Regression analysis, with Gradient Boosting and Semantic Meaning Extraction

Marco Tasca
Politecnico di Torino
Student id: s285174
s285174@studenti.polito.it

***Abstract*—-The aim of this report is to address the problem of predicting a numerical target variable, starting from categorical, numerical and textual features. The followed approach is based on gradient boosting and sentence vectorization, which perform significantly better than the decision tree regression model used as baseline.**

I. Problem overview

The data set provided is composed by 100,000 data samples, each row contains different attributes, and represents a specific beer. The purpose of the competition was to perform a regression task to predict the *review/overall* attribute: the numerical quality score assigned to each beer. The target variable's domain consists of all the integer and half integers between 1 and 5.
The data set is divided in two parts:

- A **development** set, composed by 70,000 samples, characterized by 5 numerical features, ABV of the beer (*beer/ABV*) and 4 quality scores: *review/appearance*, *review/aroma*, *review/palate* and *review/taste*, 2 important categorical features (*beer/name* and *beer/style*), and a written review. Moreover, we have the target value. Finally, other features contain information about users who wrote the reviews and assigned the scores, such as username, gender and birthday.
- An **evaluation** set, composed by 30,000 data points with the same structure, except for the target value.

We performed a preliminary inspection on the development set, from which we observed two main issues: the presence of **missing values**, and the **high cardinality** of some categorical attributes, as shown below. The presence of missing values is a **well-known problem** for the majority of regression models, while the high cardinality of categorical attributes may lead to a very sparse and large feature matrix. As a matter of fact, regression models need to be trained with numerical values, encoding categorical features is a fundamental step of pre-processing, but if the cardinality of their domain is too high, it may be unfeasible. To solve these issues, we may discard all the features presenting those characteristics, but this approach would result in a drastic **reduction of training data**, leading to a regression model with **lower performances** and poor generalization power. We decided to adopt a mixed strategy, better described in *section II*.

TABLE I
FEATURE INSPECTION IN DEVELOPMENT SET

Feature	unique values	missing values
beer/ABV	336	3107
beer/name	14770	0
beer/style	104	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

In our data set it is also present **plain text**, in the form of written reviews, one for each beer. It is important to notice how valuable this feature can be. In order to extract value from it, we first need to pre-process and encode it in a suitable way, our approach tried to **minimize the loss of information** and maximize the performance of the model.

The last step of our inspection was to investigate the **distribution of the target variable**, which results to be a normal distribution with a **skew of -1**, as we can see from *Fig. 1*. This will be considered acceptable during our regression task.

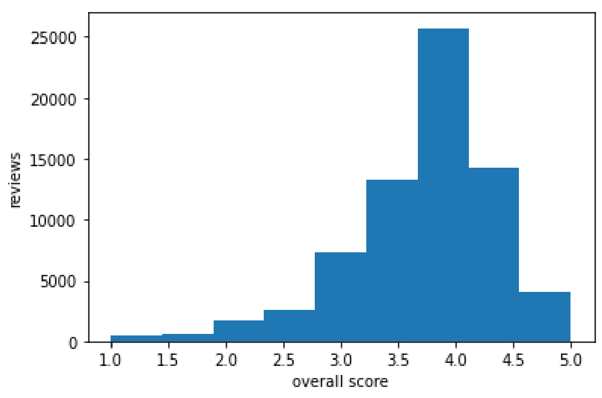

Fig. 1. distribution of the target variable

II. Proposed approach

In each phase of our project, from the pre-processing step to the final result, we used R^2 for evaluation. R^2 is called *coefficient of determination* and it is a statistical measure of how well the regression predictions approximate the real data points. An R^2 equals to 1 indicates that the regression predictions perfectly fit the data.

A. Pre-processing

1) Management of missing values: Some attributes have more than half of values missing, as we can see from *Table 1*. In those cases the simplest and most efficient strategy has been to **discard the whole feature**, in particular, we pruned the data set from the attributes of the users, except for the profilename.

Two features, *beer/ABV* and the written review, *review/text*, present a low number of missing values, and discard them would result in a significant loss of information. Instead, we decided to **fill them with standard values**, an empty string for review, and the average for ABV.

2) Management of categorical attributes: To fit our regression model, all the attributes must be converted in **numerical format**, it is therefore necessary to encode the categorical features in a suitable way. However, the high number of unique values can be a hard problem to solve. Our first approach was to **one hot encoding** all the categorical features, ending up with more than 25,000 columns in sparse format, to later perform **linear dimensionality reduction** by means of truncated singular value decomposition (SVD [1]), which can efficiently operate on sparse matrix. We then compared the result with a greedy approach: the **elimination of all the features** with a cardinality greater than 10,000 (*beer/name* and *user/profilename*). Due to the negligible variation in R^2 score, and the significant improvement in velocity, we adopted the second strategy. Later we proceeded to one hot encoding the remaining categorical attribute, *beer/style*, without performing any dimensionality reduction.

