Prediction of quality score in the *Wine Reviews* dataset

Marco Saponara
Politecnico di Torino
Student id: s277323
s277323@studenti.polito.it

***Abstract*—In this report we propose a pipeline to assign a quality score for a specific wine review inside the *Wine Reviews* dataset. In particular, after having managed several missing values, we perform feature extraction through tf-idf and one-hot encoding. Then, these numerical features are passed as input for the hyperparameter tuning of two regressive models, based on random forest and SVM. The proposed approach achieves a high score in terms of R^2, outperforming the competition baseline defined for the problem.**

I. Problem overview

The project consists in the prediction of a quality score associated with a wine review. The challenge, then, is to build a regression model capable of inferring correctly the wine's quality given the content of a specific review.
The *Wine Reviews* dataset for this competition is available at the following link. The records are collected in tabular format and contain geographical information (i.e. *country*, *province*, *region_1*, *region_2*), as well as more specific ones (i.e. *description*, *designation*, *winery*), beyond of course the column *quality*, which is the only numerical variable. Moreover, the 150.930 entries are divided into two parts:

1) a *development set*, *dev.tsv*, containing 120.744 entries where the quality is available;
2) an *evaluation set*, *eval.tsv*, containing 30.186 entries with unknown quality.

Our model will be trained and validated through the former and tested through the latter.
Let us focus on the development set. The first thing to notice is the presence of several duplicates: we cannot know whether they have an actual purpose, such as increasing the importance of some records, or they are the result from an erroneous aggregation between similar datasets. Due to this uncertainty, we decide to be as much conservative as possible by assuming the first hypothesis. Secondly, from Table I it is clear that our data are affected by a prohibitive dimensionality and a large number of missing values: in fact, the preprocessing step will be mainly focused on these two problems. Finally, it is important to observe that, despite the irregular behavior of the other variables, *quality* does not have any missing value and it is well approximated by a Gaussian distribution with parameters $\mu \approx 46$ and $\sigma \approx 12$, as we can see in Figure 1.

column name	# distinct attributes	# missing values	% missing values
country	48	5	0.004
description	85002	0	0
designation	27626	36518	30.24
province	444	5	0.004
region_1	1206	20008	16.57
region_2	18	72008	59.64
variety	603	0	0
winery	14101	0	0
quality	86	0	0

TABLE I: three useful statistics regarding the columns in the development set

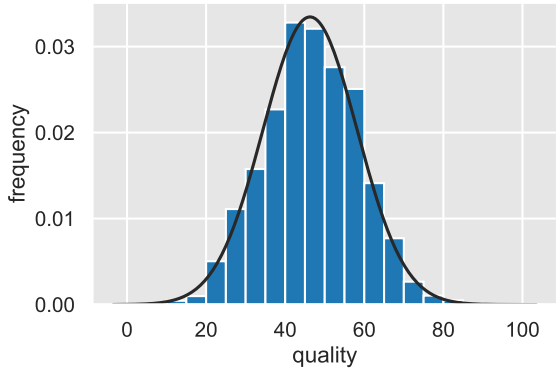


Fig. 1: distribution of the quality score in the development set

II. Proposed approach

A. Preprocessing

1) Missing values: let us start the analysis from column *region_2*: this is the variable with the highest amount of missing data. Fortunately, only a negligible percentage of them is an actual lack of information, because non-null values of *region_2* only occur when *country* is equal to *United States*, so it is reasonable to assume that this variable is not defined for different countries. These missing values are by consequence replaced by an empty string.
Regarding columns *country* and *province*, there are only five co-occurring missing values. We decide to remove them, as they do not represent a significant sample of the dataset.
The challenging part of this step involves the variables *designation* and *region_1*. In order to find suitable candidates

for the replacement, we adopt the same strategy for both columns: the dataset is aggregated at different granularity levels, starting from the most specific to the most general; for each aggregation, we examine the content of each group so that every missing value in the column under analysis belonging to that group is replaced by the most frequent non-null attribute. The worst case scenario happens when a group contains only null values. On the opposite side, if a group does have non-null values and they all share the same attribute, the replacement is most likely correct. In the intermediate cases, we rely on a majority vote based on frequency and few uncertainty is introduced by consequence, but the obtained results are completely reasonable.

The proposed approach achieves a remarkable result when applied to the *designation* column, where it is unable to replace only 91 and 28 missing values respectively in the development and in the evaluation sets. Results get poorer with column *region_1*, where this strategy does not manage to replace 19.223 and 4.874 missing values. This may be due to the fact that the available data does not provide enough information about *region_1*, regardless of the granularity of the aggregation.

To conclude, we remove the rows of the development set associated to the remaining missing designations and we fill the remaining ones in *region_1* with empty strings, as we did with *region_2*.

2) Feature extraction: this is a crucial step to achieve our task and several approaches, equally valid, can be chosen. We propose two alternatives.

