# Regression Pipeline for Wine Quality Prediction

Luca Campana
*Politecnico di Torino*
Student id: s290085
s290085@studenti.polito.it

***Abstract*—The presented project aims to produce a ML algorithm that could predict, as accurately as possible, the quality score of any previously-unseen wine, basing on one of its reviews. This is concretely done by building an efficient regression model: to improve its quality, it has been useful to explore and preprocess the available training data, other than setting proper values for the model's hyper-parameters. The proposed pipeline achieves satisfactory results regarding accuracy and robustness, both in the development set and in the evaluation set.**

## I. Problem overview

The proposed task consists in a regression problem. The dataset used for this project is a *record-type* set, composed of textual wine reviews, and it is divided in two parts:

- a *development set*, containing 120,744 labeled rows, corresponding to as many wine reviews;
- an *evaluation set*, containing 30,186 unlabeled rows, corresponding to the reviews to perform regression on.

To better understand the data domain, it is useful to show a point from the former dataset as an example:

| | |
|---|---|
| **ID** | 33103 |
| **Country** | US |
| **Description** | Savory on the nose with modest apple and pear aromas accented by a curious smoked ham note [...] |
| **Designation** | Semi-Dry |
| **Province** | New York |
| **Region_1** | Finger Lakes |
| **Region_2** | Finger Lakes |
| **Variety** | Riesling |
| **Winery** | Bellangelo |
| **Quality** | 40.0 |

TABLE I: A development set sample.

The first fundamental fact to notice is that, except the Index needed to identify each sample and the related quality score, every single field is textual. Since none of these allow to identify any sort of ranking among data, we can safely state they are all categorical attributes. This means that we cannot perform any form of aggregation or comparison among data, since none of the attributes has order, addition or multiplication properties.

The second analysis to be done over the development set regards the distribution of the quality scores: we know they are all numerical integers, but we are interested in exploring their statistical frequencies, as it is shown in Figure 1.

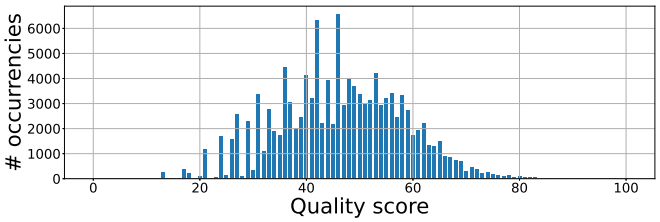


Fig. 1: Quality score distribution over the whole dev. set

From a visual inspection, it is clear that the distribution of quality scores in the dataset is nearly normal, with the exception of a few variations, particularly in the left-side of the graph. Even if this is not a mandatory characteristic [1], it is still a good starting point for our algorithm development.

An additional exploration to be done over the starting dataset regards the amount of unique and missing values that are present in each of the data attributes. This will help us in the choice of the proper approach to every single field, in order to gain a better overall algorithm quality.

| | **# missing** | **# unique** |
|---|---|---|
| **Country** | 5 | 48 |
| **Description** | 0 | 85,005 |
| **Designation** | 35,518 | 27,800 |
| **Province** | 5 | 444 |
| **Region_1** | 20,008 | 1,206 |
| **Region_2** | 72,008 | 18 |
| **Variety** | 0 | 603 |
| **Winery** | 0 | 14,105 |

TABLE II: Missing and unique values, for each attribute.

As it can be seen from Table II, the only fields that have a relevant number of missing values are *Designation*, *Region_1* and *Region_2*. Unfortunately, since we are dealing with categorical attributes, it is not trivial to figure out an efficient way to fill them. Certainly, it is not a good choice to simply drop rows with missing values, since the dataset would lose nearly all of its points. Furthermore, the fact that unique descriptions are largely lower than the total number of rows

in the dataset is a sign that there could be some duplicated points.

One last useful examination is a correlation analysis: we are interested in highlighting fields whose information is completely contained in another (more complex) one. This follows from the fact that we can intuitively perceive a kind of hierarchy among attributes, especially for the geographical ones. The result of this analysis can be stored as a correlation matrix made of binary values, and it can be easily visualized using a heatmap (Figure 2).

