Beer Data Analysis

Distributed Data Analysis and Mining

A.Y. 2022/2023

Authors

Pierluigi Basile
Marco Mannarà
Emanuele Sabatini
Simona Sette
Federico Volpi

UNIVERSITÀ DI PISA

Contents

Dataset Presentation

The dataset "Beer data analytics" resides on Kaggle[1] and contains information about different aspects that characterize beers like the different types of beers, the beer style, the absolute beer volume, the beer name and brewer name, appearance, taste, aroma, overall ratings, and consumers review.
The dataset has been made available through csv file and its dimensions are 414 megabytes and contains 528870 records.
There are thirteen features: five of those (*beer_name, beer_style, review_profileName, review_textt* and *review_time*) are categorical attributes, while the remaining ones are numerical.
The original 13 features, their descriptions, the original data types and the value ranges contained inside each one are the following:

Features	*Data type*	*Description*	*Value ranges*
beer_ABV	Double	Degree of alcohol level for each beer.	0.01-57.7
beer_beerId	Int	Distincts beer ID.	3-77.3k
beer_brewerId	Int	Distincts brewer ID.	1-28k
beer_name	string	Distinct beer names.	18339 unique values
beer_style	string	Name given to beers that share characteristics (color, flavor, aroma, alcohol content, etc.).	104 unique values
review_appearance	double	Beer appearance score attributed by a user.	0-5
review_palette	double	Beer color palette score attributed by a user.	1-5
review_overall	double	General score attributed to a specific beer from a specific user.	0-5
review_taste	double	Beer taste score attributed by a user.	1-5
review_profileName	string	Nickname of the user who made the review.	22801 unique values
review_aroma	double	Beer aroma score attributed by a user.	1-5
review_text	string	Textual content of a review made by a user.	528372 unique values
review_time	string	Temporal instant recording of the review submission in UNIX format.	884m - 1.33b

Tab 1. Dataset features descriptions.

[1] https://www.kaggle.com/datasets/gauravharamkar/beer-data-analytics?resource=download

Data Exploration

Each dataset record contains a specific review made by a specific user for a particular beer: the number of reviews for each beer is not homogeneous, there are 18339 different beers and 104 different beer types, the most common one is the 'Sierra Nevada Celebration Ale', an american christmas IPA, while the most common typology of beer is the American IPA.
Missing values are present but only on three features among all:

- *Review_profileName* and *Review_text* with 115 and 119 missing values respectively, corresponding to the 0.02% of the whole dataset.
- *Beer_ABV*, that contains beer's alcohol content, has approximately 20280 missing values, corresponding to the 3.8% of the whole dataset.

Data Cleaning and Preparation

After a first exploration phase, a process of solving problems discovered within the dataset followed.
At first, the focus was on finding a reasonable way to fill the vast majority of missing values detected, specifically regarding the *beer_ABV* feature since it was the one with the biggest number of missing values present within the set.
In order to deal with less data before starting this process, a feature selection phase was made resulting in the elimination of *review_time* and *review_profileName* from the dataset, since they were not considered features of interest for the further analysis. The same was "temporary" made also for the r*eview_text* feature since it will be considered just in a specific classification task made on this feature, but not for all the other analyses.
Before its exclusion, however, in view of future NLP analysis, it was decided to treat the 119 missing values of *review_text* by deleting the rows, since a loss of the 0.02% of the set would have been acceptable.
In order to fill the missing values inside *beer_ABV* the chosen criterion was to fill them depending on a condition:

- If beer with the same *beer_name* exist and it exist a value for their *beer_ABV*, the filling was done with the rounded average of *beer_ABV* values present in the beers with the same name;
- In case of unique *beer_name* values (so there was no record inside the set where these specific values for *beer_name* exist) it was decided to take the rounded average of the *beer_ABV* value referring to the shared *beer_style* between the record to fill and the ones within the dataset.

This process of missing values filling was realized by exploiting Spark SQL queries for the rounded average calculus since it was considered the most effective and dynamic method to proceed in data extraction and processing.
For the two filling process a combination of different Spark functions were used in order to only fill the remaining NULL values inside the *beer_ABV* feature: the process was composed by a left join between the spark dataframe generated from the rounded average generation considering the *beer_style* (or *beer_name* in the first filling phase) and the spark dataframe that needed the filling, considering the *beer_style* (or *beer_name* in the first filling phase) as foreign key on which to base the join.

