COPENHAGEN BUSINESS SCHOOL

BUSINESS DATA PROCESSING AND BUSINESS INTELLIGENCE

Veritas in Vino

Applying machine learning in the prediction of extraordinary wines

Author:

Christian Stolborg

Abstract

The wine industry is in rapid growth, with revenues expected to increase from $326b to $434b from 2020 to 2027. Despite this, procurement in the industry is largely based on the subjective opinions of the procurer. This paper challenges the status quo and recommends a machine learning-driven procurement strategy based on predictive models trained on the quantitative data of the online wine community platform Vivino.com. The paper finds that an XGBoost produces the best out-of-sample results, achieving R^2 values of up to 86%. In an attempt to leverage the qualitative data of the Vivino platform, natural language processing models are implemented to predict whether or not a new vintage will become very popular. Based on 5 different classifiers, the logistic classifier has the highest out-of-sample precision of 62%. In an attempt to improve out of sample precision, the XGBoost and Logistic models, are combined which increases the precision of the model to 80%. Following the finalization of the analyses, source code and models are deployed to https://www.dsba-vinoveritas.com/, where the reader is strongly encouraged to try out the online wine rating predictor, which is derived from this paper.

Keywords: Oenology, Wine, Machine Learning, NLP, Sommelier, Predictive Modelling, Wine Characteristics, Red Wine, Inferential Statistics.

Pages: 11 Characters w. spaces: 68,250 30. November 2020

TABLE OF CONTENTS

1 Introduction

Topic Importance Relevance Motivation Case Company Introduction Problem Formulation Research Question(s) Delimitations Advanced Organizer

2 Conceptual Framework

Explain the theory behind CRISP-DM and Datawarhouses, which are the primary conecptual models used in this paper.

2.1 CRISP-DM

Something on CRISP-DM

2.2 Data Warehouse using the Star-Schema

Something ON DW

2.3 Evenutal Only if space: Machine Learning modelleing

Something on how modelleing was done.

3 METHOD

In this section it is outlined how the data is collected, stored and used for analysis. The process is illustrated in the flow chart ins Fig. 3.0.1. Initially, the Vivino website is web-scraped using Python and the packages Selenium, BeautifulSoup4 and Requests. The data is then stored in a No-Sql database **accounting for its horizontal structure**. In preparing the data for analysis, relations are constructed and the data is transformed and loaded into an SQL database. Then preprocessing is performed to account for missing values and errors as well as to output datasets that are ready for **subject-oriented** modelling. Finally, a visual analysis is done in the Microsoft Power BI service, regular machine learning models are trained using intrinsic vintage features and natural language processing is performed on vintage reviews.

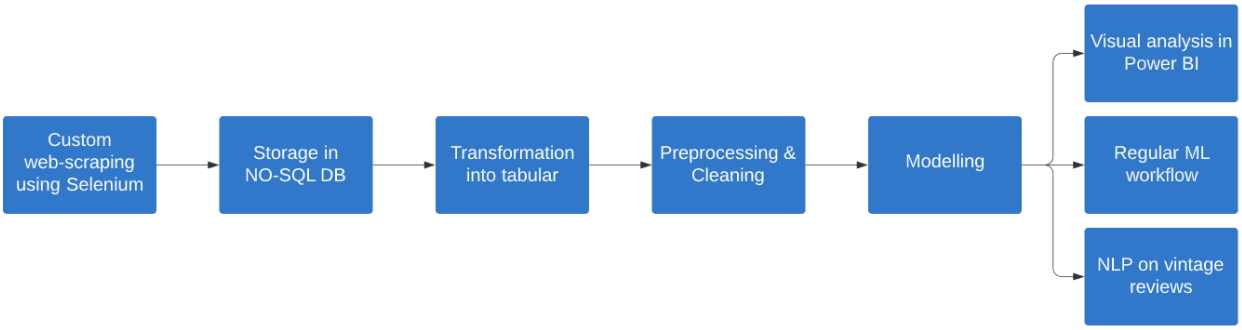

Figure 3.0.1: Process Diagram. Since we don't have control over how the Vivino database evolves the data is intially stored in No-Sql and then transformed into relational format. This ensures potential future errors in the data fetch will arise internally in the system, rather than externally.

