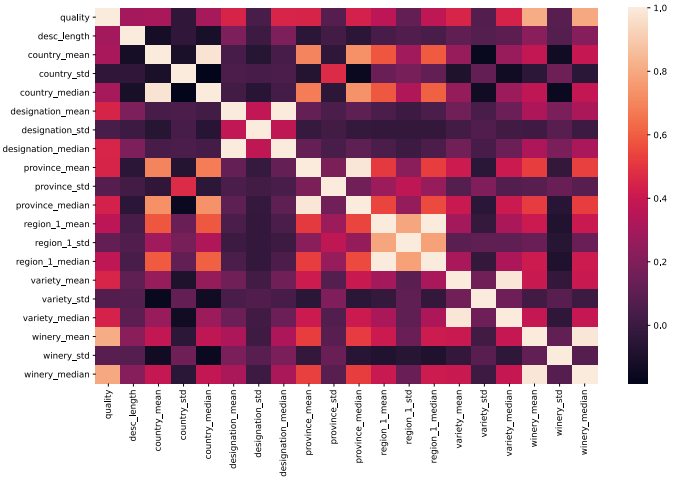


Fig. 4: Correlation among features

After having removed these features, in order to extract more information, we encoded features with two additional variability measures, *skewness* and *kurtosis*, proceeding in the same way like before. In conclusion, after features encoding and selection, as shown in Table II, the number of columns has become 25, which was manageable. All the null-values which remained in the dataset have been filled with zero values.

| Feature | Statistical measures computed |
|---|---|
| quality | - |
| description length | - |
| country | mean, std, skewness, kurtosis |
| designation | mean, std, skewness, kurtosis |
| province | mean, std, skewness, kurtosis |
| region_1 | mean, skewness, kurtosis |
| variety | mean, std, skewness, kurtosis |
| winery | mean, std, skewness, kurtosis |

Table II: Measures computed on the target variable by feature

[2]Even removing stop words and considering only best words the number of column would have been very big

[3]Working on the target variable we extracted information from the development set only, no information from the evaluation set was used

However, there is a drawback as well; by using the target variable as the basis to generate encoded features we are likely to encounter the *overfitting* problem. Indeed, while grouping data, it may happen to group very few records, which statistic is likely to be misleading for our model. This is very common in encodings such as *target encoding*, where, for all the categorical features, each category is replaced by the target variable mean for that specific category group [3]. Among the features computed through the target variable statistical measures, *mean* turned out to be an extremely important feature in our dataset, so we have taken this problem into account[4]. In particular, in order to limit overfitting we have re-computed the means in a very intuitive and efficient way, showed below:

$$mean = \frac{m_c \cdot r_c + m_g \cdot \eta}{r_c \cdot \eta}$$

This is a smoothed mean, where $m_c$ is the mean of the quality for a category, $r_c$ is the number of records of such a category, $m_g$ is the dataset global target variable mean and $\eta$ is the weight that we want to assign to the global target variable mean. With this small change in mean computation we make sure to better generalize the mean of groups with few records which may lead to overfitting. In fact, if the number of records in a group is lower than $\eta$, we will rely more on the global target variable mean, instead of the group mean. When $\eta$ is 0, the formula becomes the simple mean computation. So, $\eta$ is an hyperparameter of our preprocessing steps to be tuned.

### B. Model selection

The algorithm which has been used is *Random Forest* of *scikit-learn* library [4]. In this algorithm multiple base models are combined in order to improve stability and reduce overfitting. Random forest is an ensemble of decision trees built at training time. This algorithm selects *N* times a random sample with replacement from the training set and then it trains trees on these samples. Since feature subsets for each tree are randomly sampled, trees are *decorrelated* and, in the end, each tree prediction is taken into account in order to produce a single and reliable result.

For this regressor we identified the best configuration of hyperparameters with a grid search, which we will discuss in the following section

### C. Hyperparameters tuning

There are two sets of hyperparameters to be tuned:

- $\eta$ for the preprocessing steps
- random forest regressor parameters

In most of the cases, a global weight for the mean around 300 works well in order to limit overfitting [5]. For tuning this parameter we inspected the *mean squared error* for $\eta$ in a range between 0 and 550 as shown in Figure 5. We used a 80/20 train/test split, training a random forest with a naive configuration. After choosing $\eta$ parameter we run a grid search with 5-fold cross validation on random forest, based on the hyperparameters defined in Table III.

