Summer call project of DSL
Farzad Imanpour Sardroudi
Politecnico di Torino
Student id: s289265
s289265@studenti.polito.it

***Abstract*—In this report we propose a possible approach for predicting the overall score associated with a beer review. In particular, this approach consists on encoding the categorical features, managing outliers, null values, and a textual field. Once completed the preprocessing step, three regression models have been compared and they all outperformed a naive solution, obtaining a satisfactory result.**

I. Problem Overview

The proposed competition is a Regression Problem based on a collection of 100000 beer reviews, divided into development set 70000 and evaluation set 30000, The aim of the competition is to predict the overall score, defined for a given review, by considering 13 features reported in Table 1. All the categorical features present a wide cardinality do- main, in a range between 9 (appearance) and 14770 (name).

From Table I it is clear that both development and evaluation set 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 this aspect must be taken in consideration for the processing phase.

In addition to these categorical features, there is the text field, that contains the textual description of the review. We discovered that there are 69976 unique text over a total number of 70000. Also, we discovered that the development set does not contain any duplicates.

Also we can comprehend from Fig. 1 that out target feature has negative skewness because Negative Skewness is when the tail of the left side of the distribution is longer or fatter than the tail on the right side. The mean and median will be less than the mode. Overall has a skewness equals to -1.0235 and a kurtosis equals to 1.6286. The behavior can be observable also in a graphical representation of the variable reported in Fig .1.

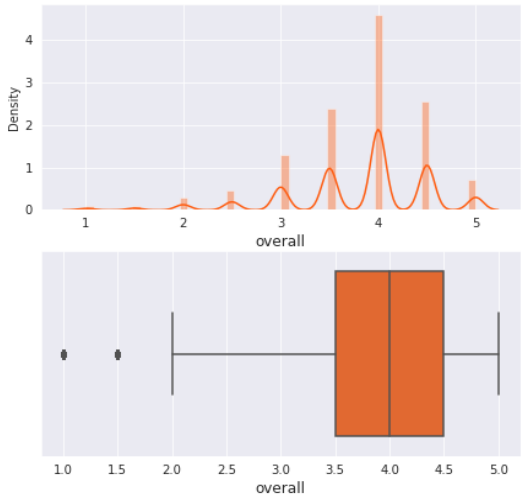

Fig. 1. Distribution of the overall

TABLE I
CARDINALITY AND MISSING VALUES IN DEVELOPMENT SET

	Development set		Evaluation set	
Feature	Cardinality	Null values	Cardinality	Null values
ABV	3443	4.439%	1577	4.283%
name	14770	0.0%	9305	0.0%
style	104	0.0%	104	0.0%
appearance	9	0.0%	9	0.0%
aroma	9	0.0%	9	0.0%
palate	9	0.0%	9	0.0%
taste	9	0.0%	9	0.0%
text	69976	0.026%	29996	0.013%
ageInSeconds	57307	79.079%	25144	79.093%
birthdayRaw	1883	79.079%	1369	79.093%
birthdayUnix	57237	79.079%	25096	79.093%
gender	3	59.741%	3	59.567%
ProfileName	10574	0.02%	7272	0.01%

It is also interesting that the overall associated to most categorical values has a considerable variance. For example in Figure 2 it is possible to observe the top 5 styles (based on the median overall), and most of them covers a wide range.

Furthermore, From Fig .3 we can comprehend that aroma, palate and taste have the highest correlation with target feature compared to other features.