3.1 DATA COLLECTION: METHODS AND TOOLS

As mentioned, the analysis will be based off the Vivino.com website. Vivino is both an online wine marketplace as well as a community for wine enthusiasts. It is most readily comparable to a social network, in which the users can rate vintages from 1-5 stars, write reviews and follow each others profiles. As a result, Vivino hosts an array of information about its' vintages such as price, country of production, used grapes etc. but also a large amount of subjective attributes such as average ratings, average pct. sweetness/boldness in taste and user reviews.

Unfortunately, Vivino does not have an open API and therefore a custom crawler is developed in Python to acquire their data. The crawler ran from 31-10-2020 to 12-11-2020. Since some attributes has the need for horizontal scalability and, more importantly, because we don't control how the Vivino database will evolve in the future, the data is initially stored as JSON files. See Appendix A for an example. This will provide for a more flexible structure that can easily handle changes in the underlying attributes. As the data is highly hierarchical it is stored in 7 different JSON files, with appropriate keys to connect all files together. Concretely, each dataset contains inputs with its own key as well as a foreign key to the level one up. **Maybe something on how files are updated; Using temporary dicts to update a permanent one**. In conclusion, the crawler yields data sets concerning vintages (an instance of a wine), wines, wineries, regions and countries. Finally, the last two data sets concern information about reviews and users. The vintages dataset display information about each vintage such as price, rating, year. It is the product the users on Vivino.com can actually purchase. The wine dataset then holds aggregate information about vintages that has the same wine name, e.g. which grapes were used, how are the taste features such as level of sweetness/boldness etc. The

wineries data contain information about each winery such as the winery's average rating, its geographical location, number of times it has been reviewed and so on. Regions and country datasets are primarily geographical information, even though it also holds information such as most popular grapes and the amount of users per region. Reviews dataset is simply information on each review and the users dataset contains a users number of followers, number of reviews given and more.

TABULARIZING THE DATA USING ER MODELLING

The No-Sql storage made sense from an operational viewpoint as the high degree of flexibility results in a **lower probability of having to update storage rules later making the storage more future-proof**. However, for analysis purposes it is not suitable and all data described above are transformed into a tabular Sql database. The transformation is implemented by using the entity relational model in which each level in the data hierarchy represent an entity. As such, each entity will contain information related to the level of the hierarchy it represents.

Other models were also considered such as more traditional data warehousing techniques like the star or snowflake schemas. However, these approaches were not chosen even though they work very well with highly hierarchical data. The reason for this is grounded in the fact that due to the web-scraping dependency, at no point in time will it be certain that the full Vivino database is stored. As a result, when computing important aggregated metrics such as average winery ratings this would in some cases be based on an incomplete amount of ratings, making it deviate from its true value. However, Vivino.com offers these aggregated metrics themselves meaning the analysis will still be able to incorporate their true values rather than an approximation. The relations are shown in Fig. 3.1.1.

