An Ensemble Approach to Predict Beer Quality Score

Leonardo Maggio
Politecnico di Torino
Student id: s292938
292938@studenti.polito.it

***Abstract*—In this report we propose a possible approach to predict the quality score associated with a beer review. First, we handle missing values and perform feature extraction through *tf-idf* and *one-hot encoding*. Then, different regression models are trained on this processed data and compared. Finally, the best performing models are combined in an ensemble, obtaining a fulfilling result.**

I. Problem overview

The *Beer Reviews* data-set gathers 100,000 reviews expressed by the users of a website for beer benchmarks. The proposed competition consists in building a regression model capable of inferring the overall quality score assigned to a beer by its reviewer, expressed as a number between 1 and 5 (half scores included). The records are collected in tabular format and contain the beer evaluation and the textual description written by the reviewer, as well as several information about the user.

The data-set is divided in two portions:

- a development set, containing 70,000 labelled entries, as the target feature, *review/overall*, is provided;
- an evaluation set, containing 30,000 unlabeled entries.

We will use the former set to train, in a supervised manner, a regression model capable of predicting the overall scores of the latter.

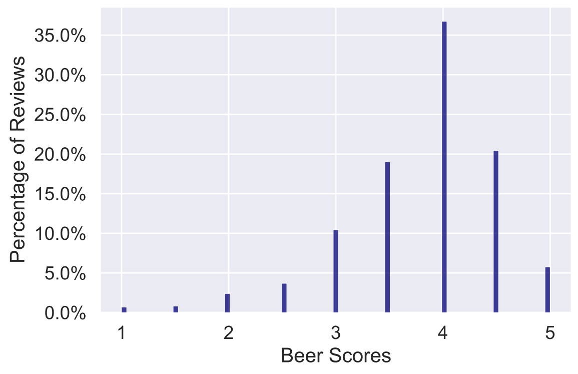
Let us now focus on the development set, of which Table I provides a summary. The high cardinality of some categorical features and the considerable presence of *null values* must be taken into account during the pre-processing phase. In addition, we must understand the content of the *review/text* column, since the full text of the review may contain further information about the beer and criticisms regarding quality, which are highly meaningful to the task.

Before continuing, one more aspect worth analyzing is the distribution of the overall beer scores. In fact, the data-set appears to be quite unbalanced, with a left-skewed distribution as shown in Figure 1. A further inspection of the development set shows that the average score for a beer is 3.82 and the 25th, 50th, and 75th percentiles are respectively 3.5, 4.0, and 4.5. We can assume that the majority of the website users are not Bier-Sommeliers, but ordinary people, who tend to consume beers of their liking and to review them in order to recommend them to others.

TABLE I
Development set summary

Feature	Type	Cardinality	Missing Values
beer/ABV	numerical	336	4,44%
beer/name	categorical	14770	0,00%
beer/style	categorical	104	0,00%
review/appearance	numerical	9	0,00%
review/aroma	numerical	9	0,00%
review/overall	numerical	9	0,00%
review/palate	numerical	9	0,00%
review/taste	numerical	9	0,00%
review/text	textual	69975	2,57%
user/ageInSeconds	numerical	1952	79,08%
user/birthdayRaw	categorical	1882	79,08%
user/birthdayUnix	numerical	1882	79,08%
user/gender	categorical	2	59,74%
user/profileName	categorical	10573	2,00%

Fig. 1. Beer Scores Distribution



II. Proposed approach

A. Preprocessing

1) *Missing values*: Table I shows that the features relating to the age and gender of the user have more than half of missing values. Thus, we decide to discard these columns.
Moreover, there are blank reviews and missing profile names. We decide to keep these rows, because we consider the corresponding beer evaluation important, and we replace the missing fields with the string "*other*".
Besides, the *beer/ABV* column collects the measure of "alcohol by volume" of the beers and presents several missing data. It is known that beers of the same style

tend to have similar strength [1], as Figure 2 also shows. Therefore, we decide to fill in the missing values of *beer/ABV* with the *beer/ABV median* of the respective style.