3) Pre-processing and transformation of written reviews: The last variable that has to be pre-processed is *review/text*, this feature consists of plain text, a beer's review written by the user. To prepare it we proceeded in removing numbers, punctuation, excess whitespaces, and casting to lowercase. We took in account three different methodologies, to find the best cost-score approach:

- **Tfidf with word frequency threshold** [1]: term frequency–inverse document frequency, it is a numerical statistic that is intended to reflect how important a word is to a review in the entire collection or reviews. After some tuning, we kept the words with a frequency between 0.9 and 0.1, resulting in a sparse matrix of 163 columns.
- **CountVectorizer & SVD** [1] this has been our baseline for text transformation. First, we one hot encoded each word, and counted its occurrences in every reviews, then we applied dimensionality reduction, ending up with 150 columns.
- **FastText** [2] [3]: an open-source, free, lightweight library developed by Facebook, that allows users to learn text representations and text classifiers, able to extract **semantic meaning** from plain text, and convert it in a numerical form. We first trained it, and then used it to translate each review in a vector of 150 floating numbers.

At the end of our tests, better described in *section III*, we decided to adopt the approach with FastText, which turned out to be the best one overall. In *Table II* it is possible to see a comparison, the tests were executed in cross-validation with 5 folders.

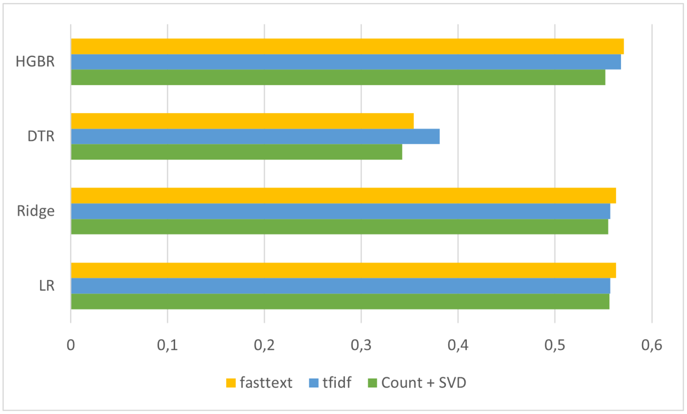

Fig. 2. text transformation score with different models

B. Model selection

We tested several models and compare results, in order to find the best combination of pre-processing and model itself. Tests have been made using **cross-validation with 5 folders**. The mean of R^2 score, calculated for each folder, has been

TABLE II
COMPARISON BETWEEN DIFFERENT APPROACHES IN TEXT TRANSFORMATION

model	R^2 **scores**		
	CountVectorizer + TruncatedSVD	**TfIdf + frequency threshold**	**FastText sentence vectorizer**
Linear Regression	0.556	0.557	0.563
Ridge	0.555	0.557	0.563
Lasso	-2.963e+29	-9.060e+29	-2.964e+29
Decision Tree Regression	0.342	0.381	0.354
Hist Boosting Gradient Regression	0.552	0.568	0.571

used as evaluation metric. The entire pre-processing pipeline described before has been fit and applied on the training set, and then applied to the test set. This process has been repeated for each one of the 5 folds of cross-validation, in order to obtain reliable scores.

The only exception is represented by FastText, in this case we transformed all the reviews **one time**, training FastText's model with all the text contained in the development set, and then converted them in **sentence vectors** at the beginning of our training. This simplified approach had to be adopted due to time reasons.
We will now present the models selected for testing, the scores are shown in *Table II*, to implement them we used Scikit-Learn library [1].

- **Linear Regression:** a simple linear regression model, useful to obtain a starting point and to evaluate the linearity of our regression task.
- **Decision Tree Regression**: a simple regression model used as baseline, we chose this because it is based on the same basic tree-structure as Hist Gradient Boosting Regression model.
- **Lasso & Ridge**: two robust and more complex versions of linear regression. They differ in how the normalization of the linear regression equation is performed, in particular, Lasso is more aggressive and tends to put the coefficients to zero more rapidly than Ridge.
- **Hist Gradient Boosting Regression**: an ensemble of decision trees, where each one tries to improve the prediction of the previous one, using loss function and gradients. We preferred this one over Gradient Boosting Regression, because it is indicated for data sets with more than 10,000 samples, and it is significantly faster.

From these preliminary results, we can see how even the simplest Linear Regression is obtaining a result in line with the much more complex and powerful Hist Gradient Boosting Regression, this indicates a **strong linear relationship** between independent variables and the dependent one. It is also possible to notice how all the models outperformed Decision Tree Regression, which is prone to over-fitting, except for Lasso, that probably needs a fine tuning to better perform.

C. Hyper-parameters tuning

We then proceeded to optimize the chosen model, Hist Gradient Boosting Regression, obtaining a slight - but present-improvement in the overall performances. First, we **manually modified** some parameters to then study the trend of the score, by means of graphs, and then proceed to the tuning of the model through a **grid search**. After various tests we came to the conclusion that this approach was impractical, due to the complexity of the model. Therefore, we opted for a more advanced tool, and we chose **Optuna** [4]. the metric used for the evaluation is R^2.

