Wine Reviews

Gauthier Cler
gauthier.cler@gmail.com

Anas Buyumad
anas.buyumad@epitech.eu

February 2, 2020

Abstract

This paper defines and explains through chronological steps the approach adopted during this data science and visualisation project. We justify our implementation choices, library uses and we interpret obtained results and performances. Finally we suggest areas of improvement on current baseline and relate on dataset applicability.

Contents

1 Introduction

Wine is one of the most popular drink in the world. Every wine is unique and there exist a lot of different varieties. But when the time comes where you have to choose which wine to drink, you get lost and you start wondering if there exist any way you could find the right wine, or if you even know what exactly makes a great wine.

The goal of this project is to propose visualisations of a wine reviews dataset along with a quantitative analysis.

Project source code, project description and all additional resources are available at

`https://github.com/gauthiercler/wine-reviews`

2 Approach

We define our approach toward this problem in distinct steps, detailed in the following sections.

2.1 Dataset selection

Among several millions of different datasets of many kinds available online, we decided to go with **wine-reviews**, a popular dataset available on Kaggle, an online community Data Science platform.

`https://www.kaggle.com/zynicide/wine-reviews`

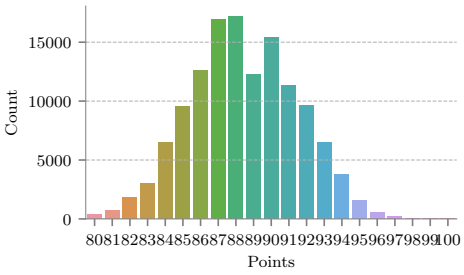
We will be using **winemag-data_first150k.csv** file as our input dataset file. It is about 130,000 rows of data, shared across 10 columns (plus 1 column for id). Each entry represents a detailed review on a wine from an user of `www.wineenthusiast.com` website.

Dataset Description		
Column name	Data type	Missing values
country	categorical	5
description	text	0
designation	categorical	45735
points	numeric	0
price	numeric	13695
province	categorical	5
region_1	categorical	25060
region_2	categorical	89977
variety	categorical	0
winery	categorical	0

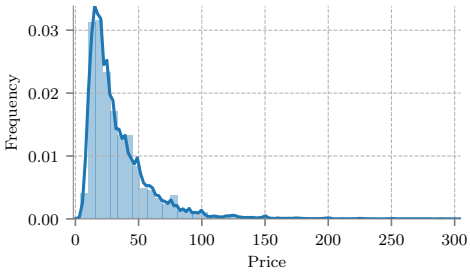
One of the reasons for choosing this dataset is the large context diversity on the different columns. The description field is quite long and text analysis could be applied to extract relevant features from raw text. However, we can see that some fields such as region_1 and region_2 contain many missing values and have to be handled correctly. We will go through a deeper analysis on the data distribution and possible columns correlations in the next section.

2.2 Exploratory data analysis

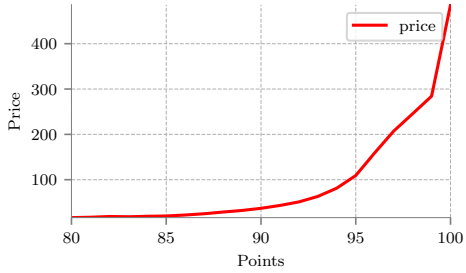
After describing dataset structure in the last section, let's perform an exploratory analysis on columns and try to find possible correlations between fields. In this dataset, only two columns are of numeric type. Using matplotlib and seaborn visualisation libraries, we can plot distribution. *Points* can be defined as discrete data $\mathbb{D} = \{80, 81, 82, \ldots, 100\}$, while *Price* is continuous. We can see that *Price* frequency is roughly centered on $x \in [0, 100]$ (Figure 1b).



