Beer Quality Regression Problem

Salvatore Junior Curello
Politecnico di Torino
Student id: s268066
s268066@studenti.polito.it

***Abstract*—In this report we introduce a possible approach to a *Beer Reviews* Dataset regression problem. In particular, the proposed approach consists in extracting significant information from each beer review and combining them with other additional features contained in the dataset, in order to predict the quality score of a specific beer. These predictions were built using different regression algorithms which will be evaluated according the R2 score metric.**

I. Problem overview

The proposed project is a regression problem concerning a *Beer Reviews* Dataset which contains beer reviews in tabular format. It counts 100.000 entries, each of which refers to a review expressed by an user on a website for beer benchmarks. Each review is characterized by both numerical and categorical attributes. A textual description is provided as well. The quality score is indeed reported on the feature named *"review/overall"*. The goal of this project is to build a regression model capable of inferring the beer's quality given the content of the review. The dataset is divided into two parts:

- a development set, contains 70000 samples with their relative quality value
- an evaluation set, comprised of 30000 samples.

We will use the development set to build a regression model to correctly predict the beer's quality in the evaluation set.

We can make some considerations based on the development set. First, the dataset contains 13 features: *"beer/ABV"*, *"beer/name"*, *"beer/style"*, *"review/appearance"*, *"review/aroma"*, *"review/palate"*, *"review/taste"*, *"review/text"*, *"user/ageInSeconds"*, *"user/birthdayRaw"*, *"user/birthdayUnix"*, *"user/gender"* and *"user/profileName"*. According to a preliminary inspection summarized in Table I, some features have a very large number of unique and/or missing values. The presence of missing values is a relevant issue for many regression models, while the high cardinality of the domain for some of the categorical attributes may lead to a very large and sparse matrix.

We can notice that the columns with the highest percentage of missing values are *user/ageInSeconds, user/birthdayRaw, user/birthdayUnix, user/gender* and *user/profileName*, which is a more specific information of the reviewer. In particular, the features *user/ageInSeconds, user/birthdayRaw, user/birthdayUnix* contains the same information so it will be useless to include all of them during the analysis.

The feature *"beer/style"* specify the style of each beer, Figure 1 shows a bar chart representing the top 15 beer styles associated to the highest number of reviews. As we can see "American IPA" is the beer style with the highest number of review associated, this could means that users are likely to review this particular beer style.

TABLE I
UNIQUE AND MISSING VALUES IN DEVELOPMENT SET

Feature	# unique values	# missing values
beer/ABV	336	3107
beer/name	14770	0
beer/style	104	0
review/appearance	9	0
review/aroma	9	0
review/palate	9	0
review/taste	9	0
review/text	69975	18
user/ageInSeconds	1952	55355
user/birthdayRaw	1882	55355
user/birthdayUnix	1882	55355
user/gender	2	41819
user/profileName	10573	14

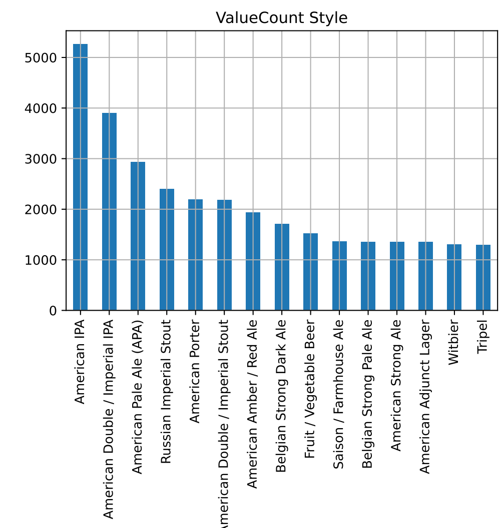

Fig. 1. Top 15 beer styles associated to the highest number of reviews

The feature *"review/text"* contains the text of each review. A review is mainly driven by adjectives that, most of the time, give the sentiment to the statement.

The column *"review/overall"*, instead, is expressed as a number between 1 and 5 (with half scores allowed). In order to explore the data distribution, we used a box plot, shown in Figure 2, implemented with *Seaborn*, in which is possible to summarize the data distribution of a feature and get a nice visualization. At first glance through the box plot, it is possible to observe that the data distribution is mostly in the interval ranged about [3.5, 4.5]. This could be seen also in Figure 3 in which it is represented the distribution of the target value. The remain column *"beer/ABV"* has been considered as numerical value while the columns *"beer/name", "review/appearance", "review/aroma", "review/palate", "review/taste", "beer/style", "user/gender", "user/profileName"* have been considered as categorical so they need a bit of preprocessing before to be included in the regression model. Finally, no duplicates were detected.