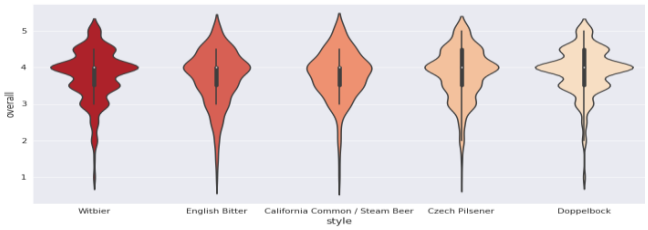

Fig. 2. Top 5 styles by median overall

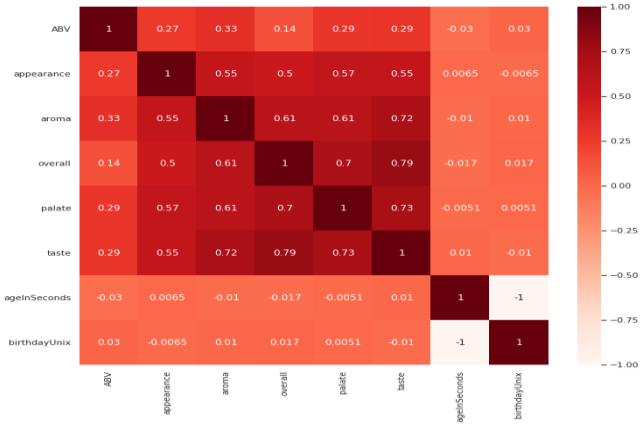

Fig. 3. Correlation between features

II. Proposed Approach

A Data preprocessing

The categorical attributes have their missing values replaced by a new value, representing an unknown quantity. Scikit-learn's OneHotEncoder is used to perform this transformation, with the only parameter differing from the defaults being handle_unknown='ignore'. This allows us to handle values never seen seen before by returning a vector of zeros. Other approaches such as Label Encoding and Target Encoding which were explored but resulted in either a decrease in performance or statistically insignificant difference [1]. Additionally, One-Hot encoding provides interpretable features, since each attribute value maps to one and only one column, allowing us to compare them easily. it creates a binarycolumn for each category, as showed in TABLE II.
Assuming that a single cell is saved as an integer value (4 Bytes), a normal matrix would occupy approximately

$$\sum_{i=0}^{n} Card(column_i) * 70000rows * 4Byte \approx 60\ GByte$$

Fig. 4. Memory usage of a the feature matrix

A possible solution is using a sparse matrix representation, where for each row only the non-zero cells are saved

$$n * 70000rows * 4Byte \approx 3.6\ MByte$$

Fig. 5. Memory usage of the feature matrix with a sparse matrix representation

TABLE II
ONE HOT ENCODING

style		overall
Oatmeal Stout		4.5
American Strong Ale		4.0
American Strong Ale		3.5
One Hot Encoding		
Oatmeal Stout	American Strong Ale	overall
1.0	0.0	4.5
0.0	1.0	4.0
0.0	1.0	3.5

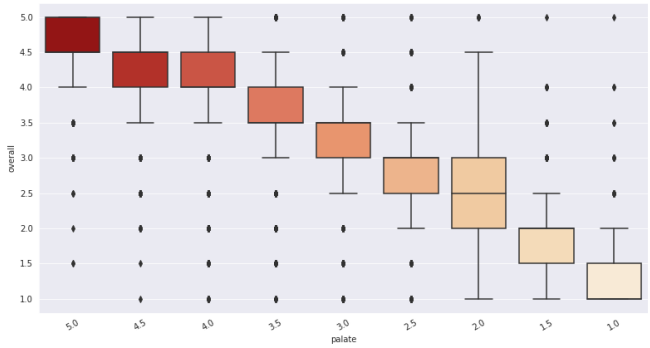

Fig. 6. Overall distribution in Palate

As we can observe from Fig .3 the *Palate* and overall has direct correlation, when the palate increase the overall will increase as well.
Furthermore, the age features are the features with lowest correlation with target feature and also they are representing almost the same information, so ageInSeconds, birthdayRaw and birthdayUnix(the number of seconds that have passed since Jan 1, 1970) can be summarized to one feature.
Then, as it can be clearly seen in Table 1, *age features* contain around 80% and gender contains around 60% of null values, furthermore we decided to impute missing values use **KNN imputer** which utilizes the k-Nearest Neighbors method to replace the missing values in the datasets with the mean value from the parameter 'n_neighbors' nearest neighbors found in the training set. By default, it uses a Euclidean distance metric to impute the missing values. One thing to note here is that the KNN Imputer does not recognize text data values. It will generate errors if we do not change these values to numerical values, A good way to modify the text data is to perform one-hot encoding or create "dummy variables". The idea is to convert each category into a binary data column by assigning a 1 or 0. Another critical point here is that the KNN Imptuer is a distance-based imputation method and it requires us to normalize our data. Otherwise, the different scales of our data will lead the KNN Imputer to generate biased replacements for the missing values. For simplicity, we will use Scikit-Learn's MinMaxScaler which will scale our variables to have values between 0 and 1.

Lastly, the text feature required a refined preprocessing, in fact it has been removed from the main data structure and it has been preprocessed separately. The first step regards the *tokenization*, it splits a piece of text into smaller units called tokens. Then, we removed the stopwords (most common English words) and for each token we:

- converted it to *lowercase*,
- removed digits and punctuation,
- removed the grammatical inflections (*lemmatization*).

The lemmatization technique is really effective in Natural Language Processing (*NLP*) problems, because it tries to reproduce the intended meaning, by grouping together similar words, so they can be analysed as a single item. For example *plays*,*play*,*playing* will be all considered as *play*. Another similar technique is the *stemming*, that extracts the root (consulting, consult et simila will be transformed in consult). So, the main difference is that the former extract a meaningful reducedword, whereas the latter is interested just on its root form. They present similar performances, even if the stemming is a bit faster.