(a) Points count



(b) Price frequency



(c) Price distribution on points

Figure 1: Numerical data fields distribution

By analyzing the distribution of our data, we can observe that the points of the wines follow a normal distribution (figure 1a) and that their price follow a log-normal distribution (figure 1b). By plotting the evolution of a wine's price over its attributed points (figure 1c), we can recognize an exponential distribution, indicating a correlation between a wine's quality and price. We can thus validate the trivial assumption that the better the wine, the pricier it is.

For non-numerical data, it is not mathematically possible to detect correlations. We can however apply wine-related general knowledge in order to get the most out of it.

For example, we know for a fact that France is the top wine region in the world, while red wines tend to be more expensive than white wines. [1] [2]

Though this kind of information alone is not very pertinent, when coupled with wine-specific characteristics contained in the description field we can gain valuable insight on the wine we are trying to analyze (Figure 2). One such insight would be a wine qualified as having great quality tannins, meaning a higher quality wine, which in turn means a higher price especially if it is a French red wine.

Thus, it is in our interest to process information such as origin (Figure 3), variety or description because an enormous amount of correlations can be found and deduced.

Figure 2: Most frequent words in description field

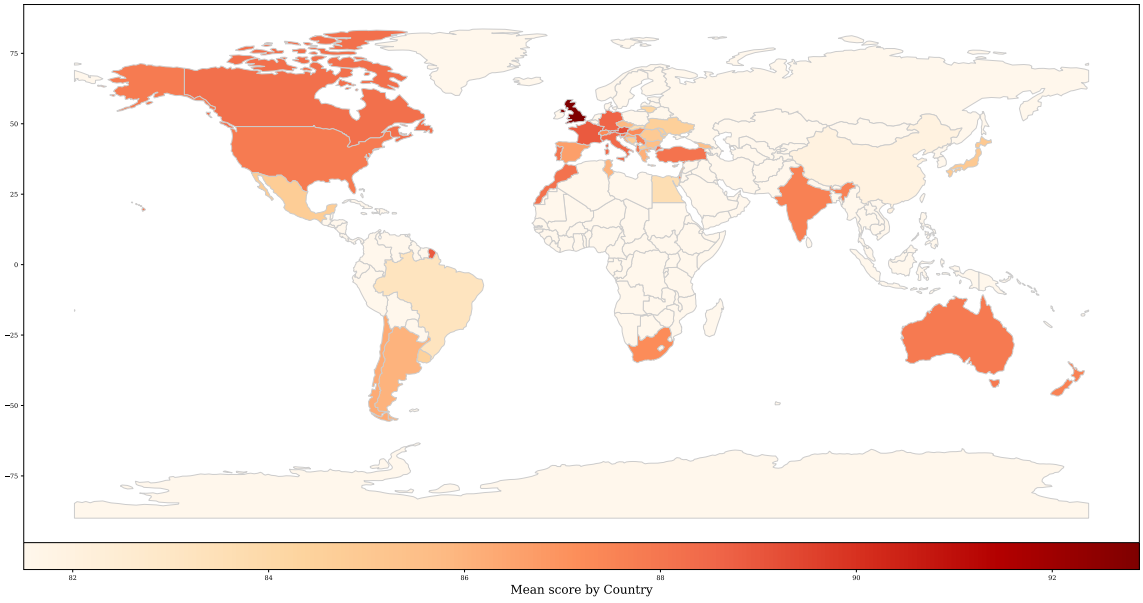


Figure 3: Mean Score by country map

2.3 Data preprocessing

Before training and testing a model on our dataset, we must pre-process its data according to our analysis in order to get the best predictions possible.

Firstly, we randomly sort our dataset and remove duplicate entries.

```
X = wines.sample(frac=1).reset_index(drop=True)
X.drop_duplicates(inplace=True)
```

We then choose to drop certain columns for different reasons:

- *Designation* is dropped because 29% of its entries are null values, with almost 40 000 unique values. Knowing the vineyard where the grapes were used for the wine is too specific and would not give us any useful insight especially when contemplating the training time cost along with the sheer number of potential features.
- *Winery* is dropped for the same reasons as designation, 16 757 unique values is too specific, and knowing in which winery the vineyard is does not add much to quality prediction.
- *Region_1* is dropped because 16% of its entries are null values and 1 228 unique values giving the place inside a province or state where the wine was grown is too costly all the while giving very low benefits to prediction accuracy. We already possess the *Province* feature giving us a broader and more accurate insight.
- *Region_2* is dropped for the same reasons as *Region_1*, being just a more precise version of it, boasting 61% of null values.