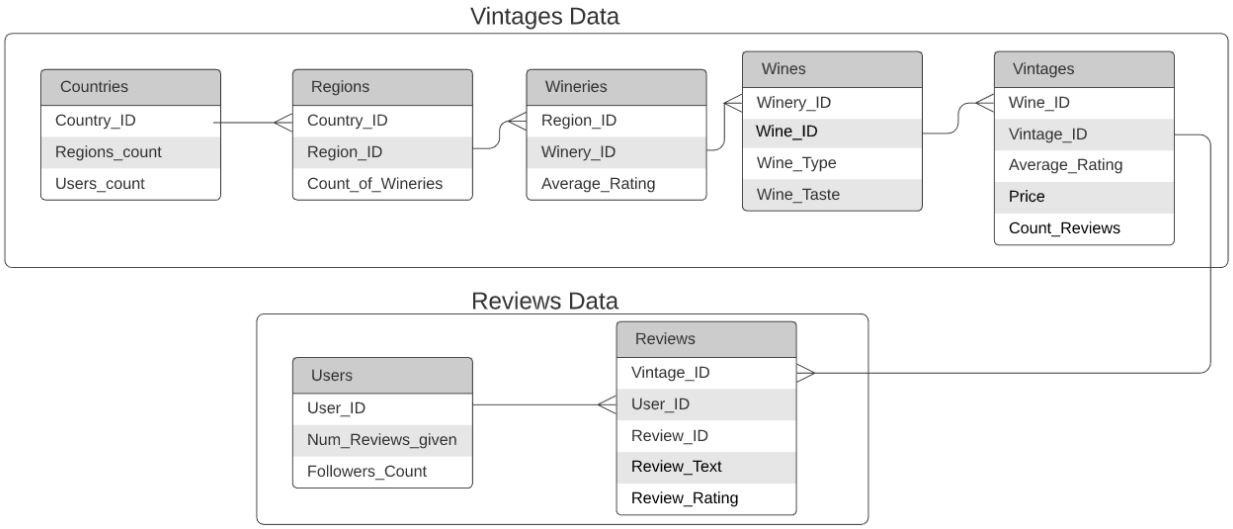

Figure 3.1.1: Data structure. The 7 data sets are aggregated into tabular form using the relationships shown. The final result is a data set working on vintage level and one working on review level. Lines show cardinality. Attributes shown for each data set are non-exhaustive.

From the relations in Fig. 3.1.1, the data can be easily aggregated from the 7 tables shown, into two tables that will be used for analysis. Note, that whilst some of the JSON files only contain simple key:value pairs, they are easily converted to tabular, however, others contain more complex relationships with several layers. An example of the No-Sql storage is shown in Appendix A. Note that, attributes with several levels such as *foods* are one-hot encoded to

account for the presence or non-presence of e.g. beef, and attributes *taste structure* and *flavor* are simply expanded into several columns thus unfolding their inherent key:value pairs. **Perhaps something on how this is future-proof is more sub-attributes are added to the site or changes are made**.

In Conclusion, the data has been transformed in a way that it can now be easily aggregated. In the coming sections, the data will be explained from two views, a vintage centric view, and a reviews centric view. It should be noted that aggregating the data into one rather than two sets were also entirely possible, however, given the computational requirements of running certain NLP models and due to time constraints in training, the data is separated to allow **easier parallel processes**.

3.2 Data Description at Column Level & Preprocessing

A basic description of a selected subset of attributes is given in Table 3.2.1 along with descriptive statistics. This is merely an extract as there are 55 features in total. From features such as *price*, it is noted that the data is quite noisy with unrealistic prices at 0, as well as a standard deviation of ~ 8 times the mean. Furthermore, some variables contain a large number of NULL values. For now, the missing values are dropped reducing the vintages dataset to 48,452, however, many of these missing values were caused due to problems with the web crawler and future research should incorporate these into their analyses. Furthermore, errors such as strings in integer variables are removed. Filtering is applied to *year* and *price*, removing vintages produced earlier than 1980 and only including prices from 45DKK to 5,000DKK, since wines outside this range is bought at too low frequencies and are not of interest in the particular business case. Additionally, the top 1 and bottom 1 percentiles are removed to account for outliers. This threshold is set very conservatively as to not risk removing extreme values as these, contrary to outliers, are expected to be predictable. For further discussions on extreme values versus outliers see Nystrup (2018). All these implementations are performed in the Python library Pandas[1].

Even though imputing strategies and feature engineering are not applied in this analysis, it could be a highly relevant strategy to apply in future uses with a heavier emphasis on building machine learning models. Additionally, the data is shown to non-linear making Borup et. al's (2020) methods on linear variable selection as preparation step for non-linear learning with e,g. LASSO models an attractive preprocessing step. In essence, it is a simple technique that entails tuning the C, or equivalently, the λ parameter in a LASSO regression to tune the number of non-zero $\hat{\beta}^{\lambda}$ variables, i.e. features. In python, this step can simply be added to the ML workflow using *sklearn's* Pipeline functionality and *GridSearchCV* to tune for the optimal number of features.

3.3 Data Pre-Processing: Methods, Tools and Techniques (For example using SQL, RDBMS)

To be deleted. Data Filtering, Data Discarded, Transformation, and Combination

[1] For source code see the self-contained notebook in the associated Github.

Table 3.2.1: Descriptive statistics of a selected subset of features. There is a total of 55 different variables across all datasets. The reviews dataset is much larger since one vintage can have many reviews.