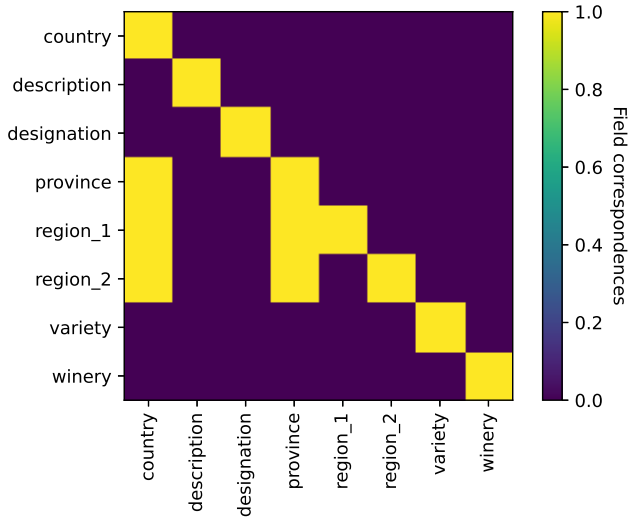


Fig. 2: Field comprehension matrix visualization: all the entries that share *Province*, *Region_1* or *Region_2* also share the *Country* value; all the entries that share *Region_1* or *Region_2* also share the *Province* value.

The most important result of this exploration is that all the information contained in *Country* is fully enclosed in *Province*, so we can safely state there is no reason to take *Country* into account unless we not consider *Province*. Unfortunately, a similar logic cannot be applied to fields *Region_1* and *Region_2*, too, since they include a considerably higher percentage of missing values.

## II. Proposed approach

Before starting to report the concrete steps followed for both preprocessing and parameters tuning, it is necessary to make a premise: all the described considerations and operations over data are made by testing the related evolution of the model behavior, and decisions are driven by the quality trend, evaluated in terms of $R^2$ *score*, from one step to another.

This is concretely done by using an 80/20 *train/test split* on the development set: both parts get preprocessed together, then the model is trained with the former and evaluated with the latter.

### A. Preprocessing

The first process suggested by the previous exploration is to eliminate duplicate points in the dataset: this reduces the development set to 85,025 rows, making data less redundant and computation faster. As one may notice, there are still 20 duplicated descriptions. From a further exploration, it comes out that sometimes two identical descriptions refer to two really similar wines, and led to the same scores; other times, instead, they refer to completely different wines and scores: this means it is not a good idea to consider this field alone for the algorithm development, but it still could contain some useful information.

We know that all data, to be submitted to whichever ML model, has to be numerical: therefore, we need to transform the fields in this sense. This can be done in multiple ways [2], but the simplest and most efficient in terms of running time are the following:

- *One-hot encoding*: used for low cardinality fields;
- *Bag-of-word representation* with *binary weighting*: used for high cardinality document-type fields, such as *Description*.

By the nature of these methods, we know that every point will be turned into an array with values 0 and 1: the overall transformations will generate a really sparse matrix, that can be handled in an agile and highly-computable way by using the Python built-in *scipy.sparse* package. Working with a very sparse matrix causes *course of dimensionality*, so it will require a very careful handling in order not to produce degenerations. On the other hand, this choice has also good advantages: every produced column will likely have mostly 0 values, with only limited exceptions. This implies we will not need to perform standardization or normalization over data, since their output would be very similar to what we already have.

To decrease the possibility to consider highly correlated fields, it is useful to follow a *greedy* strategy for feature subset selection. Starting from limited simple fields, we gradually add further attributes to our model, and make considerations about their additive utility basing on the model variations, both in quality score and in overall complexity.

The first fields to be modeled with OHE are *Province* and *Variety*, for the simple fact that they have low cardinality and limited missing values. Then we try to add *Description*, modeled through *tokenization* in words, elimination of English *stopwords* and *stemming*: the resulting model has better quality score, but the transformed matrix gains more than 10,000 columns. The next attributes to be considered are *Region_1* (through OHE) and *Designation*. From a further inspection of this particular field and its cardinality, it is clear that the smartest way to handle it is through BoW, this time without stemming, but eliminating all words that appear only in one single row. In fact, this field's composition suggests there could be, in the same value, some parts that contain more information than others, implying that in this case BoW is the most well suited representation. As an additional advantage, the result of this process is a sparse matrix with less than 10,000 columns, less than half of what we would have if we had been treating it with OHE. Similar considerations can be made for the field named *Winery*, but in this case word

vectorization does not work properly: instead, one possible approach is to filter out all mono-occurrences and then run OHE. By this way, we obtain a better overall score and avoid nearly 5,000 unneeded columns. The last field to be addressed is *Region_2*: its inclusion worsens the model score, so it will not be treated.