Internally, this was made possible by the *coalesce()* spark function, that generated a new column composed by the original *beer_ABV* values with the newly computed rounded averages in addition, followed by a drop of the original *beer_ABV* column from the desired dataset and the renaming of this newly generated column with the same name of the dropped one (*beer_ABV*).

Data Understanding

We proceeded to observe the *review_overall* feature and found out that, while it was moderately correlated with the other numerical reviews, it did not match the average of the numerical reviews. This drove us to create a new feature called *review_mean* corresponding to the average resulting from *review_appearance*, *review_palette*, *review_taste*, *review_aroma* all together.
The manipulation of the values inside those columns and the creation of a new column containing them was realized with the transformation spark function *withColumn(),* whose syntax allows the access to pre-existing columns and apply expressions on the values within and transform them into new values. In our specific case the arithmetic average was applied to the values of the 4 features considering them "row by row" in order to generate the new value for the specific row inside the new *review_mean* feature.
This led us in the creation of a dataset version containing only the numerical values, that was used in the data correlation exploration.

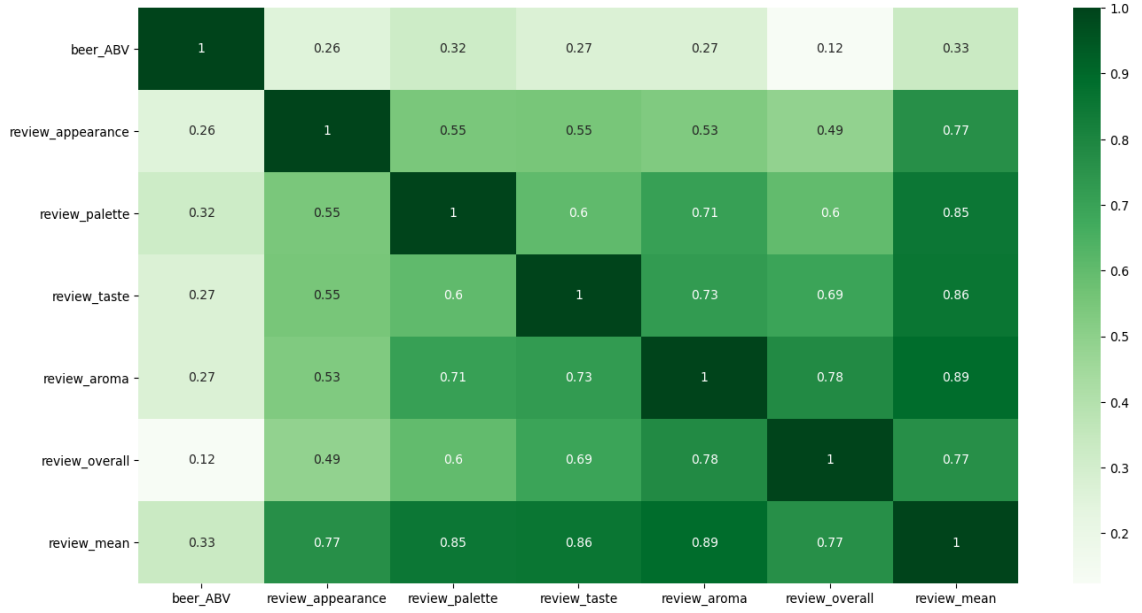

Fig 1. Correlation Matrix generated from the numerical values dataset.

From the feature correlation exploration a noticeable value of correlation (more than 0.7) between beer aroma and taste, beer aroma and the overall score assigned by the user, but also between palette and aroma can be observed. Also, it was decided not to consider to comment on the high correlation values between the *review_mean* feature and the other features because of the artificiality of this one, since it was specifically created as a mean value of other four features of the dataset, so the strong link between those is normal and expected but not meaningful.

Furthermore, another transformation was performed on the entire dataset: it was decided to apply an aggregation to the dataset using the *beer_name* and *beer_style* features, and averaging the other numerical features, excluding *beer_beerId*, *beer_brewerId* and *review_text*. The reason for this choice was to obtain a dataframe where each row consists in the average of the reviews for the same beer and not the review of a single user like the original one, so as to obtain clearer and more significant results in subsequent tasks.