Feature	SQL Data Type	Count	Mean	Std	Min	Max	Description
Panel A: Vintages dataset							
Average rating	FLOAT(24)	81,169	3.6	0.8	1.0	5.0	Avg. vintage rating from 1-5
Price (DKK)	FLOAT(24)	81,169	324.0	2,724.2	0.0	287,626.5	Retail price in DKK
Rating count	INT(255)	81,169	840.9	2,549.7	0.0	122,412.0	No. of vintage ratings
Review count	INT(255)	81,169	188.7	372.5	0.0	22,669.0	No. of vintage reviews
Winery rating count	INT(255)	81,169	51,328.3	72,580.1	21.0	535,578.0	No. of winery ratings
Winery average rating	FLOAT(24)	81,169	3.8	0.3	2.4	4.7	Avg. winery rating from 1-5
Wine Type	CHAR(9)	48,452	n.a.	n.a.	n.a.	n.a.	Red, white, sparkling etc.
Country	VARCHAR(100)	81,169	n.a.	n.a.	n.a.	n.a.	Country of production
Year	INT(255)	81,169	n.a.	n.a.	n.a.	n.a.	Year of production
Panel B: Reviews dataset							
Review	VARCHAR(500)	281,781	n.a.	n.a.	n.a.	n.a.	Review of vintage
Rating	FLOAT(24)	281,781	n.a.	n.a.	n.a.	n.a.	Rating $\in 1, 1.5...4.5, 5.0$
created_at	DATETIME	281,781	n.a.	n.a.	n.a.	n.a.	Datetime of review

3.4 Data Analytics: Modelling, Methods and Tools (For example using Tableau/Power BI)

On the basis of the measures described in previous sections, the data is made ready for analytics. An explorative data analysis is performed with Python in Jupyter Notebooks[2] as well as with Power BI, as it allows the user to interactively slice and subset many variables across the hierarchy to test for relationships. As mentioned, the preprocessing were done using Python and therefore loading the data into Power BI is relatively simple. It merely requires setting up the relations as shown in Fig. 3.1.1, and defining the data types on some variables. The specific dashboard will be explained in the *Results* section.

For the particular business case, the dashboard and exploratory data analyses shall merely be seen as a means of providing an overview of the data. What is really interesting is the predictability of vintage ratings. As a result, several machine learning models are build and trained on the vintages dataset. The only additional modelling that is not yet described, is standard scaling, $z_i = \frac{x_i - \mu}{\sigma}$, which is applied to all continuous variables. The dataset is split in an 80:20 train/test sets. On this basis, a linear regression, elastic net, random forest, decision tree and an ANN are trained and compared.

Finally, a natural language processing framework is applied to the reviews dataset to uncover whether reviews can be used as predictor's for vintage ratings. Since the primary scope will be on business intelligence in Power Bi, an in depth explanation of NLP processes is not provided here. To make reviews ready for modelling, all punctuations, emojis, emoticons and stop words are removed. The remaining text is then tokenized and the **lemma of each word is taken**. From here, a dictionary is made and filtered to only include the top 1000 most popular words. Then a bag of words representation is created, from which a count-based and a TF-IDF vectorizer are built as

[2]For source code see the self-contained notebook in the associated Github.

$$w_{i,j} = tf_{i,j} * log(\frac{N}{df_i}) \tag{3.4.1}$$

where, $w_{i,j}$ is the TF-IDF score for a term i in a document j, $tf_{i,j}$ is the number of occurences of term i in document j, N is the total number of documents in the corpus and df_i is number of documents with term i (Salton et al., 1986). Several models can be trained on the TF-IDF set to provide us with results. The workflow is described in Fig. 3.4.1 and is implemented using Pythons packages *Spacy, NLTK & Gensim*. For a deeper explanation of model choices see the self-contained notebooks on the github.

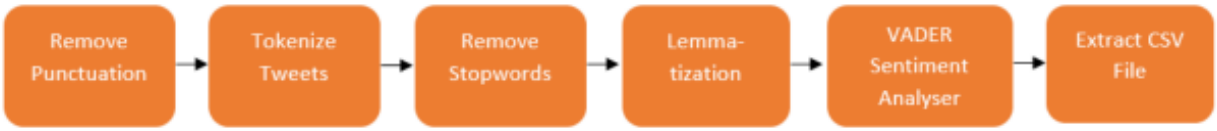

Figure 3.4.1: Natural language processing workflow for vintage reviews.