The Hyper-parameters have been tuned using **10-folders cross-validation** and **sentence vectorization** [3]. Optuna

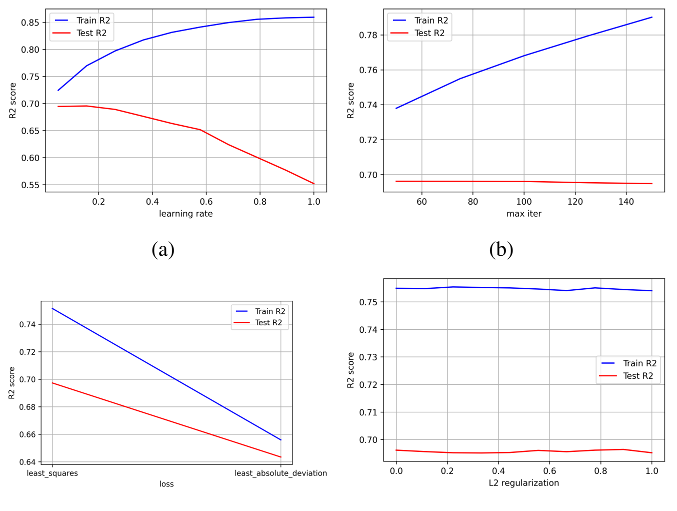

Fig. 3. graphic representation of R^2 varying parameter values

allowed us to process a large amount of different combinations, pruning at early stage the inefficient ones. In particular, we focused on those described in *Table III*. Moreover, during

TABLE III
HYPER PARAMETERS TUNED WITH OPTUNA

parameter	value	description
loss	least_squares	Function used to calculate the loss.
learning_rate	0.0919	used during gradient computation.
max_iter	188	Maximum number of iterations.
max_leaf_nodes	42	Maximum number of leaf nodes.
min_samples_leaf	97	Minimum number of samples in a leaf.
l2_regularization	0.9441	0 means no regularization.
n_iter_no_change	18	Iterations with no loss changing.
tol	8.076e-07	Tolerance in the loss changing.

this last phase of our regression task, we did not apply **any normalization** on the data set. This is possible thanks to the structure of trees: each split is performed considering only one feature at a time, as a result **trees are not affected by any monotonic transformation**.

III. Results

TABLE IV
COMPARISON BETWEEN MODELS SCORES

model	R^2 score
on development set	
Decision Tree Regression	0.354
Hist Boosting Gradient Regression (before tuning)	0.570
Hist Boosting Gradient Regression (after tuning)	0.577
on submission platform	
Decision Tree Regression	0.450
Hist Boosting Gradient Regression (before tuning)	0.704
Hist Boosting Gradient Regression (before tuning)	0.705

In this section we will briefly show and discuss the results. As we can notice in *Table IV*, we observed a significant increment in the R^2 score between our baseline and the HIst Gradient Boosting Regression model, ever before tuning. During the tuning process, we **struggled to obtain an increase** in the result. We tuned our final model for **4 hours**, in order to boost the accuracy of the predictions, the final values of the parameters, with a brief description, are shown in *Table III*.

IV. DISCUSSION

It is quite interesting to see how **all the models scores** are in line, except for Decision Tree Regression model, probably due to over-fitting, and Lasso, which has not been tuned.

The Linear Regression model, in particular, performed brilliantly, despite of its simplicity. Our hypothesis is that the problem we tried to solve, after encoding of categorical data, removing of missing values and text transformation, maintained a **high linear behaviour** in the relationship between dependent and independent variables.

Moreover, during the fine tuning phase of Hist Gradient Boosting Regression model, we managed to achieve a **very little increase** in R^2 score, even if we accurately performed it with the help of **advanced systems** like Optuna, and for a significant amount of time. Our conclusion is that the default parameters of our final regression model were already well optimized to efficiently solve this regression task. It is also important to notice that even between the different methodologies tested during the text-transformation phase, the difference in term of scores was **definitely small**.

There is the possibility that, during the pruning of attributes in the **pre-processing phase**, we unwillingly discarded a large amount of useful information, that could have been encoded in a efficient way, and **rise the final score** of our model.

We may consider to **deeper analyze** relationships between the target variable and the rejected features, and to study different methods to extract value from them.. Despite of the **struggle to obtain an increase in the scoring**, we can consider ourselves satisfied with the achieved results, and with the **validity of our approach**.

this project was made possible by the Scikit-Learn library [1], which we used extensively to process the data and implement the different models.

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] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," *CoRR*, vol. abs/1607.01759, 2016. [Online]. Available: http://arxiv.org/abs/1607.01759

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *CoRR*, vol. abs/1607.04606, 2016. [Online]. Available: http://arxiv.org/abs/1607.04606

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *CoRR*, vol. abs/1907.10902, 2019. [Online]. Available: http://arxiv.org/abs/1907.10902