Data Distribution

It was decided to investigate the distributions of the data features of interest within the dataset and different density plots have been created for this purpose. Those have been created using the aggregated dataset for better computational performances.
The first density plots have been made on the alcohol content and *review_mean* in order to understand the distribution of the alcohol content of the beers within the set and the resulting average values from the different kinds of reviews.

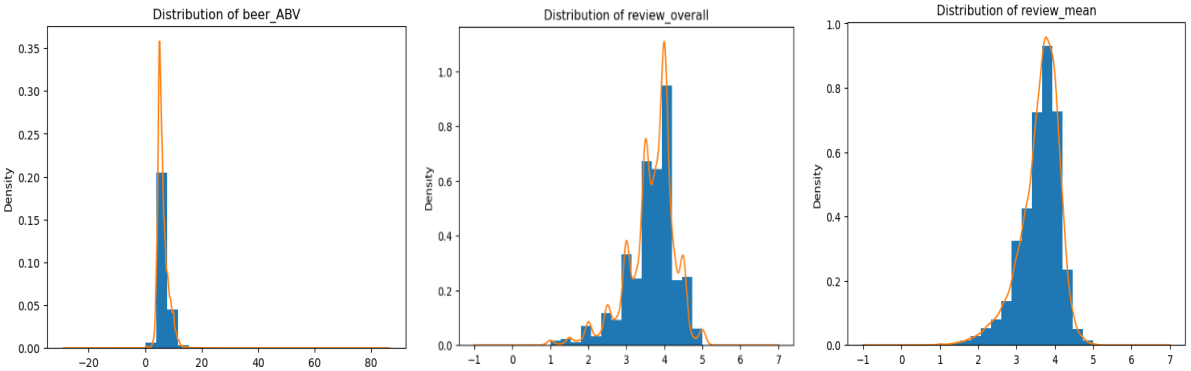

Fig 2.Beer_ABV, *review_overall* and *review_mean* distributions.

For the alcohol content one is noticeable that the majority of the beers are characterized by a pretty low value of alcohol content, and this is coherent to the real beer's alcohol contents distribution since the majority of them, especially commercial ones, do have pretty low alcohol values.
Review_overall is a feature provided by the dataset and contains the overall score given to a beer from the users, and its distribution show a clear tendency to not exceed either negatively or positively in the beer scoring; still, there are more beers with really low scores (2,3) rather than beers with the maximum score available.
The artificial feature *review_mean* contains the mean values of 4 types of reviews on different beer aspects, and shows a concentration of values between 3 and 4 and really few values that go further or backwards from this range. Since this is a mean value, the concentration of the values around the middle scores is coherent with the attitude of the average to "smooth" the results towards values that are not at the extremes of the evaluation scale.
For this reason single data distribution explorations were made specifically on the 4 features scores that contributed to the average one:

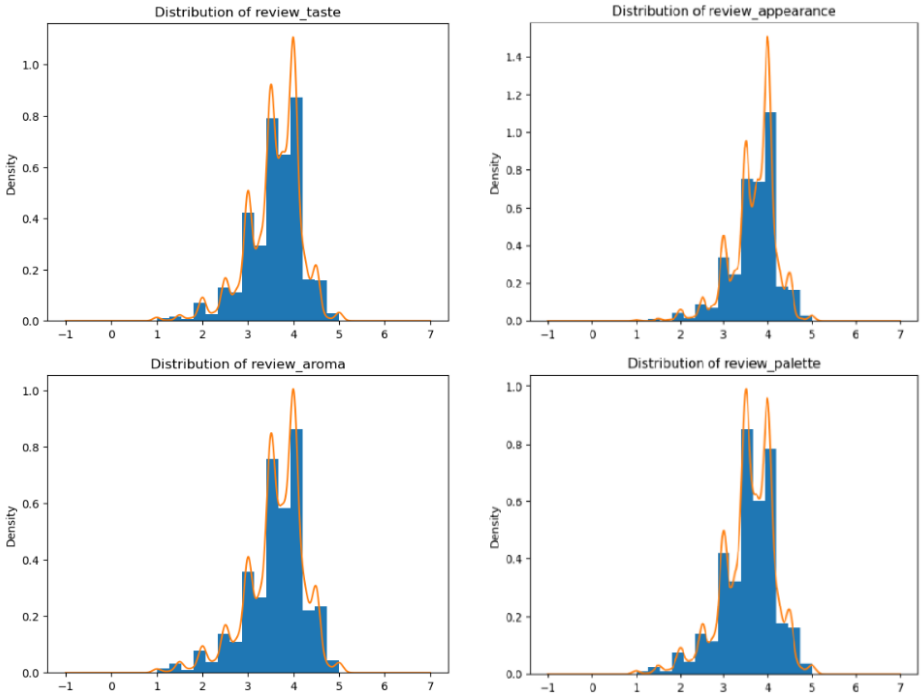

Fig 3. Distributions of the 4 features from which review_mean is derived.

The smoothing effect of the average said before is clearly visible looking at the original features distributions from which the average is derived, even if the tendency of the values to concentrate between 3 and 4 is still visible from the original features too.

Scatter Plot Explorations

In order to deeply explore possible correlations between different features, corresponding to different beer characteristics, it was decided to plot several scatter plots.

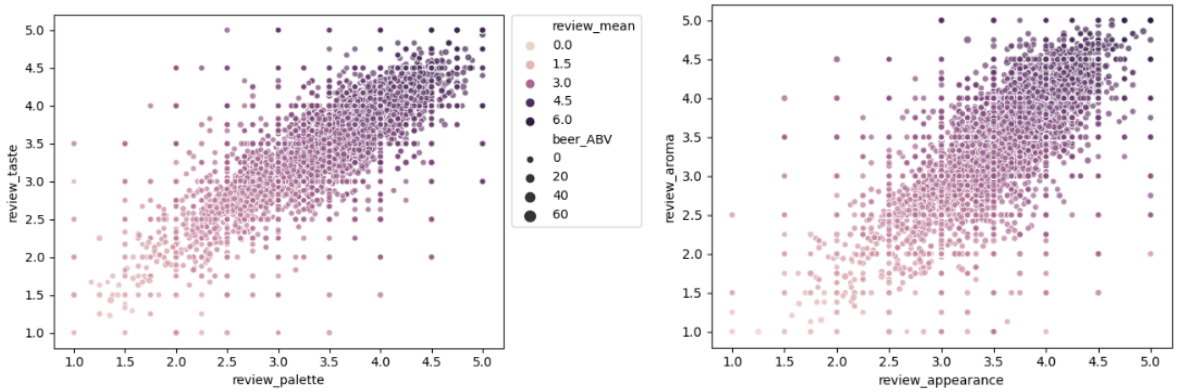

Fig 4. Scatter plots between users reviews on 4 different beer characteristics (taste and palette, appearance and aroma).

It is evident from the plots that there is a general strong correlation between all the different score levels, so that high scores in a particular beer aspect is generally linked to high scores on other aspects; that's also the reason why the *review_mean* score gradually follows the same kind of trend. Also, it is noticeable that the general alcohol content of the beer is low (below 20), reflecting the characteristics of commercial beers.

Clustering

K-means

The pre-processing phase consisted in the vectorization and application of the scaler to the features selected for this task (*beer_ABV, review_appearance, review_palette, review_taste, review_aroma, review_overall and review_mean*) post-vectorization using the "VectorAssembler" and the "MinMaxScaler" from the pySpark library.
A first approach to the tuning of the k parameter was to set a for loop to compute the silhouette score for incrementing values of k (from 2 to 21 excluded).

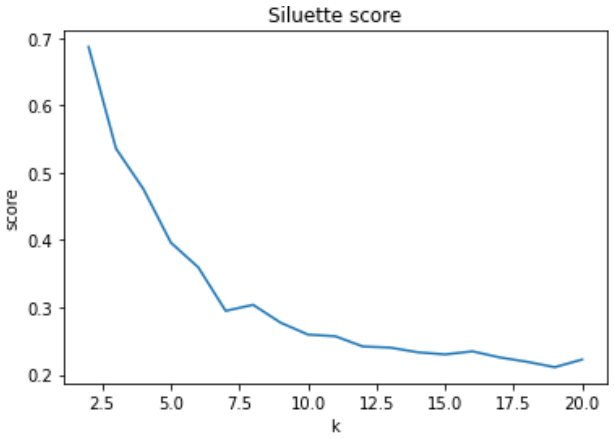

Fig 5. Silhouette values for increasing values setting (2 to 21 excluded) of the k parameter for the K-means algorithm.

