AIML421, Assignment 3 Corvin Idler, ID 300598312, idlercorv@myvuw.ac.nz

Exploring and understanding the Data [40 marks]

I used the pandas profiling tools to assess the data set. Detailed statistics and visualisations can be found in my jupyter notebook. My summary findings are as follows:

artist name: 2489 empty_field = missing data -6 different unicode scripts (maybe some patterns to be discovered here (e.g. asian scripts = higher likelihood to be Anime?) -only 2346 unique values… out of 50k entries… so there must be a lot of songs per artist (Bach and Mozart are case in point)
track_hash: 43155 unique values out of 50k. So either 7k duplicate songs in training data set (worst case with different class labels) or hash collisions!

If we investigate these hash collisions (e.g. `'0wY9rA9fJkuESyYm9uzVK5'`), we can see that all textual attributes are identical, but the genre is the same.

If we set up a frequency table to relate the first character of the hash with the class label, we can clearly see that there is predictive value in the hash value.

track_hash	Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock
0	0.489812	0.013127	0.000000	0.000000	0.237951	0.259111	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.412723	0.387532	0.000000	0.000000	0.199746	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.082630	0.000000	0.000000	0.000000	0.831480	0.000000	0.085890	0.000000
3	0.000000	0.000000	0.000000	0.346491	0.000000	0.000000	0.083083	0.000000	0.570426	0.000000
4	0.000000	0.000000	0.000000	0.415139	0.000000	0.000000	0.000000	0.386656	0.000000	0.198205
5	0.000000	0.000000	0.000000	0.023408	0.000000	0.000000	0.000000	0.660249	0.000000	0.316342
7	0.000000	0.000000	0.000000	0.000000	0.503230	0.000000	0.000000	0.000000	0.000000	0.496770

track_hash	Alternative	Anime	Blues	Classical	Country	Electronic	Hip-Hop	Jazz	Rap	Rock
01	0.055814	0.0	0.0	0.0	0.944186	0.0	0.0	0.0	0.0	0.0
02	0.769565	0.0	0.0	0.0	0.230435	0.0	0.0	0.0	0.0	0.0
03	1.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0
04	0.095833	0.0	0.0	0.0	0.904167	0.0	0.0	0.0	0.0	0.0
05	0.608696	0.0	0.0	0.0	0.391304	0.0	0.0	0.0	0.0	0.0
…	…	…	…	…	…	…	…	…	…	…
7l	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	1.0
7j	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	1.0
7k	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	1.0
7l	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	1.0
7m	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	1.0

228 rows × 10 columns

For the first character there isn't perfect class separation, but things become even clearer if we take the first two characters. We see that many rows now belong to one single class (e.g. Rock). With 50k data points and 228 rows it's clear that many rows (if not all) represent many entries in the training data set.

As per the email of the course instructor, this is evidence of data leakage if one knows or assumes that the hash wouldn't be available in reality before on knows the genre. The hash is therefore to be excluded as a feature from further analysis.

track name: 41699 unique values. Not surprising that some songs have the same title
popularity: 99 discrete values -approximately normal distributed (with two modes though), but inexplicably high number of 0 which is likely (or at least partially) due to missing data (e.g. nobody voted yet for that song.
acousticness: horseshoe type distribution -a lot of values close to zero but only one single entry exactly zero. So no indication of a lot of missing data
danceability: approximately normally distributed -Seemingly no missing values
duration in ms: 4939 values are negative (-1). Must be missing data according to my understanding of the space-time continuum :-) -One extreme value (4830606ms = >80min) might be an outlier or not, but might need some treatment as it messes up any plain vanially standardisation of the data
energy: seemingly no missing data -rather oddly shaped distribution…

instrumentalness: 15001 missing entries (30%) -maybe need to check if that is the same across classes