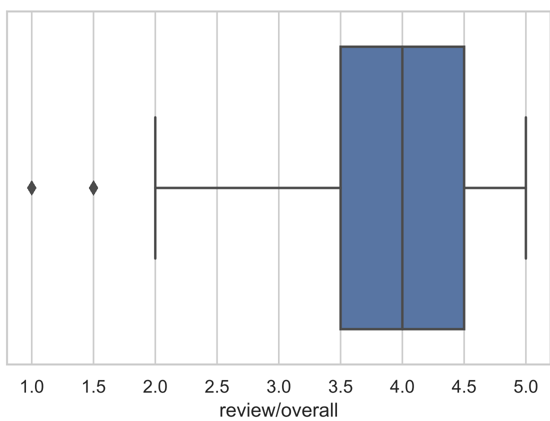

Fig. 2. Box Plot of "review/overall"

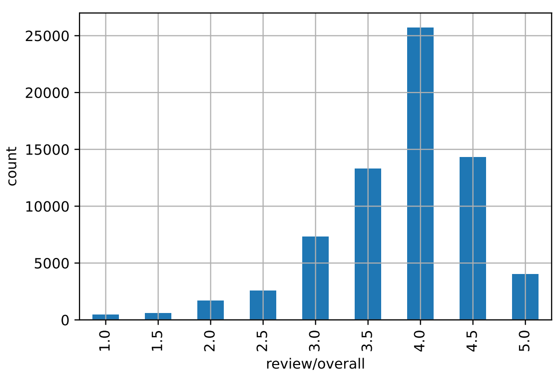

Fig. 3. Distribution of "review/overall"

II. Proposed approach

A. Preprocessing

The first issue to consider concerns the features *user/ageInSeconds, user/birthdayRaw* and *user/birthdayUnix*, all of them give the same information. For this reason, we decided to keep only one of them. We considered to use only the feature *user/ageInSeconds* while *user/birthdayRaw* and *user/birthdayUnix* have been discarded. In order to make this feature more interpretable we converted the seconds in years and we renamed the feature *"user/age"*. Since this dataset contains heterogeneous data types, we need to apply different preprocessing and feature extraction pipelines to different subsets of features. According to that, we created the preprocessing pipeline for numeric, categorical and textual data. To do that we used the classes *Pipeline* and *ColumnTransformer* of *Scikit-learn (sklearn)* [1]. The presence of missing values can affect the regression model. For this reason, the numeric features have been standard-scaled after mean-imputation (which produced a sparse matrix of shape (70000, 2)), while the categorical data is one-hot encoded after imputing missing values with a new category *('missing_namecolumn')*. In particular, the class OneHotEncoder of scikit-learn created a sparse matrix of shape (70000, 25487). We set the parameter *handle_unknown* equal to *'ignore'*, in this way when an unknown category is encountered during transform, the resulting one-hot encoded columns for this feature will be all zeros. The feature *"review/text"* is a string value and it is the actual text of the review. Its missing values have been replaced with *"missing_review/text"*. The preprocessing step, in this case, can be simplified by the use of two libraries: *Natural Language Toolkit(nltk)* [2] and *sklearn*. To do that we used a tokenizer class which make use of two *nltk* functionalities: *word_tokenize* and *WordNetLemmatizer*. In particular, the first one returns a tokenized copy of text, using NLTK's recommended word tokenizer, while the second allow to convert a word to its base form. Then we removed the tokens with punctuation and chars and, finally, we used the *TfidfVectorizer* class from *sklearn*. The parameters that have been tested are: different value of *min_df* and *max_df*, in order to exclude both too frequent and too rare words that would not be relevant from an informative point of view; for the *tokenizer* parameter, since require a callable object, we passed the result of the tokenizer class already mentioned; for the stopwords, instead, we used the *nltk* already-available function *stopwords*. We also tested different values for the parameter *ngram_range*, in this way not only a single word is considered but also a group of *n* words of an adequate dimension. Table II shows the different values assigned to these parameters. All of these procedures conducted on the textual feature did not bring any improvement in term of final score, for this reason, we used the *TfidfVectorizer* class with default parameters removing only the english stopwords. Figure 4 shows a Wordcloud for the top 200 most significant words obtained after the text-preprocessing phase. As expected, since we did not set any value for *max_df*, the most common word is *beer* but we can also see some adjectives that could describe a specific beer (e.g., light, sweet, nice, dark). Once the transformation of the textual feature is concluded we get a sparse matrix of shape (70000, 58963). In summary, by using the *ColumnTransformer* class, our attributes characterized by different data types have been transformed separately and the features generated by each transformer have been concatenated to form a single sparse matrix of shape (70000, 84452) in which 70000 is the number of samples and 84452 is the number of features. Furthermore, we also applied a SVD technique in order to reduce the dimensionality of the matrix but it resulted computationally expensive without improving the results.