The resulting plot shows a clear peak for a value of k equal to 2, so it was decided to proceed with this setting for the k parameter.
The K-means algorithm was trained with a number of k equal to 2 on a tailored dataset for this task, which contains *beer_name, beer_style* and *features_scaled* as features.
The silhouette for this parameter setting provided a score of 0.687, which indicates a pretty decent separation between clusters and at least not an overlapping situation.
After the centroids computation and through the use of a function purposely defined to inverse the MinMax normalization of the centroid and retrace to the original data ranges, followed by a merge of the predictions with the original dataset through an inner join, it was possible to plot different scatter plots considering different couples of attributes to visualize the 2 clusters generated by the K-means algorithm. It was decided to plot the clusters for the following couples of dataset attributes: *review_aroma* and *review_taste*, *review_appearance* and *review_overall*, *beer_ABV* and *review_overall*.

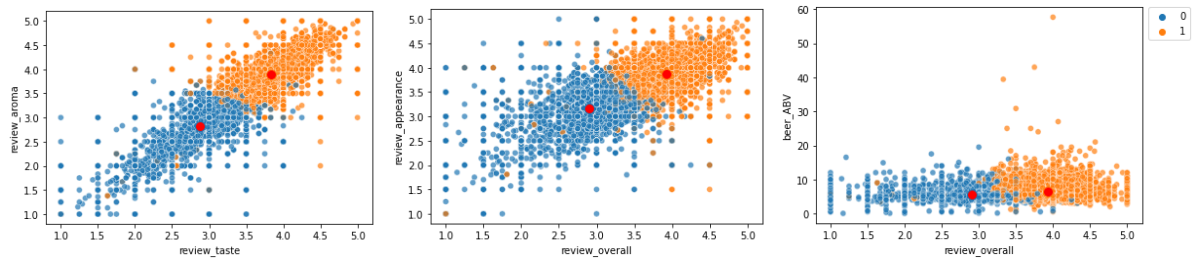

Fig 6.Scatter Plots generated by different combinations of attributes with the purpose of showing the clusters generated by the K-means algorithm with k=2.

As the silhouette score previously suggested, it is possible to appreciate a well-visible separation of the 2 clusters generated by the K-means algorithm and overall a good data separation in the cartesian space.

Bisecting K-means

In order to try algorithms based on different mechanisms, beyond the K-means it has also been decided to make an implementation of the Bisecting K-means algorithm. As done for the K-means, also here it was decided to choose the most ideal value for k through a for loop to compute the silhouette score for incrementing values of k (from 2 to 15 excluded).

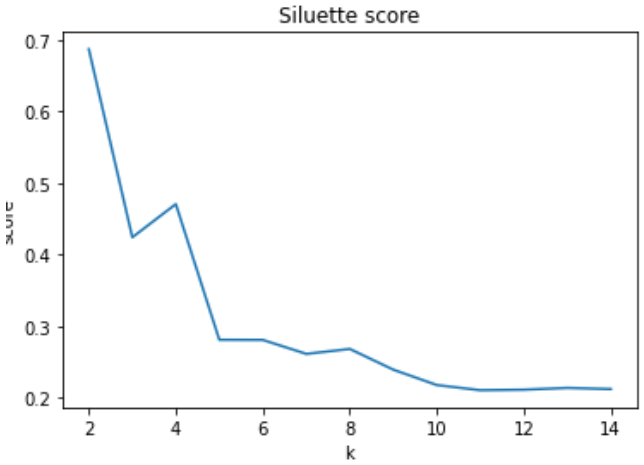

Fig 7. Silhouette values for increasing values setting (2 to 15 excluded) of the k parameter for the Bisecting K-means algorithm.