At this point, we try to eliminate some of the analyzed fields to speed up the algorithm: surprisingly, leaving out the field *Description* not always decreases the model complexity, but it also enhances the overall quality. Most likely, the remaining fields already hold all the information contained in this one, thus it becomes totally unnecessary.

From a further inspection it comes out that, only for the *Variety* field, filtering out values that occur in less than a given number of rows maintains steady the score: therefore, this threshold will be one of the hyperparameters to tune.

One last possibility to report is to use the same transformations over the full development set, and not only over non-duplicated data. By this way, as it was predictable, the score increases: this can be a result of the fact of having identical rows both in the train and in the test set. On the other hand, the fact of having duplicates could also help the algorithm to better understand the data context and generalize: so, we can maintain parallelly the two approaches. As it is logical, the model complexity is not affected at all, except for what derives from *Variety* threshold value.

### B. Model selection

For the entire project development, the following models have been used:

- *RidgeCV*: this model is particularly useful in handling high-dimensional data [3], and automatically performs both regularization and variable selection. This avoids *overfitting* and decreases the overall algorithm complexity. It is also particularly efficient both in the model building and in the evaluation phase, and for these reasons it has been used for all the preprocessing stage. Since it has an embedded *Cross-Validation* function (which, in this case, has been performed through *K-fold* with K = 5), the scores returned by this model are robust against noise and outliers. RidgeCV has been only used to compare different parameter configurations, so it did not need any particular hyperparameters tuning.
- *MLPRegressor*: this model (a traditional FFNN for regressions) is the best choice for accuracy, and it is particularly successful in modeling non-linear relationships [4], other than handling sparse data. In addition, if well tuned and trained, it is also able to perform an automatic further data transformation step, in order to learn by itself the best-working feature representation that will boost model performances. For these reasons, tuning, validating and using the MLPRegressor to produce the final prediction over unseen data has allowed to gain some additional $R^2$ points.

The main drawback of MLPRegressor is that it needs a complex hyperparameters tuning in order to get the best-working configuration. This is obtained through a grid search, as it is described in the following paragraph.

### C. Hyperparameters tuning

As outlined in the previous section, the hyperparameters that need to be tuned for the project purposes are:

- the *Variety filtering threshold*, used for preprocessing (hereinafter called *th*);
- MLPRegressor parameters.

By assuming orthogonality for the two sets of hyperparameters, we can tune them separately through as many grid searches, summarized by Table III.

| **Model** | **Parameter** | **Values** |
|---|---|---|
| Preprocessing | *th* | 0→49 |
| MLPRegressor | *hidden_layers* | {(10,), (10,100), (8,64,256), (32,64,128,256,256,256), (64,128,128,256,256,512,512,1024)} |
| | *random_state* | {42} |
| | *early_stopping* | {True} |
| | *activation* | {'relu', 'tanh'} |
| | *learning_rate* | {'constant', 'adaptive'} |
| | *max_iter* | {500} |
| | *validation_fraction* | {0.05} |

TABLE III: Hyperparameters tuning.

As it is logical, the more complex parameter to tune is the number (and composition) of hidden layers for the FFNN. Here, the grid search focuses on understanding how the model behavior changes among really different network structures: therefore, the values for this particular parameter have been chosen randomly.

Both searches have been made on the full set and on the reduced set, following the parallel analysis described in the first paragraph.

## III. Results

The tuning outcome for the *th* parameter is summarized in Figure 3: the best threshold value for the reduced set is equal to 18, while for the full set it reduces to 2.

After fixing this couple of values, the two grid searches for the regressor have been computed. Both of them returned the same network configuration, consisting of: {*hidden_layers* = (64,128,128,256,256,512,512,1024), *random_state* = 42, *early_stopping* = True, *activation* = 'relu', *learning_rate* = 'constant', *max_iter* = 500, *validation_fraction* = 0.05}. The Validation $R^2$ scores here obtained are equal to 0.754 for the reduced set, and to 0.850 for the full one.

