Data Science Lab
Project Assignment Winter 2021 - First Call

Shannon Mc Mahon
Politecnico di Torino
Student id: s289958
s289958@studenti.polito.it

***Abstract*—In this report we introduce a possible approach to predicting the quality score associated to a wine, based on the content of its review. The proposed solution infers the wine's quality by means of a regression model, that gives overall satisfactory results. For clarifications, please refer to the code provided along with this report.**

I. Problem overview

The proposed project is a regression problem on a data-set of wine reviews. Each review provides the following information: *country*, *description*, *designation*, *province*, *region_1*, *region_2*, *variety* and *winery*.

The data-set is divided into two parts:

- a development set, containing 120744 entries for which, in addition to the previously mentioned features, the *quality* of the wine is also known. The latter is measured by means of a score that ranges between 0 and 100.
- an evaluation set, comprised of 30186 entries. We will use the development set to build a regression model to predict the quality of these wines.

Let us now focus our attention on the development set.
In table I an entry from the development set is shown, so as to have a better idea of the information that each feature provides. Notice how all features are categorical, and will as such need to be converted into a numerical value (more on this in the *Preprocessing* section).

country	US
description	One of the more successful blush wines out there...
designation	Estate Grown
province	California
region_1	Dry Creek Valley
region_2	Sonoma
variety	Rosé
winery	Fritz
quality	37.0

TABLE I
Sample entry from the development set

Once we have a clear idea of how the data-set is structured, we focus our attention on the *quality* feature. One aspect worth analysing is the distribution of the wine quality. As can be seen in Fig. 1, it seems to have a Gaussian trend. A further inspection of the development set shows that the average score for a wine is 46.27, and the *25th, 50th and 75th* percentiles are respectively 38.00, 46.00 and 55.00. As intuition would suggest, it is more difficult to come across wines of awful or excellent quality.

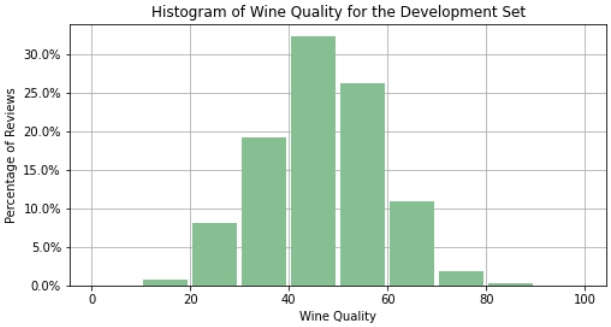

Fig. 1. Distribution of *quality* in the development set

An issue that must be addressed before moving on to the creation of the regression model is the presence of *missing values* and *duplicates* in the data.
Upon analysing the development set we see that 31553 entries out of 120744 have no missing values in any of the features. This means that only 26% of our data is complete. More information on missing values can be found in table II, while the approach used to address this problem is explained in depth in the *Preprocessing* section.

Feature	Missing Values Count	Percentage
country	5	0.004%
description	0	0%
designation	36518	30%
province	5	0.004%
region_1	20008	16.5%
region_2	72008	59%
variety	0	0%
winery	0	0%
quality	0	0%

TABLE II
Analysis of missing values for each feature in the development set

As far as duplicates are concerned, it can be easily verified that these are in fact present in the data. According to the project description provided for this assignment, each entry in

the data-set refers to a review expressed by an expert on a specific wine. We will for this reason assume that the couple *(description, quality)* uniquely identifies a review.

II. Proposed approach

A. Preprocessing

1) Missing Values: Based on the knowledge gathered in table II we can discard the feature *region_2* from our data (both in the development and evaluation set). In fact, not only are more than half of values missing for this column, but it is also possible to verify that whenever *region_1* is missing, *region_2* is also missing, so the two are related.
Furthermore, since there are 5 entries in our development set for which *country*, is not defined, we discard these reviews (one can easily check that these reviews are also the ones for which *province* is not defined). All other missing values in the remaining features are replaced with the string *'null'*.