The resulting plot shows a clear peak for a value of k equal to 4, so it was decided to proceed with this setting for the k parameter.
For this k parameter setting the corresponding silhouette score value is 0.47, which theoretically does not return a completely clean, not overlapping and well-separated cluster generation; but this setting could not be helped since a parameter setting of 2 for the Bisecting K-means would mean a methodology functioning practically identical to the K-means, that we already have implemented with a k equal to 2, so it would result in the same results. For this reason it was decided to still proceed with k equal to 4 although the situation turned out to be not ideal.
After the centroids computation and through the same function used for the K-means, different scatter plots for the same couples of attributes used for the K-means were realized in order to visualize the 4 clusters generated by the Bisecting K-means algorithm.

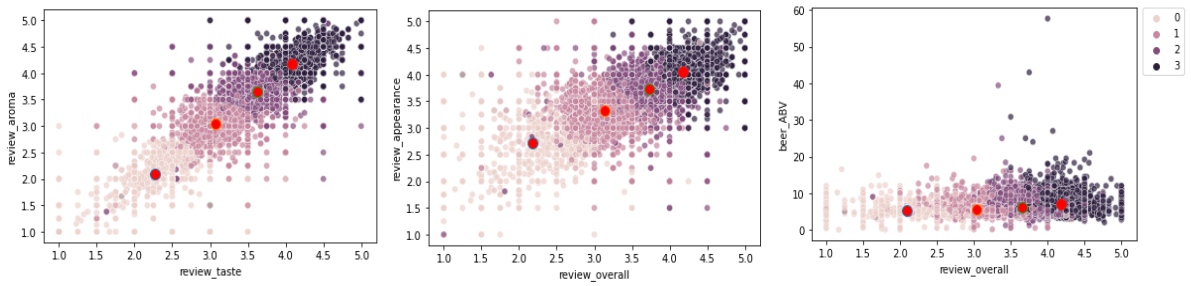

Fig 8.Scatter Plots generated by different combinations of attributes with the purpose of showing the clusters generated by the Bisecting K-means algorithm with k=4.

Despite the "risky" value of the silhouette score for this k setting, the division of the data inside the cluster is still well defined and consistent, so the 4 clusters generated by the Bisecting K-means algorithm can still be considered a decorous result.

Clusters Analysis

In order to understand and investigate possible interesting insights concerning the composition of the cluster for both the K-means cluster and Bisecting K-means ones, the possibility of querying the output schemas has been exploited, grouping by the *beer_style* feature and counting the number of points for the case when prediction is 0 and 1 for the K-means and from 0 to 3 for the Bisecting K-means.

beer_style	*Zero*	*One*
Baltic Porter	88	12
Milk / Sweet Stout	105	20
Dubbel	135	21
Tripel	217	32
Wheatwine	30	3

beer_style	*Zero*	*One*	*Two*	*Three*
Baltic Porter	2	10	32	56
Milk / Sweet Stout	5	15	47	58
Dubbel	4	17	75	60
Tripel	4	28	122	95
Wheatwine	0	3	11	19

Tab 2.Beer styles distribution through the clusters generated by K-means (on the left) and Bisecting K-means (on the right).

For the same types of beers clustered though different algorithms, is it possible to notice a major concentration of beer types in cluster 2 and 3 for the Bisecting K-means and cluster 0 for K-means, regardless of the beer style.
Then, for each column, an aggregation of the predictions with the mean cluster values was made in order to discover if the models have been able to discriminate some interesting facts inside the data concerning the different reviews and the alcoholic beer contents.
From the insights obtained by this analysis on the three features *beer_ABV*, *review_taste* and *review_overall (tab.3)*, it is possible to observe that for what K-means is concerned, the values corresponding to the two clusters means show a distance of 1 value point, and considering that almost all the reviews are contained in a range from 1 to 5 and the beer

content level for the majority of the beers inside the dataset is below 10, the data division among the two cluster stand for a "qualitative" difference of the elements contained inside them; for what Bisecting K-means is concerned, we have a difference of almost 0.5 value point for the 4 different clusters in all the three features, denoting a reasonable division of data point but surely less divisive and solid than the one performed by the K-means algorithm.
For these characteristics and the ones already discussed in the two previous subchapters, it is possible to conclude that the K-means algorithm with the k parameter set to 2 is a preferable setting rather than the Bisecting one for a clustering task on this dataset.