2) *Duplicates*: Reviews made by a user for the same beer (*beer/name*) are considered duplicates and have been dropped (58 entries).
3) *Text features extraction*: Textual features require refined pre-processing and are therefore separated from the main data-set. Reviews are processed using Scikit-learn's *TfidfVectorizer* [2]. This tool allows us to transform text into a meaningful representation of numbers, which will be used to fit the different regression models for predictions. The *tokenization* step splits the text into smaller units, and for each token we:
   - convert it to lowercase,
   - remove (English) stop-words, accents, punctuation, symbols, and digits,
   - perform *lemmatization*, which removes grammatical inflections.

   The weight of each token is measured with the term frequency-inverse document frequency (tf-idf). It favors terms that are frequent in a single sample and penalizes those that are frequent in the entire data-set.
   In particular, instead of extracting single tokens, we extract n-grams, which are contiguous sequence of n items from a given sample of text. The lower and upper bound for the range of the n-grams are set to 1 and 4, respectively.
   Besides *review/text*, we also consider *beer/name* a textual feature due to its high cardinality (see Table I) and string length (see Figure 3), although it is essentially a categorical feature. In this case, we applied *TfidfVectorizer* to extract bi-grams.
4) *Categorical features encoding*: The one-hot encoding technique is used to process categorical features. We use Pandas' *get_dummies*, which leaves the numerical features unchanged, and creates a binary column for each category. This cause a significant slowdown in performances. Hence, we apply SciPy's *csr_matrix* to have a compressed sparse matrix of the obtained results [3].

All the operations described (except the removal of duplicates) have been applied to both the development set and the evaluation set. The last step of the pre-processing phase is to combine again the extracted features to recreate the training and testing data-sets. We define two approaches:

- *all_data* set: all the textual, numerical, and categorical features are gathered to train the regression models;
- *no_name* set: copy of the previous, but the column *beer/name* is excluded to lighten the data-set.

B. Model selection

Below, we provide a brief explanation of the regressors that have been tested and some of their input parameters that we have analyzed.

Fig. 2. ABV Distribution of the 6 Most Reviewed Beers

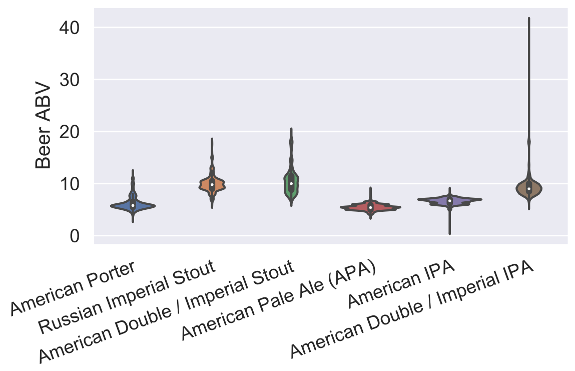
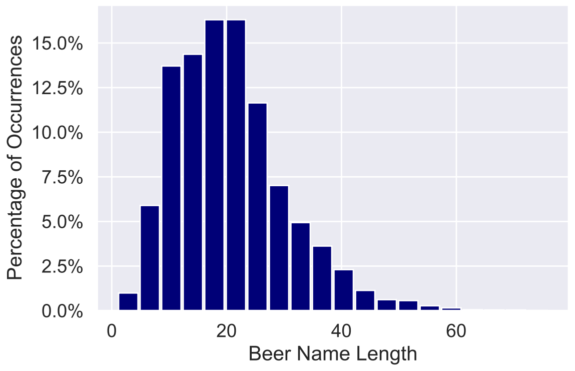