Using these particular settings, two FFNN have been built and trained on all available data, respectively for reduced and full development set.

Then, next step consisted in preprocessing correctly the evaluation set, such that the models could accept it as a submission and produce coherent predictions. To avoid bad

model behaviors, it has been chosen not to handle this set until the very last stage of the project, so all the related processing has to be done referring to the columns obtained in the preprocessing of the labeled set. By a practical point of view, both for OHE and for BoW representations, unlabeled data have been processed by the means of predetermined vocabularies, obtained from the previous transformations of development set. This approach also has a logical motivation: for similar tasks, the model is generally supposed to be built in advance, when unlabeled data is not available yet. One of regression goals, in fact, is to build a model that is the most robust possible in handling previously-unseen data.

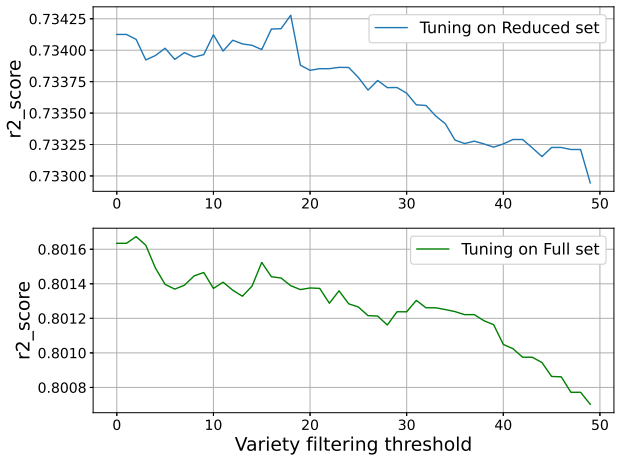


Fig. 3: *th* tuning, both for full and reduced development set.

The obtained evaluation matrices have then been submitted to the two full-trained models. The produced predictions have been evaluated with a Public $R^2$ Score equal to 0.865 for the full model, and to 0.848 for the reduced one: we can fairly assume that duplicate rows enhance model quality and its ability to generalize. However, both scores highly outperforms the *naïve* baseline, which is equal to 0.436, and do not show signs of possible overfitting.

For a brief comparison, we built a random guesser, that assigns each record a quality score, following a normal distribution with $\mu = 50$ and $\sigma = 10$, which is similar to the one observed in Figure 1. The output of this model gets a Public Score of -0.775.

## IV. Discussion

The described approach is fairly satisfying, as it outperforms both baseline and random guess, other than a significant number of scores in the leaderboard. Moreover, the fact of having chosen to build a FFNN grants extremely fast predictions.

However, the score suggests that there could be some further improvement to be made:

- It could be introduced a better approach to treat missing values, and to detect outliers. Submitting more coherent and neat datasets can undoubtedly help regressors to produce more accurate results. However, the fact that we only treat categorical variables does not help identifying a valid approach to any of these issues.
- More sophisticated weighting schemes, such as *tf-df* or *tf-idf*, could be used to preprocess the *Designation* field. Adding terms to measure different in-document and overall frequencies could overcomplicate and slow down the model, but it could also provide some additive information that could help it in understanding the data domain.
- The grid process for MLPRegressor could be deepened, particularly for the parameter regarding hidden layers' composition. Indeed, the fact that model quality had been improving as we tested more complex structures is a clear indicator that, by adding further layers and neurons, we could keep improving the overall algorithm score. So, running a grid search with a wider set of possible configurations will likely enhance the model asset, in terms of both accuracy and $R^2$ Score.

To conclude, although there is definitely space for refinements and enhancements, we can be satisfied with the performances showed by this particularly described type of approach.

## References


[1] R. H. Myers, *Classical and modern regression with applications*, vol. 2. Duxbury press Belmont, CA, 1990.

[2] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, pp. 1–41, 2020.

[3] A. M. E. Saleh, M. Arashi, and B. G. Kibria, *Theory of ridge regression estimation with applications*, vol. 285. John Wiley & Sons, 2019.

[4] D. F. Specht *et al.*, "A general regression neural network," *IEEE transactions on neural networks*, vol. 2, no. 6, pp. 568–576, 1991.