2) Duplicates: By making use of the definition previously given to uniquely identify a review, every time two or more reviews with identical *description* and *quality* are found in the development set, only the first occurrence is preserved, while all others are removed from the data-set.

3) Description and Designation: To process these two features we will make use of *TfidfVectorizer*, provided by *Scikit-learn* [1]. Recall that the term frequency–inverse document frequency of a term is a numerical value computed separately for each document (in our case description or designation) of the data-set. It is directly proportional to the number of occurrences of the term in the document, and inversely proportional to the number of documents in which the term is present. The *Vectorizer* allows us to remove accents, perform character normalization, specify the language we are considering (English) so as to remove its stopwords, and extract different n-grams (contiguous sequence of n items from a given sample of text). The lower and upper bound for the range of the n-grams were set to 1 and 3 respectively. Lastly, when building the vocabulary terms that have a document frequency strictly lower than a given threshold *L*, and terms that are present in a proportion of documents that is greater than a given threshold *U* will be ignored. In this solution we set $L = 2$ and $U = 0.9$.
We can then, separately for the *description* and *designation* features, learn the vocabulary and inverse document frequency on the development set, and return the document-term matrix. Once this task is complete, it is possible to transform the documents of the evaluation set to document-term matrix as well.

4) Winery, variety, province and region_1: These features are processed by applying *one hot encoding* (*Pandas* [2] supports this feature with *get_dummies*). However, before encoding these categorical attributes, the values assumed by the evaluation set are taken into consideration. More specifically, for each of these features if we find a value that is *not* also present in the development set we discard it, as our model will be trained on the development set and for this reason will not be able to gather information from values that are not in it. Subsequently, we can concatenate the development and evaluation sets, and apply *one hot encoding*. Lastly, after having re-separated the development and evaluation set (which can be clearly distinguished since only the former has the feature quality defined), *SciPy*'s [3] *csr_matrix* is applied to the obtained results so as to have a compressed sparse matrix. The advantage of this format is it renders operations on our the matrices faster and more efficient.
Having terminated the preprocessing phase, the next step consists in defining the model that will be implemented to perform the regression task on the development set.

B. Model selection

Four of *Scikit-learn*'s regressors will be used. A brief explanation of these and their main parameters will be given, for more information please refer to the official *Scikit-learn* documentation.

- *LinearRegression*
  Fits a linear model to minimize the residual sum of squares between the observed targets in the dataset and the ones predicted by means of the linear approximation. Among the various input parameters we have:
  - *fit_intercept*: boolean, specifies if the intercept for the model should be calculated. If set to false, then the data should already be centered.
  - *normalize*: boolean, specifies whether to perform normalization.
- *LinearSVR*
  Linear Support Vector Regression. The following input parameters will be analysed:
  - *C*: Regularization parameter which is inversely proportional to the strength of the regularization. Must be positive;
  - *loss*: Defines the loss function (e.g. L1 loss, L2 loss);
  - *fit_intercept*: boolean, specifies whether to calculate the intercept for the model;
  - *dual*: boolean, specifies which optimization problem to solve (primal or dual).
- *KNeighborsRegressor*
  Regression based on k-nearest neighbors in the training set. As input parameters we will consider:
  - *n_neighbors*: Number of neighbors to use;
  - *weights*: weight function applied for the prediction (uniform, distance);
  - *algorithm*: Algorithm used to compute the nearest neighbors (ball tree, kd tree, brute). When dealing with sparse input it is not possible to use trees. In these cases the setting of the parameter is overridden, and brute force is applied;
- *VotingRegressor*
  Fits several base regressors on the dataset and averages the individual predictions to form a final one. This

regressor will, for the purpose of our solution, take as input the three previously defined.