4 Results

In the previous section, the methodology on gathering, storing and transforming the data were explained, as well as a brief overview of the analytics & modelling approach taken. In this section we will take a deeper dive into the results of these to 1) get a better understanding of key relationships 2) through cross-sectional machine learning workflow understand the predictability of vintage ratings and 3) use natural language processing to further explore the constituents of highly regarded wine.

4.1 Dashboard

The primary purpose of the dashboard is to provide an interactive way of exploring key relationships in the data on various subsets. In Fig. 4.1.1, an example of a dashboard is displayed. It is built with a number of slicers, allowing the user to subset the data on many dimensions. All the graphs are built using the internal Python editor with *Matplotlib* and *Seaborn*, which generally allows for more customization than the standard graphs in Power Bi. The primary drawdown of this approach is the loss of the ability to click on specific data-points in a graph and use these for slicing the remaining graphs. The only way to slice the these graphs are through external filters and slicers. However, as the most pressing issue in the particular business case revolves around building statistical models, the ability to slice all the way down to specific data points becomes less interesting as it takes away focus from the overall distributions of variables. As a result, the usage of Python charts in Power BI does not pose an issue for the time being. It should also be noted that drilling all the way down into vintage level is mostly redundant, as this information is better presented on the Vivino.com site.

The dashboard features four different graphs along with 4 key indicators. All graphs are plotting vintage ratings. The top graphs show ratings' interaction with *vintage year* and *Price (DKK)* respectively, whilst the bottom graphs are univariate and show various features of the underlying ratings' distribution. This ensures the target variable can always be seen in different dimensions. Interestingly, Fig. 4.1.1 show some non-linear patterns in the data, such as the top right graph, in which price and ratings seem to be highly linearly correlated at prices below 1500DKK, whereas

hereafter their relationship seems to fade out. It can also be concluded that vintages from before $\sim$ 2006 generally are guaranteed a higher rating than 3.5, with expensive vintages greatly increasing the probability of getting a rating above 4.0. Equivalently, the dashboard show that ratings centers around 4.0 and is slightly left-skewed. However, these conclusions are obviously only valid to the full dataset for red wines, and the sub-distributions might easily change when performing further slicing. A major weakness to the dashboard setup is the difficulty in in showing the interaction between many variables at once. Since this is highly dimensional data such a feature can be crucial in exploring the interactions between variables as well as their univariate distributions. A good implementation of this is created using the Python Pacakage *Pandas-Profiling*, an exhibit of which is shown in the appendix[3].