key: no missing values

liveness: right skewed distribution, no missing values

loudness: left skewed distribution, seemingly no missing values

mode: no missing values

speechiness: no missing values. more like a weilbull distribution

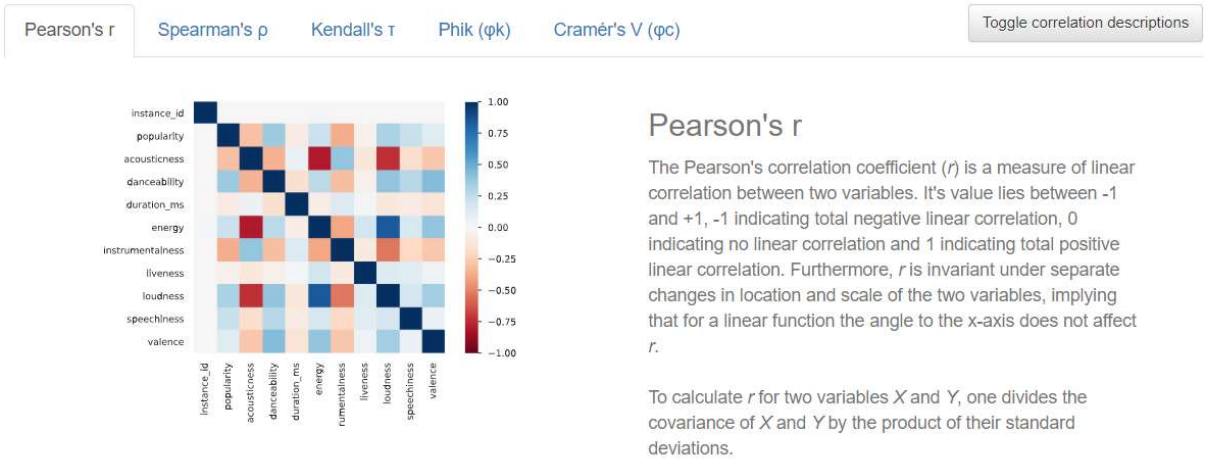
tempo: 4980 missing values encoded as '?', hence python thinks it's a categorical variable. Hence, I converted it back to numeric for the by class historgrams below

obtained_date: majority on 4th April… ca. 10% on 3rd April and some other dates

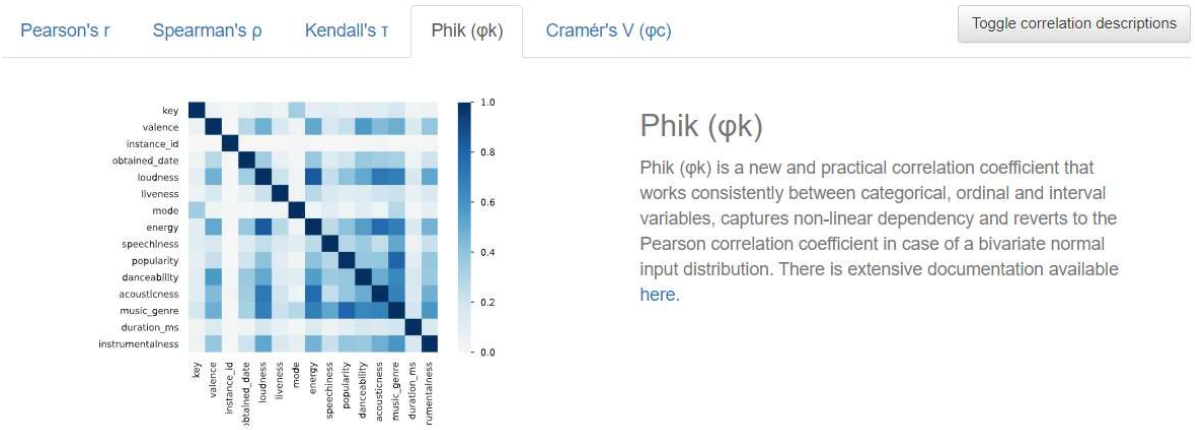
valence: humpy distribution… like some beta distribution

Correlation analysis:

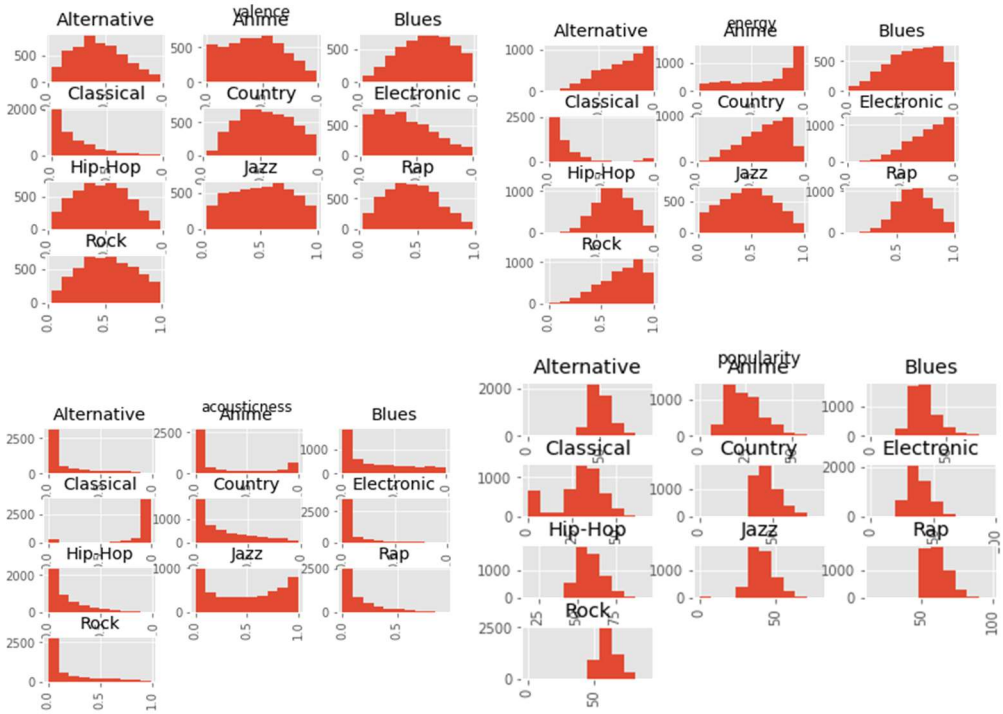
correlogram show some non-surprising positive correlation between loudness and energy and some highly negative correlation between energy and acousticness as well as loudness and acousticness so there might be some collinearity present