C. Hyperparameters tuning

Hyperparameter tuning is a vital step in finding the best configuration of the parameters of our regressors.
First of all, the development set is divided into two parts: a training set, used to train our model, and a validation set, that is necessary to evaluate the performance of the model. The latter will be determined by calculating the coefficient of determination R^2 on the wine quality feature (which is what we wish to predict).
The split of the data we wish to obtain can be achieved by means of *Scikit-learn*'s *train_test_split*,

$$x\_train, x\_val, y\_train, y\_val = train\_test\_split(x, y)$$

where x is the development set deprived of the data of the *quality* feature, which is instead stored in y. We shall use *x_train* and *y_train* to train the model, and *x_val* and *y_val* to evaluate the regressor's performance.
Once we've split the development set, we proceed as follows:

1) Fit *LinearRegression, LinearSVR, KNeighborsRegressor* regressors to the training data *(x_train, y_train)* using the *default* parameter configuration;
2) Make a prediction *y_pred* for each model using the validation data *x_val* ;
3) Calculate the R^2 score associated to each regressor by comparing the expected wine quality *y_val* with the prediction obtained in the previous step *y_pred*.

These steps allows us to establish a *baseline* R^2 score for each regressor. The goal is now to improve these scores by means of *Hyperparameter tuning*.

Parameter	Possible values	Optimal value
fit_intercept	[True, False]	True
normalize	[True, False]	False

TABLE III
GRIDSEARCHCV FOR LINEARREGRESSION

Parameter	Possible values	Optimal value
C	[1, 10, 100, 1000]	1
loss	['L1','L2']	'L2'
fit_intercept	[True, False]	True
dual	[True, False]	False

TABLE IV
GRIDSEARCHCV FOR LINEARSVR

Parameter	Possible values	Optimal value
n_neighbors	[3,4,5]	5
weights	['uniform','distance']	'distance'
algorithm	['ball_tree', 'kd_tree', 'brute']	'brute'

TABLE V
GRIDSEARCHCV FOR KNEIGHBORSREGRESSOR

For this purpose *Scikit-learn*'s *GridSearchCV* is implemented. It performs a cross-validated grid-search over a parameter grid that must be defined for each regressor, and applies a *fit and score* method, so that once the search is finished we have access to the best configuration for our parameters. The grid search uses 3-fold cross validation on the training set, and once the optimal hyperparameters are found they are tested on the validation set. Tables III, IV, V show the grids implemented for the regressors, and the optimal configuration found. Our final regressor, which will be used to make predictions on the evaluation set, is obtained by providing the *VotingRegressor* with the optimal versions of each of the three regressor (these are obtained by considering the optimal hyperparameter configurations previously determined).

III. RESULTS

In table VI we can see how using the optimal configurations found with the grid search allowed us to improve the R^2 score (we do not see an improvement in LinearRegression for which the default configuration coincides with the optimal one). In particular, the *VotingRegressor* yields an R^2 score of 0.773.

Regressor	Baseline R^2	Optimal R^2
LinearRegression	0.732	0.732
LinearSVR	0.723	0.764
KneighborsRegressor	0.671	0.685
VotingRegressor	-	0.773

TABLE VI
R^2 OF BASELINE REGRESSOR VS OPTIMAL REGRESSOR

IV. DISCUSSION

The final R^2 score obtained is promising. In fact, the closer the value is to 1, the better the model is at making predictions. It is now possible to fit the created model to the whole development set, and then predict the wine quality for the reviews of the evaluation set (the obtained values are stored in a CSV file).
It is reasonable to assume that the data was distributed uniformly between the development and evaluation collections. It follows that, if the regression model performs well, the distribution of wine quality obtained for the evaluation set should resemble that of the development set.

Fig. 2. Distribution of *quality* in the evaluation set

If we compare Fig. 2 with Fig. 1 we see such similarity. The goodness of our model is further confirmed by the submission of the CSV file, for which the R^2 score on the public leaderboard is 0.892. One may thus infer that the model generalizes well.

REFERENCES

[1] 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.

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

[3] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.