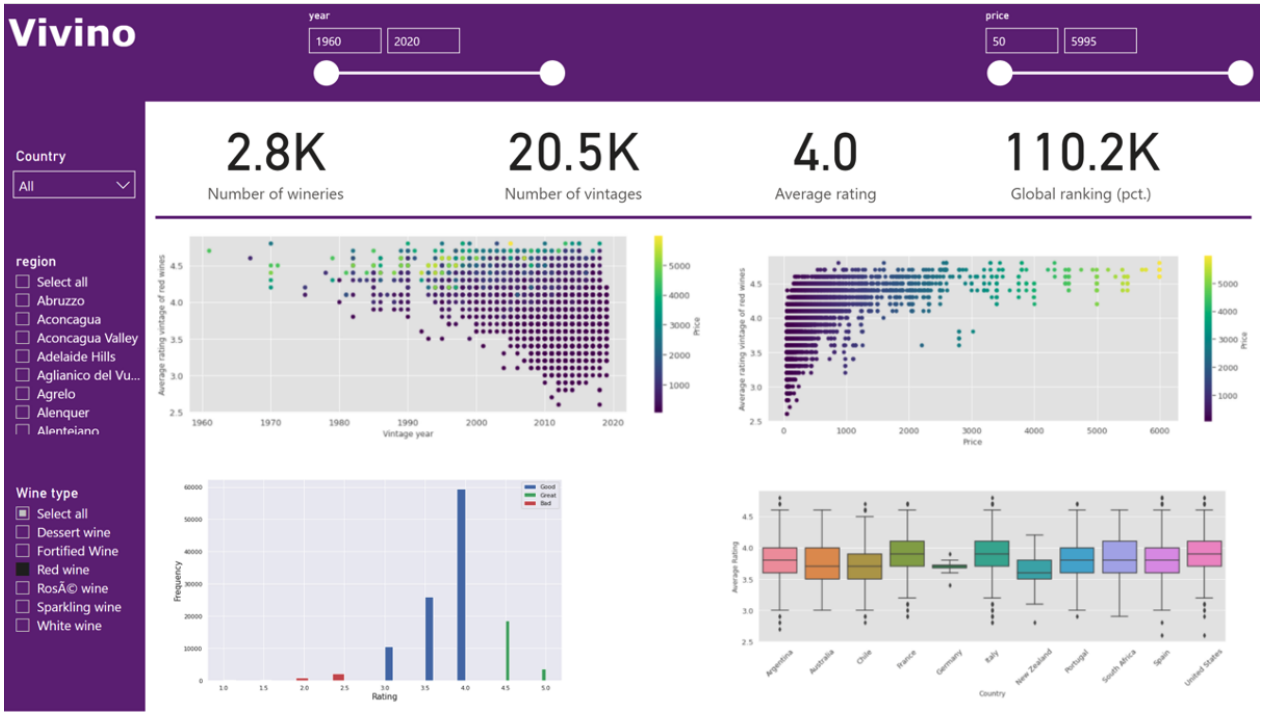

Figure 4.1.1: Dashboard exhibit subsetted to show only red wines.

In an attempt to predict future highly rated vintages, several machine learning models are trained. As mentioned, the target variable is vintage ratings, and the independent variables are the remaining attributes in the vintages dataset, which were described in Table 3.2.1. To ensure model diversity and thus maximize the probability that they are exposed to different types of errors 3 types of models are trained, namely linear, tree-based and neural net. A summary of their out-of-sample performance is shown in Table 4.1.1. It is clear that the tree-based ensemble models, Random Forest & XGBoost, by far has the best performance across all shown metrics. Unsurprisingly, as the data has some non-linear characteristics discussed previously, both linear models are outperformed. More surprisingly, the artificial neural network is also outperformed out-of-sample. This can be a result of either (1 poor model tuning and a need for a different architecture or 2) the ANN is overfitting the data, thus having a too high variance in out-of-sample predictions.

As the XGBoost slightly outperform the random forest we will focus on interpreting this model. From Table 4.1.1 it is clear that the XGBoost explains 86% of the dependent variable's (vintage rating) variance. More importantly, the *Extremeness* parameter indicates that only 2.1% of predictions will be further off than 0.3 rating points. Combined

[3]For the full profile see INSERT_URL.COM

Table 4.1.1: Out of sample performance of various models.

	Linear	Random Forest	Elastic Net	XGBoost	ANN
MSE	0.035303	0.014411	0.059330	0.013859	0.028673
R^2	63.3%	85.6%	34.9%	86.0%	70.2%
Extremeness	10.5%	2.3%	21.4%	2.1%	7.7%
Average overprediction	0.005680	0.000484	0.003612	0.000142	-0.026293

with an average overprediction close to zero, it can be concluded that the XGBoost's predictions are somewhat symmetrically centered around the actual out-of-sample values. This is largely confirmed in the scatter of predicted vs. actual values shown in Appendix B.

A clear advantage of launching ML models, is that they can also be used to find new patterns in the data and inform us about these relationships. Since XGBoost is an ensemble model, evaluation can primarily be performed through feature importance's or better yet through SHAP values. Since feature importance has some major weaknesses such as inconsistency (**Source**), the SHAP values will be used here. In Fig. 4.1.2, the SHAP values for the 10 best predictors are shown in a violin plot. As can be seen, *price* and *year* are among the top 3, which serves as an additional entitlement for these variables being in the previously presented dashboard in Fig. 4.1.1. A high price generally seem to result in a higher model impact, although it can be noted that the majority of the mass has a shap value below 0.0. *Winery_ratings_average*, the second most important feature, also seem to positively correlate with the dependent variable, with a more gradual increase/decline right around 0.0. For *year*, the relationship is less straightforward, and due to the mix of red/blue values to either side, nothing can be inferred with respect to its correlation with the dependent variable. The only thing we can say from this plot, is that *year* serves as an important predictor but it is likely in some non-linear relationship. This is confirmed in Appendix C, in which vintage between 2005-2014 generally leads to a lower predicted rating, whereas vintages before and after this period leads to more positive performances.