Fig. 4. Wordcloud of the 200 most frequent words

B. Model selection

The following algorithms have been tested:

- *Ridge*: it is an extension of linear regression where the loss function is modified to minimize the complexity of the model. This modification is done by adding a penalty parameter that is equivalent to the square of the magnitude of the coefficients.
- *Lasso (Least Absolute Shrinkage and Selection Operator)*: while Ridge tends to lower uniformly all the coefficients (coefficients already close to 0 do not affect the sum of squares), Lasso tends to assign values very close to zero to some coefficients, in the sense that even smaller coefficients affect the sum.
- *Random Forest*: this algorithm is an ensemble of decision trees. The idea is to decorrelate the trees, so first of all a certain amount of bootstrapped sets is generated from the original training set and then on each of this set a tree will trained but when building these decision trees, instead of looking at the entire table of possible splits, we randomly pick, at each split, a fixed number of predictors which is less than the total number. Usually is the square root of the number of the predictors but it can be different. This approach, typically, avoids the overfitting problems of decision trees but still maintaining some degree of interpretability.
- *Gradient Boosted Machines (GBM)*: they are tree-based ensemble models that can be used both for classification and regression problems. Rather than aiming to predict the outcome of the target variable we want to predict, it tries to predict model errors. These errors are estimated from the residuals, i.e. the difference between the actual observed value and the predicted value. These residuals will be used to fit a new model which will be ensembled together with the model in which we obatained the residuals. At each iteration the model aims to estimate the residuals which gradually reduce. After a certain number of iterations the overall error of the model is reduced compared to the beginning. In our case, we used the implementation given from the library *LightGBM*. Its characteristic is that it grows tree vertically while other algorithm grows trees horizontally meaning that it grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm. We used this algorithm because it works well with dataset containing categorical data, it can handle large size of data and takes lower memory to run.

C. Hyperparameters tuning

We used an 75/25 train/test split in the development set and we ran a grid search with cross validation with 5 folds for all the algorithms. The metric used for the comparisons will be the R2 score and the hyperparameters that we used are defined in Table II (we used default value where it is not explicitly specified). In particular, we analysed different value of *alpha* for Ridge and Lasso. For the LightGBM we considered the parameters *num_leaves*, i.e. the max number of leaves in one tree, and *min_child_samples* which is the minimum number of the records a leaf may have.

TABLE II
HYPERPARAMETERS CONSIDERED

Model	Parameter	Value
Preprocessing	*min_df*	{5, 10}
	max_df	{0.2, 0.3}
	ngram_range	{(1,2), (1,3), (1,4)}
Ridge	*alpha*	{0.01, 0.1, 1.0, 10, 17, 18, 20}
Lasso	*alpha*	{1, 0.1, 0.01}
LightGBM	*num_leaves*	{16, 32, 64, 128}
	min_child_samples	{10, 20, 30}
Random Forest	*n_estimators*	{100, 200}
	max_depth	{5, 10}

III. Results

The performances of Lasso Regression are lower than the other models. Its best configuration has been obtained with alpha=0.01 in which we got an R^2 score of 0.5972 on the test set and 0.601 on the public evaluation set. We got more interesting result with the other models. The R^2 scores reported in Table III and IV, show the mean of the five R^2 scores obtained by Ridge and LightGBM respectively (with the configuration among the ones reported in Table II) during the fine tuning process (one score for each fold used in the cross-validation). In particular, with LightGBM (*min_child_samples*= 20 and *n_leaves*=64) we obtained an R^2 score of 0.7071 on the test set and 0.712 on the public evaluation set while with Ridge the score is slightly higher since we reached an R^2 score of 0.7125 on the test set and 0.716 on the public evaluation set, highlighting that a strong regularization increase the final score. For Random Forest we tested a grid search with *n_estimators*:

TABLE III
HYPERPARAMETERS TUNING FOR RIDGE

Model	Alpha	R^2
Ridge	0.01	0.17939
Ridge	0.1	0.52687
Ridge	1	0.66220
Ridge	10	0.70603
Ridge	17	0.70740
Ridge	18	0.70741
Ridge	20	0.70733

TABLE IV
HYPERPARAMETERS TUNING FOR LIGHTGBM

Min_child_samples	Num_leaves	R^2
10	16	0.7025
10	32	0.7049
10	64	0.7041
10	128	0.7021
20	16	0.7028
20	32	0.7052
20	64	0.7054
20	128	0.7025
30	16	0.7032
30	32	0.7053
30	64	0.7047
30	128	0.7026

[100, 200] and *max_depth*: [5, 10]. Its best configuration was $\{$*n_estimators*=200, *max_depth*=10$\}$ in which the R^2 score obtained was 0.6782 on the test set and 0.680 on the public evaluation set. We used the whole development set to train the regression algorithms with their best configurations. Then they have been applied to the evaluation set in order to obtain the predictions.

IV. DISCUSSION

All the models (except Lasso) provide a result which is sufficient to exceed the baseline provided. As already said we obtained best performance with LightGBM and even more with Ridge. Furthermore, the application of the one-hot encoding for the categorical features and the TF-IDF for the textual feature provided us satisfactory results. An interesting fact regards the data type of the features *"review/appearance", "review/aroma", "review/palate", "review/taste"*, as shown in Table I the unique values of these features is 9, since it is not a very high value we tried to make a deeper analysis in which we initially considered them as numerical values and then as categorical ones. Concerning the public evaluation set and considering them as numerical values with LightGBM, the obtained R^2 score is equal to 0.713, increasing the score of just 0.001. Instead, considered them as categorical values helped us to increase the score of Ridge of 0.005 on the public evaluation set, reaching, in this way, our final score of 0.716. The following are some aspects that might be worth considering to further improve the obtained results:

- Other feature extraction approach may be considered, an example could be *Word2Vec* which is one of the ideas of modern statistical NLP, where it can associates words with points in space. Then word meaning and relationships between words are encoded spatially [3].
- Since only a limited set of hyperparameters has been studied, it could be interesting to run further grids search in order to explore new configurations that could perform better, especially for Random Forest in which the hyperparameter tuning phase was, by far, the most computationally expensive step in the entire process. Of course, the increasing of both *max_depth* and *n_estimators* can lead to better performances but, at the same time, it adds complexity. The lower score obtained (related to the Random Forest), compared to Ridge and LightGBM, shows that even if we imposed the maximum depth of the tree to avoid overfitting, we are loosing some information that could be relevant for the predictions. Furthermore, even if Random Forest is less interpretable because predictions are made by hundreds of trees and not by a single tree, it can provide global feature importances which allows us to identify the most important attributes that drive the quality of the predictions.
- It could be used a *MLPRegressor* which trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It is suited for describing complex patterns of relationships among variables than statistical models due to their ability to capture non-linear relationships in data [4].

To conclude, Figure 5 shows a diagram (obtained with *sklearn.set_config(display='diagram')*) which summarize the structure of the pipeline used in order to get the best predictions for the evaluation set.

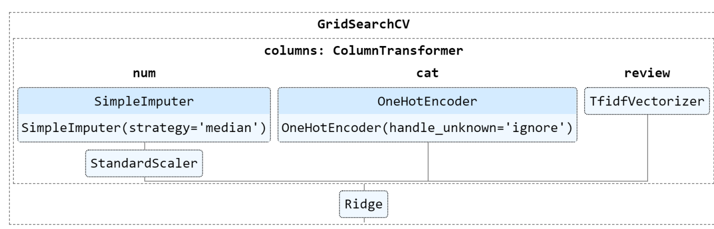

Fig. 5. Structure of the pipeline used for the predictions

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] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[3] J. Polpinij, N. Srikanjanapert, and P. Sopon, "Word2vec approach for sentiment classification relating to hotel reviews," in *Recent Advances in Information and Communication Technology 2017* (P. Meesad, S. Sodsee, and H. Unger, eds.), (Cham), pp. 308–316, Springer International Publishing, 2018.

[4] S. Lee and J. Y. Choeh, "Predicting the helpfulness of online reviews using multilayer perceptron neural networks," *Expert Systems with Applications*, vol. 41, no. 6, pp. 3041 – 3046, 2014.