beer_ABV

Prediction	Mean Cluster
1	5.493069597069607
0	6.509032818801233

beer_ABV

Prediction	Mean Cluster
1	5.573957477651627
3	6.957799812617074
2	6.148804211581811
0	5.239629068887209

review_taste

Prediction	Mean Cluster
1	2.91174908424907
0	3.81653594771238

review_taste

Prediction	Mean Cluster
1	3.09791978738825
3	4.06212991880074
2	3.61939583855603
0	2.32843300529901

review_overall

Prediction	Mean Cluster
1	2.93601282051281
0	3.91516200806561

review_overall

Prediction	Mean Cluster
1	3.154102440202942
3	4.16075733916302
2	3.71802080721987
0	2.25268735806207

Tab 3. Mean clusters corresponding values for three different dataset features done for the K-means (on the left) and the Bisecting K-means (on the right).

Classification

After the clustering phase the work proceeded with the classification phase, that was realized over a specifically predefined dataset containing the beer names, the vectorization of the features *beer_ABV, review_appearance, review_palette, review_taste* and the *review_aroma* and the classification target. All these features and the target ones are

computed or retrieved by the aggregated dataset (see Data Understanding chapter).
For the classification task two target variable were created:

1. *review_bin*: this feature was realized for the binary classification and was obtained binarizing the *review_overall* feature. The binarization was realized through the binarizer method made available by pyspark.ml.feature. The method requires the setting of a threshold to split the continuous values, the input column and the label to assign for the second range. The output of this method is a new column with the discretized values of the given columns.
2. *review_target*: this feature was realized for the multiclass classification and was obtained rounding the *review_overall* feature.

As threshold for the *review_bin* generation a value of 2.5 was chosen, in order to obtain a value of 0 if the value of *review_overall* was less than 2.5, a value of 1 otherwise.
The tools exploited to realize the parameter tuning for the classification task and the train-test split (ParamGridBuilder, TrainValidationSplit) were imported from pyspark.ml.tuning library, while the tools exploited in order to get some goodness measures of the various classifications (MulticlassClassificationEvaluator, BinaryClassificationEvaluator) were imported from the py.spark.ml.evaluation library; finally, the classifier used to realize the different classifications implemented (RandomForestClassifier, LogisticRegression, GBTClassifier, NaiveBayes) were imported from the pyspark.ml.classification library.
Both multiclass classification and binary classification tasks were accomplished. Before the application of each algorithm a grid search was realized using TrainValidationSplit in order to tune the different models among the different combinations of training and validation dataset according to their best value of accuracy for the classifier implemented for the multiclass classification task, and according to their best value of area under the curve for the classifier implemented for the binary classification task (because of the non-availability of the accuracy measure).
After the classification processes, two different evaluation phases were made depending on the classification task type: for the binary classifications a binary evaluator was applied, whereas for the multiclass classification a multiclass evaluator was applied, both allowing the comparison between different accuracy results from the different classifiers applied.

Multiclass Classification

The multi classification task was realized using *review_target* as target variable and two different classifiers were used: the random forest and the logistic regression.
For both these models TrainTestSplit was used in order to select and apply to the model the best combination of parameters values to use with the Test data, according to the best achievable accuracy.
For what random forest is concerned, the following tab shows the results obtained in terms of accuracy and F1 measure for the TrainTestSplit chosen model parameters values.

Random Forest

Parameter to set	Candidate values	TrainTestSplit chosen ones	Model accuracy	Model Error	F1 score
numTrees	5, 10, 15	10	0.82	0,18	0.8
maxDepth	5, 10	5			
impurity	Gini, Entropy	Entropy			

Tab 4. Random Forest parameter setting choices and reached performances.

For what logistic regression is concerned, the following tab shows the results obtained in terms of accuracy and F1 measure for the TrainTestSplit chosen model parameters values.

Logistic Regression

Parameter to set	Candidate values	TrainTestSplit chosen ones	Model accuracy	Model Error	F1 score
regParam	0.0, 0.05, 0.1, 0.2, 0.5	0.0	0.82	0,18	0.8
threshold	0.3, 0.5, 0.7	0.3			
maxIter	50,100	100			

Tab 5. Logistic Regression parameter setting choices and reached performances.

Is it possible to notice that both the classifiers reach optimal performances in terms of accuracy, reaching the same results, so they both seem to satisfy the multi classification final goal.