The weight of each token has been measured with the term frequency-inverse document frequency (**TF-IDF**), defined as follows:

$$TF - IDF(t) = freq(t, d) \times \log \frac{m}{freq(t, D)} \qquad (2)$$

Where *t* is the token, *d* is the current document, and *D* is the collection of *m* documents. The idea of the TF-IDF is to stand out tokens that occur frequently in a single document, but rarely on the entire collections of documents. In particular

where t is the token, d is the current document, and D is the collection of m documents. The idea of the TF-IDF is to stand out tokens that occur frequently in a single document, but rarely on the entire collections of documents. In particular, instead of extracting single tokens, we considered the 4-grams, that are subsequences of 4 tokens.

In addition, we tried 2 different weighting approaches (tf-df and tf-idf) and several combinations of TfidfVectorizer parameters (min_df∈{0.1,0.01,0.001},max_df∈{0.7,0.8,0.9}),ngram_range∈ {(2,4),(2,5),(1,4)}. We also changed N, the maximum number of words to extract, and let it vary in the range (150-1500). However, we noticed that all these changes did not affect much the performance, so we preferred the most effective method in terms of memory saving and computation time (min df = 0.8 , max df = 0.01 and ngram=(2,4)).
The wordcloud shown in Fig. 5 gives us a graphic idea of the words' distribution.
Finally because both One-Hot encoding and TF-IDF return sparse matrices, their concatenation still is sparse. After concatenating them we have 70000x30170 sparse matrix of type '<class 'numpy.float64'>' with 1034016 stored elements in COOrdinate format>.

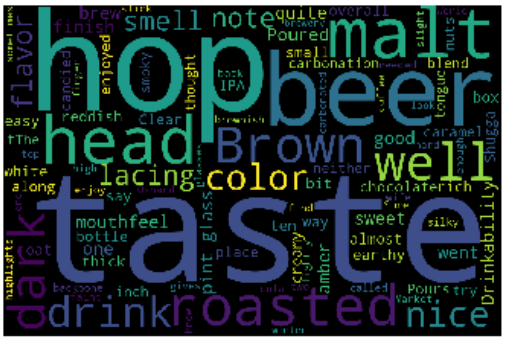

Fig. 7. WordCloud of reviews

B. Model Selection

The following algorithms have been tested:

• **Linear Regression:** it is a model that assumes a linear relationship among its features. The prediction is made by computing a weighted sum of the input features and the intercept.

• **Ridge:** it is a regularized version of Linear Regression and it forces the learning algorithm to keep the model weights as small as possible. This regularization can be made with the hyperparameter α. If it is very large, all weights end up very close to zero.

• **Stochastic Gradient Descent Regressor (SGD):** it picks a random instance in the training set at every step and computes the gradients based only on that single istance. Due to its stochasticity, the cost function bounces up and down and slowly converges to the minimum. This randomness is good for escaping from the local optima, but on the other hand it cannot settle at the minimum. This issue can be partially solved by considering the behaviour of the well known simulated annealing algorithm. For all these regressor models, we found out the best con- figurations through the grid search approach, as it is explained in the next section.

C. Hyperparameters tuning

The best configuration can be found by using the Grid Search Cross Validation technique that works as follows :

1) Generates all the possible candidates that needed to be trained.
2) Select the best configuration.

For fine-tuning, we did not use GridSearch or RandomSearch for finding the best parameters, these methods are time-consuming methods and since our data is large, it is not optimal to use these methods for fine-tuning.
We did this process in a fully experimental way, by trying different parameters.

TABLE III: SUMMERY OF HYPERPARAMETER TUNING

Model	Hyperparameters	Values
Linear Regression	*fifit intercept* *normalize*	*{*True, False*}* *{*True, False*}*
Ridge	*alpha*	*{*0.01, 0.1, 1, 10*}*
SGD Regressor	*loss* *penalty* *alpha* *eta0*	*{*'squared loss"*}* *{*'l1','l2'*}* *{*1e-5,1e-4,...,1*}* *{*0.01,1*}*

III. RESULTS

We will now compare the results obtained with the three models, using the hyperparameter discussed in the previous section

• The best combination of parameters found for the Linear Regression is { fifit intercept=True, normalize =False}. With this configuration we obtain R2 = 0.698. We might consider to perform a deeper search to further optimize this model. Anyway, since we are considering it only as a baseline, the score obtained is more than satisfactory.

• The best combination of parameters found for the SGD Regressor is loss = 'squared loss0 penalty = 'l1' alpha = 1e-5 eta0 = 0.1}. With this configuration we obtain R2 = 0.682. We might consider to perform a deeper search to further optimize this model, which is lowest among other 2 models.