Since each record is represented by a set of arbitrarily long and extremely diversified strings (they may vary from one word, e.g. *country='Italy'*, to a full meaning text, e.g. *description='The complexity starts with dusty mineral tones that slowly segue into perfectly mature blackberry, balsam tones and spicy oak notes at the end. It is that amazing evolution of aromas, which reads like a storybook of Tuscany, that makes this wine a standout.'*), a bag-of-words model should be the most suitable representation for our dataset.

Let us now introduce the most naive approach. First, for each record, all the attributes are merged in one string in order to obtain a corpus of text documents containing all the information inside the dataset. Then, we apply the tf-idf weighting scheme, after the standard procedures of case normalization, text tokenization and stop-words removal (performed thanks to the *nltk* library available in Python). Note that we avoid stemming mainly because the majority of the words inside the corpus represents proper nouns and they should not be stemmed; moreover, stemming is strongly language-dependant and our dataset, despite being mostly in English, also contains other languages. Tf-idf provides a sparse representation of the dataset, whose dimensionality is mainly determined by the so-called cut-off value, as it is shown in Figure 2.

This approach has several advantages, for instance each feature ranges between 0 and 1 and the dimensionality is easily manageable, but it also has some drawbacks: the significance of a specific word with respect to our target, i.e. the quality score, may not be related to its frequency inside the textual corpus, so that we may lose relevant information just because the frequency of some specific words are below the cut-off threshold; moreover, the tokenization step may split words which are much more effective when considered as a whole, and this is particularly evident with geographical names.

It is possible to partially overcome these issues with the introduction of one-hot encoding inside the model. In particular, the second proposed approach consists in applying OHE to the columns with a relatively low number of distinct attributes, while aggregating the other ones into a text corpus and processing it with tf-idf. For this specific situation, we consider more appropriate to perform one-hot encoding on columns *country*, *province* and *variety*, reaching approximately 10^3 binary features, and to complement them with tf-idf performed on a corpus composed by the aggregation of columns *description*, *designation*, *region_1*, *region_2* and *winery*.

This operation clearly leads to the partial solution of the problems listed above, but it necessarily increases the dimensionality and the sparsity of our dataset. As we know, the majority of machine learning algorithms may be severely affected by the *curse of dimensionality* and by the related *overfitting* process, but these topics will be covered in the next sections.

B. Model selection

The following algorithms have been tested:

1) random forest regression: it is an ensemble learning method that operates by constructing a multitude of decision trees at training time and providing as output the average prediction of the individual trees. This typically avoids the overfitting problems of decision trees but still maintaining some degree of interpretability.

Random forests usually manage to capture more complex and non-linear dependencies than an ordinary linear regression and this is the reason why we consider this model to be more suitable for this situation.

2) support vector regression [1] : as a supervised-learning approach, SVR trains using a symmetrical loss function, which equally penalizes high and low misestimates. Using Vapnik's ϵ-insensitive approach, a flexible tube of minimal radius is

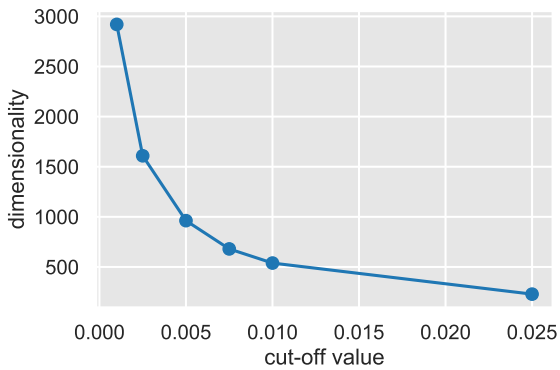


Fig. 2: number of features w.r.t. the *min_df* parameter

formed symmetrically around the estimated function, such that the absolute values of errors less than a certain threshold ϵ are ignored both above and below the estimate. In this manner, points outside the tube are penalized, but those within the tube, either above or below the function, receive no penalty. One of the main advantages of SVR is that its computational complexity does not depend on the dimensionality of the input space. Additionally, it has excellent generalization capability, with high prediction accuracy.
Due to computational restrictions, we choose a *linear* kernel.

C. Hyperparameters tuning

The hyperparameters to be tuned can be divided in two groups:

- the cut-off value (*min_df* in Python) in the preprocessing step;
- the hyperparameters involved in the random forest and SVR algorithms.

In order to choose the most suitable value for the former, we apply a grid search on *min_df* through a 70/30 hold-out validation on the development set, then training a random forest and an SVR with their default configurations and assessing their performance based on the resulting R^2. This procedure is obviously performed both for the two proposed alternatives in the feature extraction step.
After completing this step, we can run a grid search with the above validation strategy on both SVR and random forest, based on the hyperparameters defined in Table II.