Fig. 3. *beer/name* Length Distribution



- Linear Regression: Scikit-learn's *LinearRegression* fits a linear model to minimize the residual sum of squares between the observed targets in the data-set, and the targets predicted by the linear approximation.
  - *fit_intercept* (bool): specifies whether to calculate the intercept for this model.
  - *normalize* (bool): specifies whether to perform normalization before regression, by subtracting the mean and dividing by the l2-norm.
- Ridge Regression: Scikit-learn's *RidgeRegression* solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Regularization improves the conditioning of the problem and reduces the variance of the estimates.
  - *alpha* (positive float): specifies the regularization strength; larger values imply stronger regularization.
- Linear Support Vector Regression: Based on support vector machines method, Scikit-learn's *LinearSVR* applies linear kernel method, and works well with large data-sets.
  - *C* (positive float): specifies the regularization strength; larger values imply stronger regularization.
  - *loss*: specifies the loss function.
- Gradient Boosted Decision Trees (GBDT): GBDT is an ensemble model of decision trees, which are trained in

sequence. Trees are grown using information from the previous iteration, in order to improve performance on the region of the data-set where previous models were unsuccessful. LightGBM's *LGBMRegressor* speeds up the training process of conventional GBDT by up to over 20 times while achieving almost the same accuracy [4].

  – *n_estimators* (int): specifies the number of boosted trees to fit.
  – *min_child_samples* (int): specifies the minimum number of data needed in a child (leaf).
  – *num_leaves* (int): specifies the maximum tree leaves for base learners.

- Voting Regressor: Scikit-learn's *VotingRegressor* fits several base regressors, each on the whole data-set. Then it averages the individual predictions to form a final prediction. We will use only the best performing models among the previous ones as base estimators.

C. Hyperparameters tuning

The hyperparameters tuning step is to find the best configuration of the parameters for our models. For this purpose, we exploited Scikit-learn's *GridSearchCV*, which implements a "*fit*" and a "*score*" method, such that the parameters of the estimator used to apply these methods are optimized by *cross-validated grid-search* over a parameter grid of possible values. *R2* has been selected as scoring strategy to evaluate the performance of the cross-validated models. We performed 5-fold cross-validation on our regressors, fitting both *all_data* and *no_name* sets, and we obtained the same optimal configuration for the hyperparameters, shown in Table II.

TABLE II
HYPERPARAMETERS TUNING

Model	Hyperparameters	Tested Values	Optimal
LinearRegression	fit_intercept	[True, False]	True
	normalize	[True, False]	False
RidgeRegression	alpha	[0.1, 1, 10, 100]	10
LinearSVR	C	[0.1, 1, 10, 100]	0.1
	loss	[L1, L2]	L2
LGBMRegressor	n_estimators	[100, 200, 300]	200
	min_child_samples	[10, 20, 50]	20
	num_leaves	[31, 62, 124]	31

III. RESULTS

Table III shows the R2 scores achieved on the public leaderboard by the optimal models using both *all_data* and *no_name* sets. In both cases, *RidgeRegression*, *LinearSVR*, and *LGBMRegressor* were chosen as base estimators of the *VotingRegressor*. *LinearRegression* was excluded from the ensemble due to its low performance.

IV. DISCUSSION

The results obtained are satisfactory and show how ensembles are a successful choice, able to outperform their base estimators. Our idea is that by combining different estimators we can obtain a more robust predictor, hopefully non-biased, since we have seen that the data-set is unbalanced.

TABLE III
PUBLIC LEADERBOARD R2 SCORES

Model	all_data	no_name
LinearRegression	0.663	0.656
RidgeRegression	0.706	0.707
LinearSVR	0.705	0.708
LGBMRegressor	0.709	0.708
VotingRegressor	0.717	0.716

The two approaches, *all_data* and *no_name*, yielded similar results, since some models benefited from the absence of the *beer/name* column.

LGBMRegressor performed the best among the base estimators, and we believe it still has room for improvement. As a matter of fact, this regressor has several hyperparameters that have not been tuned, because doing it with the standard grid search is computationally expensive. In this context Optuna could be used in future works. It is an innovative automatic hyperparameter optimization software framework, that efficiently search large spaces and prune unpromising trials with a stochastic approach [5].

REFERENCES

[1] G. Aubert, S. Rogai, and Y. Varoutsikos, *Birra. Manuale per aspiranti intenditori*. Giunti, 2021.

[2] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[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.

[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[5] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.