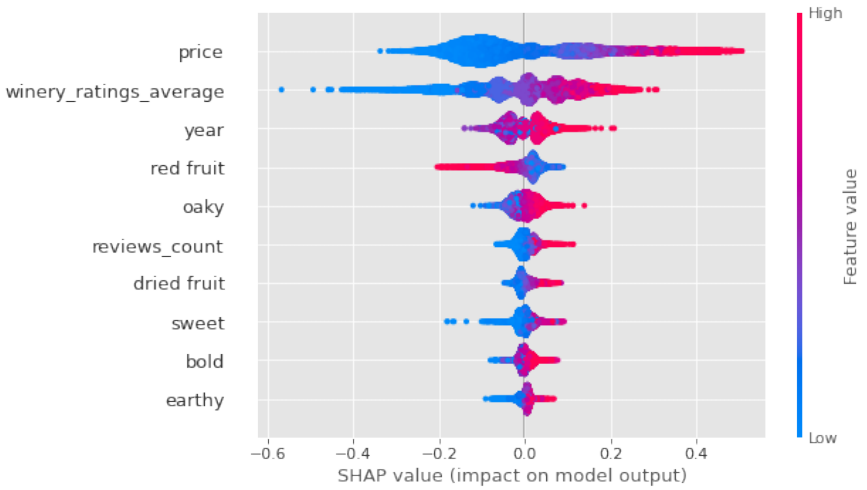

Figure 4.1.2: SHAP importance plot.

4.2 Deployment with Flask to Heroku

The XGBoost model is deployed to a web-based application as a means of using the model in practice. The model is saved in .joblib file and deployed launched https://www.dsba-vinoveritas.com/. The site is created with Flask, a micro WSGI web framework written in Python. It is chosen due to its easy to use structure. It is deployed with Heroku. The site provides an easy to use tool, in which users can easily access the model and utilize it to select the best future wines.

5 APPENDICES

A EXAMPLE OF RAW NO-SQL FORMAT


```
{
    "1169930": {
        "vintage_id": "1762122",
        "price": "113.27 kr.",
        "taste_structure": {
            "bold": "63.5475",
            "tannic": "57.0524",
            "sweet": "6.03826",
            "acidic": "63.9049"
        },
        "flavor": {
            "oaky": "55 ",
            "black fruit": "43 ",
            "earthy": "32 ",
            "red fruit": "18 ",
            "spices": "12 ",
            "yeasty": "10 ",
            "dried fruit": "1 ",
            "vegetal": "1 ",
            "citrus": "1 "
        },
        "foods": [
            "beef",
            "veal",
            "game",
            "poultry"
        ],
        "rank": [...],
        "iterations": 0
```


Figure A.0.1: Wine_ID example. The primary key at the highest level refer to the wine_id, and the values are its features. The No-Sql is optimal for storage for the features *taste_structure, flavor & foods*, since we don't control how these attributes will evovle in the future.