III. Results

The tuning of parameter *min_df* is summarized in Figure 3: these plots share similar trends, even if the two models achieve overall higher scores with the introduction of one-hot encoding during the feature extraction step. It is clear that the lower the value of *min_df*, the higher the R^2 score, with the performance of SVR being much more affected by this choice than random forest's one.
Due to computational restrictions, we set *min_df* to 0.001 (i.e. words with a document frequency strictly lower than 0.1% are ignored), but results would probably improve by decreasing this threshold even more.
Regardless of the feature extraction step, the best configuration for SVR corresponds to $\{\epsilon$: 1, C: 10$\}$, whereas the random forest performs best with the configuration $\{$*n_estimators*: 200, *min_samples_split*: 2, *max_features*: "sqrt", *max_depth*: None$\}$. The results achieved during the test phase of the fine-tuned models trained on the entire development set are summarized in Table III. As we can see, the best public R^2 scores

Model	Hyperparameters	values
random forest	*max_depth*, *min_samples_split*, *n_estimators*, *max_features*	{50, 80, None}, {2, 5}, {100, 200}, {"sqrt", "log2"}
SVR	C ϵ	{0.1, 1, 10, 50}, {0.001, 0.01, 0.1, 1, 10}

TABLE II: hyperparameters considered for our models

	preprocessing	
	tf-idf	OHE & tf-idf
SVR	0.615	0.633
random forest	0.772	0.771

TABLE III: R^2 scores obtained during the test phase on the evaluation set

obtained are slightly higher than 0.77 and are associated to the random forest models. SVR, instead, has poorer performances, achieving a public R^2 score approximately equal to 0.62-0.63, nevertheless it is remarkably higher than the baseline defined for this competition (0.436). This may be due to the fact that the dependencies between the quality score and the regressors may be highly non-linear, so that a radial function basis (*RBF*) kernel may be preferred. Unfortunately, the fit time complexity is more than quadratic with the number of samples which makes it hard to scale to datasets with more than 10^4 samples [2].
Note that, since these scores are obtained using the evaluation data, there should be no overfitting and we can reasonably assume that similar results can be obtained on the private score.

IV. Discussion

The proposed pipeline, through the simple means of data analysis and machine learning, is able to reach competitive results. It is clear that there are scope for improvement:

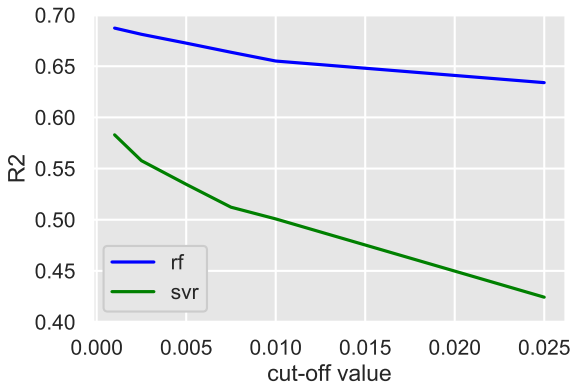
- running a more thorough grid search process, both for random forest and SVR. This process surely allows to achieve better results, but it necessarily requires a higher computational time. For instance, we tested an SVR model with configuration $\{$*kernel*: "rbf", ϵ: 1, C: 10$\}$ trained on the features extracted through tf-idf with *min_df* parameter equal to 0.0001: we reached our best score on the public leaderboard (0.772) but the whole process lasted approximately 10 hours;
- reducing the dimensionality and the sparsity of the dataset through the means of PCA or SVD;
- using a more complex model, such as a neural network (e.g. *MLP Regressor*). Nevertheless we remark the fact that it is not trivial to manage a deep learning model and we avoided this approach precisely because it was out of the scope of this project.

To conclude, we strongly believe that the R^2 achieved by our best-performing model is remarkable and satisfactory, if we consider the restricted amount of available time and the limited computational capabilities of a laptop.

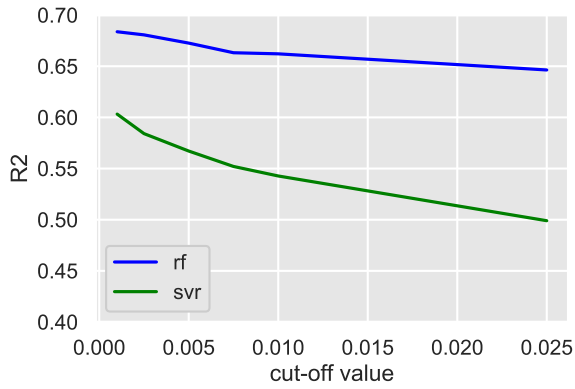
References

[1] M. Awad and R. Khanna, *Efficient Learning Machines*. Apress, Berkeley, CA., 2015.

[2] 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.



(a) preprocessing consisting of tf-idf



(b) preprocessing consisting of tf-idf and one-hot encoding

Fig. 3: R^2 value obtained with default models w.r.t. the *min_df* parameter