Binary Classification

The binary classification task was realized using *review_bin* as target variable and two different classifiers were used: the gradient boosting tree and the naive bayes.
For the binary classification, TrainTestSplit was used only for the naive bayes parameter tuning since the implementation on the gradient boosting tree classifier ended up to be too expensive in terms of computation time. As an evaluation metric for the TrainTestSplit the area under the curve was used, but the final evaluation was realized through the accuracy measure in order to be comparable with the GBT classifier final results.
For what gradient boosting tree is concerned, the parameter tuning has been done via two for loops iterating through the wanted-to-test values for maxDepth and minInstancesPerNode.
The following tab shows the results obtained in terms of accuracy and F1 measure for the chosen model parameters values.

Gradient Boosting Tree					
Parameter to set	Candidate values	TrainTestSplit chosen ones	Model accuracy	Model Error	F1 score
maxDepth	5, 10	5	0.97	0,03	0.97
minInstancesPerNode	1, 2, 5, 10	5			

Tab 6. GBT parameter setting choices and reached performances.

For what naive bayes is concerned, the following tab shows the results obtained in terms of accuracy and F1 measure for the TrainTestSplit chosen model parameters values.

Naive Bayes					
Parameter to set	Candidate values	TrainTestSplit chosen ones	Model accuracy	Model Error	F1 score
modelType	multinomial, gaussian	gaussian	0.94	0,06	0.95
smoothing	1.0, 1.5, 2.0, 3.0, 5.0	1.0			

Tab 7. Naive Bayes parameter setting choices and reached performances.

Both the classifiers do reach very high performances in terms of accuracy, with the gradient boosting tree reaching the highest results yet achieved by any classifier: they both fulfill the binary classification final goal but the best one for this task results to be the gradient boosting tree classifier.

NLP-based Binary Classification

Eventually, a last binary classification task was implemented exploiting a sentiment analysis implementation provided by the sparkNLP library, with the goal of extracting the sentiment (labeled as *positi*ve or negative) that emerges from the users textual reviews contained inside the review_text feature.
A first textual data preparation preceded the classification phase, with the aim of making the data more suitable for the sentiment classification task. This phase consisted in the application of the DocumentAssembler(), Tokenizer() and Normalizer() methods, which allowed to format, tokenize and clean the tokenized sentences from dirty characters following a regex pattern and transform words based on a provided dictionary.
The ViveknSentimentModel was used in order to implement the sentiment analysis, which is a Naive Bayes classifier specifically suitable for this task. The model was fed with the previous preprocessed textual output and returned a new feature, called *result_sentiment*, containing the sentiment annotation for each row; after another processing step, through the Finisher() method that converted output annotation into strings corresponding to the desired labels, the ultimate feature called *final_sentiment* was generated and ended up containing the desired positive and negative labels.

In order to evaluate and compare the results obtained through sentiment analysis an additional artificial feature called *sentiment* was added to the original dataset, which contains a binarization of the *review_overall* feature under the condition to return "*negative*" if the review has a score less than 4[2], "*positive*" otherwise.
The sentiment analysis model performances result to be not optimal despite the attempt to meet the negative bias of the sentiment analysis by discretizing *review_overall* with the threshold 4 instead of 2.5 as done for the previous classification tasks, and does not seem to bring any eventual gain compared to the use of the various review features already present in the dataset.

Sentiment feature

Sentiment	Count
Positive	338500
Negative	190370

Sentiment Analysis predictions

Sentiment	Count
Positive	211896
Negative	316688
Not Classified	167

Tab 8. Positive and negative labels obtained from the *review_overall* discretization and from the sentiment analysis results.

Despite the attempt to balance the "negative" behavior of the sentiment classifier, the results obtained are completely unbalanced, with vastly more negatively labeled reviews for sentiment rather than binarization.
From this it's possible to deduce that the hypothetical use of sentiment labels instead of those pre-supplied through review scores is not an optimal way and it is preferable to proceed through scores.

[2] This value of threshold was chosen because of a natural tendency of reviews to contain negative words in not very high review scores, and this hugely affects the sentiment analysis results in creating a bias towards negative inferences. For this reason, in order to make the comparisons between the sentiment driven classification results and *review_overall* "fair", it was decided to consider "negative reviews" all the *review_overall* values below 4.