• The configuration producing the best result for the Ridge is{alpha = 0.01}. With this set of hyperparameters we obtain R2 = 0.711, and we prefer this model for our experiment.

TABLE IV: HYPER PARAMETER CONFIGURATION

Model	Best hyperparameters	Public score
Linear Regression	*fifit intercept = True* *normalize = False*	0.698
Ridge	*alpha = 0.01*	0.711
SGD Regressor	*loss = 'squared loss0* *penalty = 'l1'* *alpha = 1e-5* *eta0 = 0.1*	0.682

To conclude, we conduct an interpetability analysis of the Ridge model in order to better understand which are the factors that contribute, both positively and negatively, to the overall of a particular beer. This also allows us to identify potential biases in the model, which could have catastrophic consequences,

especially for a user-facing service. Since Ridge is a linear model, its coefficients directly translate to an importance score for each of the features [2]. We fetch these coefficients from the trained, top-performing model found in grid search. In table V we include the most negative features (both categorical and textual). Table VI includes the most positive features. After further investigation we discover that these are all very selective and exclusive beers. In table VII the most negative textual features are shown. Finally, in table VIII the most positive textual features are shown.

TABLE V:
MOST NEGATIVE FEATURES

Feature Name	Value
hazy light rust	-0.88
beer	-0.88
drink slightly limited	-0.87
abv relative	-0.82
abv overall pale color	-0.79
abv pours clear ruby	-0.73
abv weyerbacher	-0.72
abv kind beer tell	-0.72
abv make sippin beer	-0.68
abstractpatterned lace nose	-0.67
accent taste complex malt	-0.66
fantastic desert	-0.65
ab pine	-0.63
almondvanilla finish	-0.63
abv low appeal better	-0.62
acceptable citrus	-0.62
doppelbock germany	-0.62
bit unexpected spice pepper	-0.61
accept pour just got	-0.61
oakey sourness aging love	-0.61

TABLE VI:
MOST POSITIVE FEATURES

Feature Name	Value
bottled reviewed	2.16
electricity traditional saison	1.85
bottled	1.06
hint tastness mouthfeel	1.02
dripping clump glass aroma	0.96
shed soapy	0.93
elixir equally akin american	0.90
faint trace lime	0.89
accented settle	0.85
accomodate generous head beer	0.82
ice sparkling straw	0.80
abv wa high did	0.80
lingers quite cocoa	0.78
abrasiveness like double	0.76
drinker say nt got	0.69
abv awesome overall	0.68
abv palatable	0.68
accessible geuze	0.68
abv lightish body solid	0.66
drinkability nt detect abv	0.66

TABLE VII:
MOST NEGATIVE TEXTUAL FEATURES

Feature Name	Value
hazy light rust	-0.88
beer	-0.88
slightly limited	-0.87
abv relative	-0.82
abv overall pale color	-0.79
abv pours clear ruby	-0.73
abv weyerbacher	-0.72
abv kind beer tell	-0.72
abv make sippin beer	-0.68
abstractpatterned lace nose	-0.67

TABLE VIII:
MOST POSITIVE TEXTUAL FEATURES

Feature Name	Value
bottled reviewed	2.16
electricity traditional saison finelytuned	1.85
bottled	1.06
hint tastness mouthfeel	1.02
dripping clump glass aroma	0.96
shed soapy	0.93
elixir equally akin american	0.90
faint trace lime	0.89
accented settle	0.85
accomodate generous head beer	0.82

IV. DISCUSSION

The current approach obtains an overall satisfactory result, outperforming the proposed baseline. In particular, removing the duplicates and managing the textual field has strongly influenced the performances. Further considerations can be made by reducing the categorical domains. It can be performed by iteratively removing subset of less frequent categorical attributes. Then, it may be helpful to reduce the overall size of the final data structure with the fairly well known Principal Component Analysis. It would require multiple tests, in order to analyze the performances with a decreasing number of components. Then, as we saw, the text feature has turned out to be fundamental for boosting the performances, so another key aspect for further improvements regards the textual preprocessing. It would be relevant to try several other different ways for preprocess it. In fact, without removing the punctuation and the non-alphabetical characters we noticed an improvement on the performances (0.882 on the public score), even if avoiding these important steps should add just ”noise”.

Furthermore, It’s Better to define the threshold used before one-hot encoding or adopt different techniques of encoding categorical features (e.g., frequency encoding);

Finally, run a more exhaustive grid search both on preprocessing and regressors hyperparameters. Moreover, additional regression models might be considered.

REFERENCES

[1] Seger, Cedric , “An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing” [2] G. James, D. Witten, T. Hastie, and R. Tibshirani, An Introduction to Statistical Learning: with Applications in R. Springer, 2013.