[4]Since means are very important in our dataset we made further work on them, while we let the other statistical measures unchanged

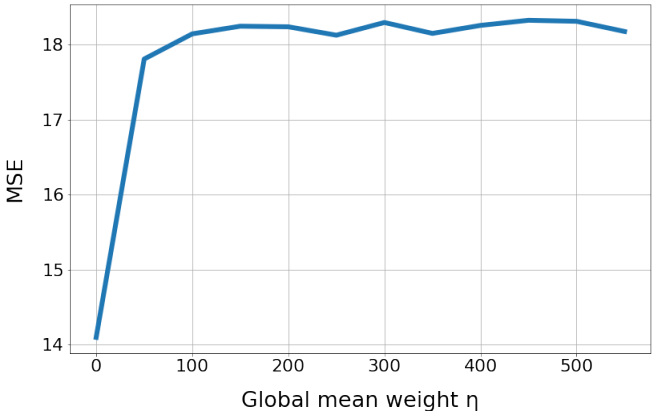


Fig. 5: Performance of random forest as $\eta$ varies

| Model | Parameter | Values |
|---|---|---|
| Preprocessing | $\eta$ | $0 \rightarrow 550$, step 50 |
| Random forest | *n_estimators* | $100 \rightarrow 500$, step 100 |
| | *max_depth* | {None, 5, 10, 15, 20} |
| | *max_features* | {sqrt, log2} |
| | *criterion* | {mse, gini, entropy} |

Table III: Hyperparameters considered

## III. Results

We selected the hyperparameter $\eta$ by looking at Figure 5. Even if assigning a null weight to the global mean would provide the lowest *mse*, and so apparently a better result, we can not rely on that information, because the encoding technique that we used is prone to overfitting. As showed in Figure 5 the *mse* becomes stable for global mean weights in a range between 100 and 550. Even if the error is bigger, this data is more reliable since we limited the overfitting caused by target means computed on small category groups, which otherwise would be misleading. A value of 450 for $\eta$ had turned out to be the best choice for our dataset and we used this parameter for running a grid search on random forest. The best configuration for random forest was found for {max_depth=None, n_estimators=500, criterion=mse,max_features=sqrt} with a $R^2$ score of 0.874. The public score obtained was of 0.833, collocating us in the middle of the leaderboard, even if not so far from the top. In addition, we evaluated the goodness of our solution by considering:

- *random guess solution*: we generated randomly the qualities for the evaluation set, obtaining a public score of -5.882, not surprising since it is random.
- *naive solution*: we used a *label encoder* for encoding the categorical features obtaining a score of 0.628.

## IV. Discussion

We can be satisfied by our result, since we overcame a naive baseline. We have to underline a difference of 0.041 with the public score due to overfitting; we were aware of this drawback and we worked for limiting it. We can be quite satisfied from this result as well, in fact without the work on the *mean* features the difference would have become 0.08, almost the double of what we obtained[5]. Probably, we could have achieved even better results in limiting the overfitting with further work on the other statistical features that we computed, *standard deviation, skew* and *kurtosis*, which however is an hard task since these features are measures of variability.

The dataset is composed by many categorical features, and most of them have an high cardinality, so our approach was effective because we extracted relevant features in a quite small number of columns, which in this case means a very fast regression model. Like the majority of the problems, the key is in the preprocessing steps; since this dataset had no numerical features apart from the target variable, it was very important to extract a big amount of information to encode. For this reason, despite our solution provides good results, more complex encoding techniques could be tried. In addition, there is always room for improvement, so the textual data could be analyzed in a deeper manner with language processing libraries such as *natural language toolkit* library [6].

## References


[1] J. Brownlee, *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery, 2020.

[2] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020.

[3] M. Larionov, *Target Encoding and Bayesian Target Encoding*. 2020.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] M. Halford, *Target Encoding done the right way*. 2018.

[6] nltk library, *https://www.nltk.org/*.


[5]It has been tried without smoothing the mean, obtaining a coefficient of determination of 0.9 in local and of 0.82 in public leaderboard