Finally, we simply remove any wine that possesses one or more feature with a null value.

```
X.drop(['designation', 'winery', 'region_1', 'region_2'], inplace=True, axis=1)
X.dropna(axis=0, inplace=True)
```

In order to gain worthwhile information from our wine's Description, we first need to reduce its complexity. We use a count vectorizer to sort the words (mainly adjectives) by their amount of occurrences throughout the dataset, allowing us to extract and keep meaningful key-words with the best potential of characterizing our wines (1000 words in our case). English stop words are also used to reject common words.

We end up with a two dimensional array, representing the weight of each key-word according to each wine, that we merge to the original dataset while dropping the description column.

```
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import hstack

vectorizer = CountVectorizer(stop_words='english')
result = vectorizer.fit_transform(X['description'])
feature_array = vectorizer.get_feature_names()
top_words = sorted(list(zip(vectorizer.get_feature_names(),
                                        result.sum(0).getA1())),
                           key=lambda x: x[1], reverse=True)[:1000]

indexes = [feature_array.index(x[0]) for x in top_words]
result = result[:, indexes]
X_merged = hstack((X, result))
```

For the rest of the data (Country, Province and Variety), we use a one-hot encoding method in order to numerically represent the values as binary. Meaning, in the case of the Country for example, 44 different features with only one having a value set to 1, stored as a sparse matrix in order to save space. This effectively

allows us to tell our model where a wine comes from or its variety, getting rid of the nominal nature of this information.

Before doing so, we also isolate the Points from the dataset.

```
from sklearn.preprocessing import OneHotEncoder

y = X['points']
X.drop('points', axis=1, inplace=True)
encoder = OneHotEncoder()
X = encoder.fit_transform(X)
```

Finally, we split our processed data into two group, train and test.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test.split(X, y, test_size=0.1, shuffle=True)
```

We define 90% as **train** set and 10% as **test** set. The goal here is to train the model on the train test, so it can generalize and create an intuition on the data. We can then validate using the test set to evaluate model understanding and reliability.

2.4 Model selection

When it comes to model selection, it's not always easy to find an estimator exactly suited for a given problem and data. In our case, we want to predict wine quality (column points) from other features. As we saw in EDA section, points are distributed from 80 to 100, thus we are dealing with a regression problem. Note that because we are working with discrete distribution, we can also define this problem as a classification problem with 20 categories.

Only two columns are numerical while the rest are categorical and One Hot Encoded.

We decide to approach the problem using a simple decision tree, as conditional splitting can be relevant to categorize wines.

```
from sklearn.tree import DecisionTreeRegressor

clf = DecisionTreeRegressor(max_depth=10)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

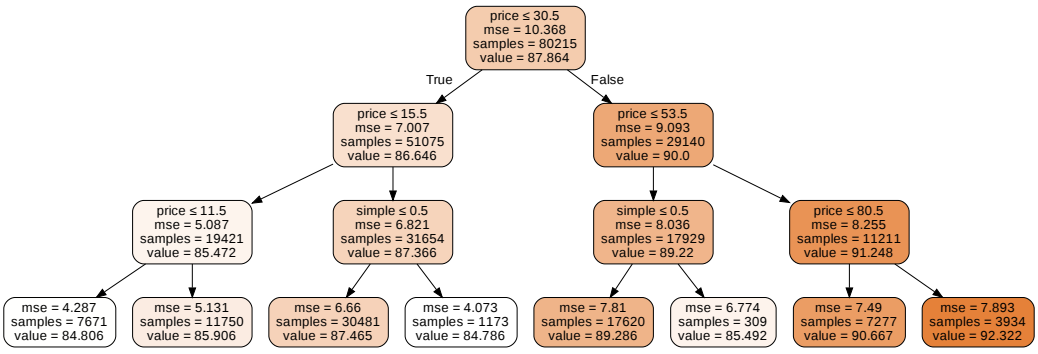


Figure 4: Subset of a decision tree with depth of 3

On figure 4, we can see an example of how a decision tree splits its nodes and create intuition on numerical data.