The Phi k correlation analysis (c.f. [Phi_K Correlation Analyzer Library — Phi_K correlation library documentation (phik.readthedocs.io)]) shows some interesting relationships (or lack thereof) between features and class labels. It looks like popularity, energy, loudness, danceability and acousticness might be features worth exploring. Key and mode might be less relevant (as singular features and maybe only relevant in relation with other features).



Distribution analysis broken down by class:



-distribution of valence and instrumentalness for classical category for looks vastly different from the rest of the classes
-huge variety in shapes or types of distributions across classes for energy. Might make things messy when trying to standardize...
-classical and jazz classes have distributions of acusticness that are rather different then for the rest of the classes
-popularity vastly different for classical than rest. Most missing values or zeros in classical!! Should be exploited rather than treated as a nuisance factor

Developing and testing your machine learning system [50 marks]

(15 marks) Discuss the initial design of your system

I started designing my system with the initial thought of establishing a baseline using the most basic prediction "system" I could come up with. My initial thought was, that a lot of artists predominantly "occupy" one genre (e.g. Mozart didn't do much else but classic music). After I checked that the test set had a sufficient number of artists that also occurred in the training set, I thought I'd give a simple artist lookup a go and assign the predominant genre of an artist to each song in the test set.

The rest (non-matching artists in the test data) I filled with a random guess out of the 10 classes.

It turns out that this approach was pretty good compared to a lot of entries in the public leadership board. A simple look up might not really be a data mining algorithm, but on second thought, it's basically a naïve bayes using just artist as a feature. I just found it a lot less hazzle to implement it as a lookup table rather than as a naïve bayes (one hot encoding etc).

I understand that this initial design is not much of a design at all, but that is basically my philosophy throughout my career. I start with the simplest algorithm or system I can think of to see how far that

gets me to know what I need to beat with more sophisticated algorithms. I find it useful to establish some baseline, particularly if in real life there is no public leadership board to one can compete with.

(25 marks) Discuss the design of one or more of your intermediary systems

Impressed by the performance of the artist name I tried to match based on artist name and song name at the same time and what was left over I matched only on artist name and the rest I random guessed. Strangely enough the performance was worse ☹

Fascinated by the textual features and informed by the initial exploratory data analysis I realized that many songs from the Anime genre contain Asian Unicode scripture.

I created a Boolean feature to flag when a song title contained Asian scripture (Han, Hiragana,Katakana) and assigned Anime as genre if that was the case. Performance improved, but not by a lot, as only a small fraction of the overall data set was covered by that feature. So strong feature by only useful in few cases.

Last but not least I created a feature to flag missing popularity information as this mainly occurred in the Classic genre. I assigned the Classic genre to all cases with missing popularity (value == 0).

Again I think a strong feature but little improvement as not too many cases covered by it.

In the end I wasn't happy with random guessing the songs for which there was no match of artist in the training data set, so I decided to try some actual algorithms (KNN, RF, XGB) using just the numerical features of the dataset. I excluded instrumentalness as I thought it had too many missing values. Gradient boosted decisions trees came out best, but the accuracy was by far not as impressive as the lookup based on artist name. Instead of incorporating categorical features into the decision trees, I decided to keep the artist look up (e.g. naïve bayes) and use the XGB prediction only for cases with non-matching artist. Performance improved furter.

Last but not least I decided to also use XGB predictions for the artist name `'empty_field'` as it felt non-sensical to assign the majority class for "unknown artist". Performance improved even further.

(10 marks). Use your judgement to choose the best system you have developed

I chose the system last described in above paragraph. Use XBG with numerical feature to predict artist name "empty field" and those that are not included in the list of artist from the training data set. For song that contain Asian scripture in the title assign Anime. For songs that have zero popularity, assign Classic and for the rest use the predominant genre of the same artist in the training data set.