B Random Forest & XGBoost predicted vs. actual values scatter

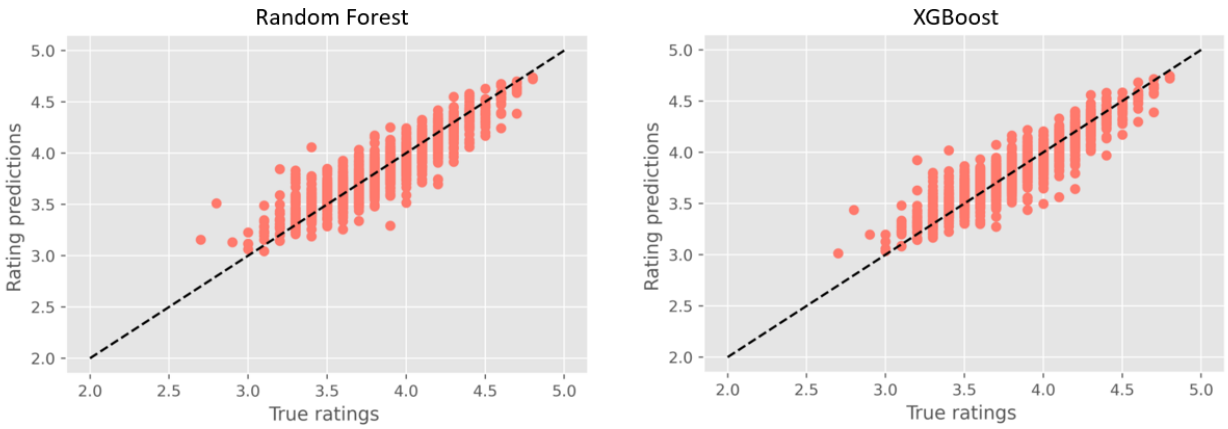

Figure B.0.1: Out of sample performance

C XGBoost SHAP of *year*

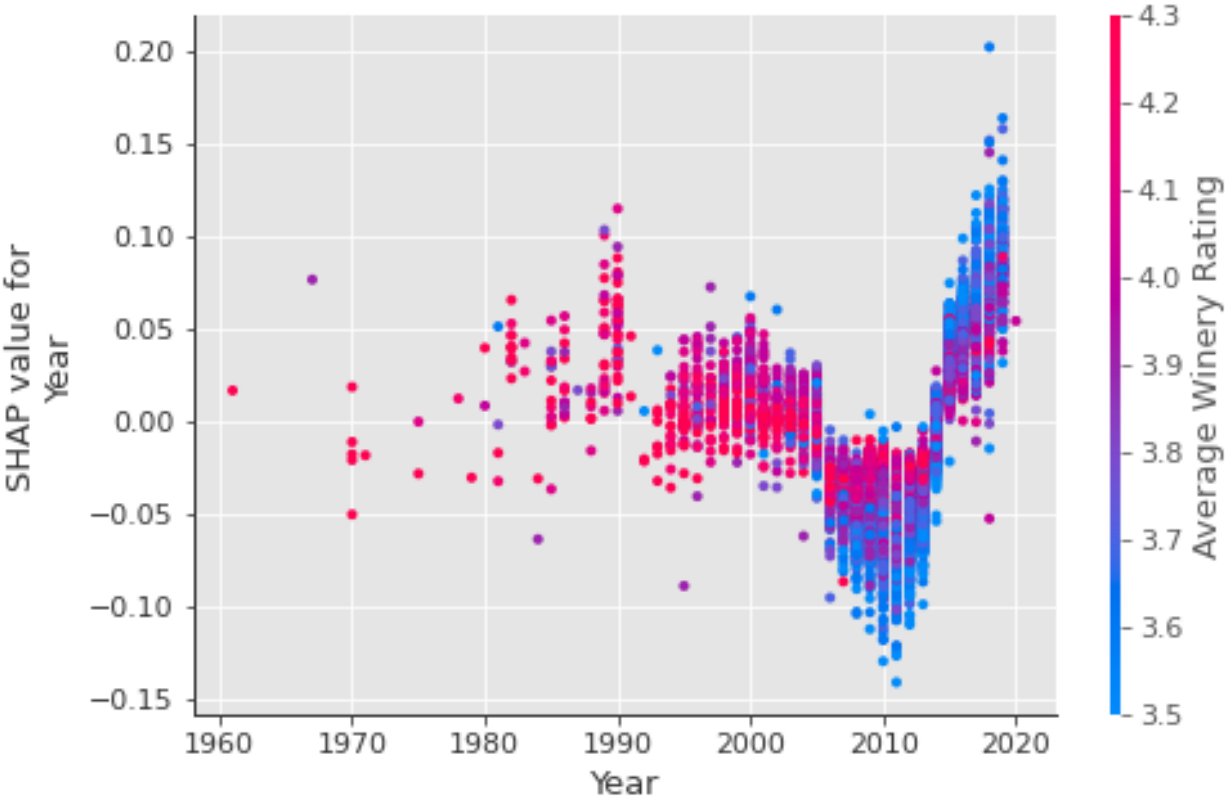

Figure C.0.1: Out of sample performance