However, we encountered a problem during training when not limiting the max depth of the tree and consequently increasing the complexity and time of training. In fact, using OneHotEncoding with a decision tree is likely to make it grow in one direction and produce a very sparse tree. Because we define a matrix of occurrences/non-occurrences of terms in the dataset, the tree will perform binary conditions for each feature. This can lead to inconvenient cases (see Figure 5).

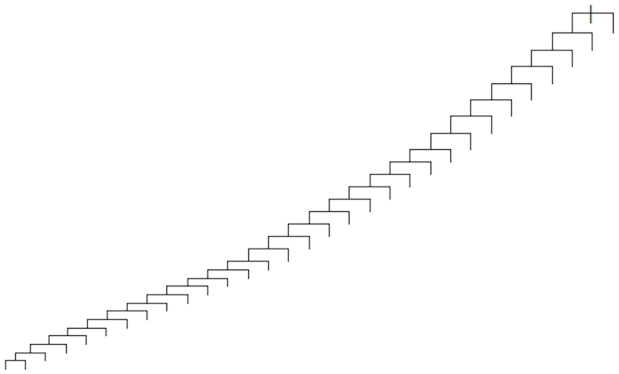


Figure 5: Decision tree using one hot encoding

To solve this issue, we decided to switch to a gradient boosting based model. Instead of going deeper in our decision tree architecture, we apply Boosting to train an ensemble of decision trees. The Boosting method is based on training and converting weak learners to strong learners. We can see on Figure 6 how a Boosting based classifier is able to converge with simple decision trees.

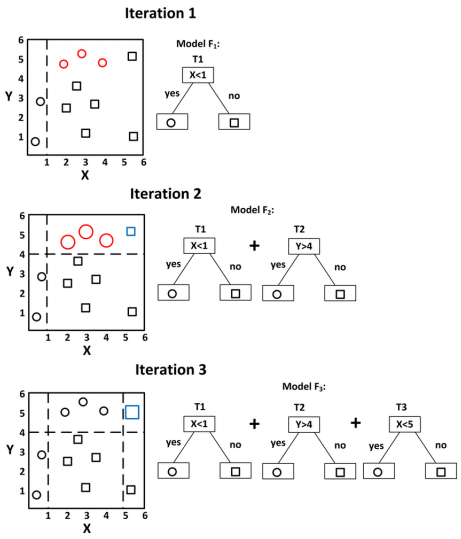


Figure 6: Gradient Boosting mechanic

The way we are evaluating the model is defined in the next section.

2.5 Model evaluation

Many metrics and methods are available to evaluate how well a regression model is performing. To evaluate the quality of our prediction, we will use several indicators. The Gradient Boosting default scoring function is the use of the coefficient of determination R^2.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{1}$$

It is defined as proportion of the variance in the dependent variable which is predictable from the independent variable. [4]. Our goal is to approach as much as possible to an R^2 value of 1, which would mean that the predictions perfectly fit the ground truth data.

We can also measure the model efficiency with other metrics such as:

- **M**ean **S**quare **E**rror is a risk function, corresponding to the expected value of the squared error loss. Near 0 is better

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \hat{y}_i)^2. \tag{2}$$

- The explained variance measures the proportion to which the model accounts for the dispersion of the data. Try to maximize until 1.

$$explained\_variance(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}} \tag{3}$$

In order to reduce model loss, we will tune hyper parameters of current model to find the best parameters combination.

2.6 Model tuning

Now that we have selected our model, we can start tuning it without overfitting or creating too much variance.