I'm pleased and annoyed at the same time by the combination of a "look up table" (naives Bayes with artist name as only feature) with XGB predictions. I didn't combine the two by means of model stacking in a traditional way, but gave hard preference to one component of the system for certain data paints (e.g. matching artists). It feels a bit quick and dirty, but the performance is not too bad.

At the end of the day I've seem many Kaggle competition where people got the low hanging fruits out of the way with hard class assignments based on human logic and knowledge rather than leaving everything to a data mining algorithm. So maybe that is one take away… that if the only tool you think you have, everything looks like a nail… but if you're a bit open minded, you can find creative solutions through a combination of methods (tools/approaches).

Challenge: Reflecting on your findings [10 marks]

How easy is it to interpret your chosen machine learning model, i.e. how easy to comprehend why certain predictions have been made?

The majority of the model is super easy… I kept it really stupid simple by making the artist name the main feature and only supplement that with recognizing Japanese scripture in song titles (to help strengthening the identification of anime songs) and using missing popularity data (to help identify classical music). Only for what is left afterwards I predicted classes with XGB.

Due to the simplicity of most of the model it's pretty "trustworthy" for known artists… but it's also less "trustworthy" for anything completely new as the class assignment comes from boosted decision trees. It's obvious that my model does OK in the public leadership board (at least at the time of writing), but it's also obvious that is fails to some degree to generalize it only partly relies on algorithms that capture the deeper underlying patterns in the data.

I feel my system is a bit of a hybrid solution. Stupid simple when it can… more sophisticated when it has to.

Public Kaggle Competitions [20 marks]

I chose the Optiver Realized Volatility Prediction[1] competition for this part of the assignment.

The competition is about forecasting of volatility of stock prices for the purpose of option pricing (time value part of the option)[2]

What are the steps involved if working on the competition?

I don't think this competition is much different from any other data mining problem. So I would argue the steps involved are the typical CRISP-DM steps.

- Business understanding: Luckily the competition provides some explanatory information about why volatility matters for option pricing [3] but also a mini tutorial about some of the financial concepts at hand and some sample code for calculating some of the financial metrics.[4]
- Data understanding: The Data page of the competition gives some explanation of the data. Data quality assessment / profiling would clearly still have to be done
- Data preparation: besides cleaning (where need be) the feature engineering will be really interesting for this competition! The competition allows for the use of external data, so "feature engineering" takes quite a different turn here. Anything free on the internet would be up for grabs….
- Modeling – What modeling techniques should we apply?
- Evaluation: a bit more tricky than normal as it's a code competition, where only the first few rows of the test set are available for download. The remainder will only be available to your notebook when it is submitted. The final private leaderboard will be determined using data

[1] https://www.kaggle.com/c/optiver-realized-volatility-prediction/overview
[2] https://www.optiver.com/insights/guides/options-pricing/
[3] https://www.optiver.com/insights/guides/options-pricing/
[4] https://www.kaggle.com/jiashenliu/introduction-to-financial-concepts-and-data

gathered after the training period closes, which means that the public and private leaderboards will have zero overlap.

- Deployment: slightly less applicable in the context of this Kaggle competition. There are some code requirements though (e.g. runtime limits)[5]

Are there any important pattern you have found on the data?

Waayy too much data and too little time ☹

How will you perform data preprocessing on the data?

The first task already is to match the order book data with the trading data and make sense of it and derive some features from it. The open ended task though is to find external data that is predictive of short terms (e.g. 10min) volatility.

What kind of machine learning techniques will you use and why?

I'll probably end up using some form of regression as a baseline and then use more sophisticated algorithms like feed forward neural networks, LSTM, LightGBM as I expect more complex interaction between features than can be captured by simple regression models.

Compare your chose Kaggle competition with our InClass competition

The Optiver competion has a clear time component to it. It is about forecasting a calculated metric over an interval of a financial time series. The dimension of time wasn't really a dominant concept for the inclass competition. The Optiver competition is a regression problem, the InClass competition is a classification problem. The amount of data involved for the Optiver competion is some orders of magnitude higher (ca. 3GB in memory) than for the InClass competition and as the use of external data is allowed, the sky is the limit. Due to the nature of the problem the different instances (here stock) can be used to inform the forecast of each other (multivariate analysis), while in the case of the InClass competition there is no such dependence between the instances. The InClass competition is a "static" dataset anyways… so nothing really evolves over time.. there is no notion of looking into the past across instances to learn something about the future of some or all instances.

Based on the above mentioned differences (e.g. classification vs. regression/forecasting and fixed data set vs. any predictive data that is available on the internet) the machine learning techniques used are obviously rather different.

[5] https://www.kaggle.com/c/optiver-realized-volatility-prediction/overview/code-requirements