Tuning is the process of maximizing a model's performance. We can do so by choosing appropriate hyperparameters, which are parameters that our model does not learn automatically and instead need to be set manually.

Among the many existing tuning method, we opted for the Grid Search, which is a simple method where one needs to define combinations of hyperparameters. Each combination will then be evaluated using cross-validation.

We started tuning the number of trees and the learning rate of our model, both being closely related and having a big effect on the computational cost.

The $n_{estimators}$ will define the number of sequential trees to be modeled.

The *learning_rate* will determine the impact of each tree on the final outcome. Lower values are usually preferred as they make the model robust to specific characteristics of the tree.

A lower learning rate will require a higher number of trees and be computationally more expensive. It is admitted that a 10-fold decrease in the learning rate should be met with a 10-fold increase in the number of trees.

One example of tuning the $n_{estimators}$ and the *learning_rate* would be:

```
hyperparameters = {'n_estimators':[100,250,500,750,1000,1250,1500,1750],
↪  'learning_rate':[0.15,0.1,0.05,0.01,0.005,0.001]}

tuning = GridSearchCV(estimator = GradientBoostingRegressor(subsample=1,
↪  min_samples_split=2, min_samples_leaf=1, max_depth=3, random_state=10,
↪  max_features='sqrt'), param_grid=hyperparameters, scoring='r2', n_jobs=-1, iid=False,
↪  verbose=999, cv=5)
tuning.fit(X_train, y_train)
```

Once we get the best combination, we can integrate it into our model and start tuning other hyperparameters.

The *subsample* will be the fraction of observations to be selected for each tree, values slightly less than 1 make the model robust by reducing the variance.

Now that our boosting hyperparameters are tuned, we can start tuning tree-specific hyperparameters.

The *min_samples_split* will define the minimum number of samples which are required in a node for it to be considered for splitting. It is used to control over-fitting.

The *min_samples_leaf* will define the minimum samples required in a terminal node or leaf. Akin to *min_samples_split*, it is used to control over-fitting.

The *max_depth* is the maximum depth of a tree. It also controls over-fitting as a higher depth will allow our model to learn relations specific to a sample.

The $max\_features$ is the amount of features used while searching for a node split. Using the square root of the total amount of features is usually advised.

2.7 Possible improvements

We took a first approach on this dataset, but we can easily think about many possible improvements about our model and the way we analyze and prepare the data. For example:

- Part-of-speech tagging: Marking up words to its corresponding part of speech, based on both its definition and its context
- Semantic analysis: Analyze how related words are to each other in the *Description* field and be able to understand proper sentences.
- Feature Engineering: Extract and generate new features from actual columns. For example extract the year of the wine from the *Title* field (available in winemag-data-130k-v2.csv).

3 Wine Recommender

Based on the dataset analysis and wine review descriptions, we made an online tool to recommend wines. By defining your maximum price, filtering by country and/or wine variety, while describing your ideal wine with keywords such as "soft, "fruity" or "dry", this tool will be able to recommend you a list of wines that would match your needs. It is available at:

`http://wine-recommender.gauthiercler.com/`

4 Conclusion

We proved throughout this document that this wine review dataset is applicable for a regression problem to predict wine quality from textual and numerical information. Even if we didn't use many complex analysis methods or Natural Language Processing techniques, we can state that our implemented model is consistent and accurate towards real and future data. Even if this dataset is partialy incomplete, we managed to extract relevant deterministic features.

References

[1] Vivino: How much does a good bottle of wine cost
`https://www.vivino.com/wine-news/how-much-does-a-good-bottle-of-wine-cost`

[2] Winefolly: Top wine regions of the world
`https://winefolly.com/lifestyle/top-wine-regions-of-the-world/`

[3] A simple example of visualizing gradient boosting.
`https://www.researchgate.net/figure/A-simple-example-of-visualizing-gradient-boosting_fig5_326379229`

[4] Wikipedia: Coefficient of determination
`https://en.wikipedia.org/wiki/Coefficient_of_determination`