Predictive Beer Analytics

Joshua Weinstein, Jim Sundkvist, Marek Kühn

***Abstract*—This project aims to apply various concepts of Python programming as well as data mining and processing. Based on a large amount of user and beer data, the application uses varying techniques including data mining, natural language processing, and color analysis to predict user-defined beers' desirability in the world through a web application.**

I. Introduction

The Predictive Beer Analytics project has been initiated in the Fall of 2014 as a part of "*Data Mining Using Python*" course at Denmark Technical University. Its purpose is not only to exercise the study material, but also to give users some useful, informative results. To satisfy this, we used our previous experience with web development and created a lightweight web-service with visual representations of the results. The users can design their own beer by defining its style, alcohol by volume, keywords and dominant label color. This information is then used to generate a desirability map.

The application to retrieve and analyse data along with our web application (without the database) can be found on our GitHub account. The web-service can be run locally with an installation of Django, MySQL, and a copy of the database.

For more information, the documentation is available on ReadTheDocs.org.

II. Data mining

Being the main aspect of any data mining project, retrieving the data, and a lot of it, is the first course of action. All data used by Predictive Beer Analytics was gathered through the Untappd API, a beer review website which provides access to the data it collects. Unfortunately, the service limited our access to their data to 100 calls an hour. This significantly increased the amount of time required to gather data. The script, `untappd.py` provides data structures to store Untappd data as well as uses the *Requests* module to make calls to Untappd's servers. Due to the available requests through the API service, it was necessary to first retrieve a list of active users and then later use their username to gather information about beer they have reviewed in the past. Over the course of roughly two weeks of mining, a reasonable amount of information was stored. Seen in Figure 1, over 200.000 user reviews are used to predict the desirability of a beer based on location. To keep this information up-to-date, `dataQuantity.py` script has been included in the library.

After data has been gathered, updated, and normalized with methods in the main script `predictiveBeerAnalytics.py`, it is split up into users and beers where users contain a location, using the Google Geocoding API, and their ratings and beers are composed of descriptive keywords, alcohol content, label's

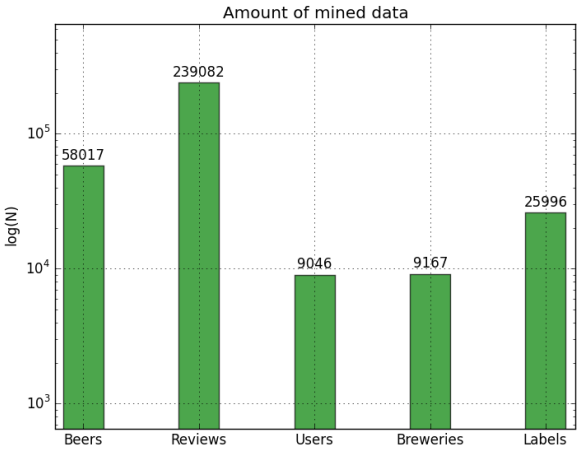

Fig. 1. Amount of used data (by the time of presentation)

url, style, and other unused information. These are then used in conjunction with other scripts we have written to analyse label colors, determine keywords from descriptions, find relationships between alcohol content and location, and determining relationships between a user's location and the style of beer they enjoy.

III. Machine learning

There are two applications of machine learning:

- **Natural language processing**
- **Image color analysis**

With its extraordinary lexical resources and easy-to-use interface, **Natural Language Toolkit** (NLTK) was an easy choice. It is used to tokenize the sentence in order to classify the words according to its type. After various filtering (usage threshold, invalid characters etc.), we calculate the average rating of each keyword. The best, worst and the most used keywords are presented on the website. The relevance of the results is to be discussed later. The actual **keyword extraction** function is defined in `keywordExtractor.py`, making it easy to reuse in other projects. It is possible to introduce regular expressions in order to improve the keyword recognition, but the results are reasonable even without using it.

While thinking about all aspects that can possibly affect the beer desirability, we introduced also the visual point of view. We have no information about the color of the bottle. Nevertheless, more than half of the beers from untappd have a link to the beer label picture attached. All we had to

do is determine how the beers are rated in relation to the label color. The **dominant colors clustering** is done with sklearn.cluster module, built on NumPy, SciPy, and matplotlib, well-known open-source libraries. All of which are also used regularly in other context in this project. Currently, we use **K-means clustering** to determine five of the most used colors in each label. The class responsible for the calculations `labels.py:Image` is designed such as the parameters like number of clustered colors are easy to change in the future.

To check if the result is correct, we use the clustered colors to rebuild the picture as in the figure 2. Since we observed a lot of labels not having rectangular shape, we implemented a **custom masking algorithm** based on the color differences. It seeks a border at which certain threshold is overrode and generates numpy.mask. This is covered by `labels.py:Mask`.

Having these colors extracted, they need to be associated with ratings. Otherwise there would be no measurable impact on the total beer desirability. We couldn't match those color directly to the user input, because that way its rating would have to be calculated every time the request is made, which is not acceptable given the amount of data. So we came up with a solution by using **template color palette**. It serves as a **fixed categorizing root**. None of the available functions were suitable for comparing the RGB values of the colors, so we designed our own classification algorithm. It uses the well-known euclidean distance formula to determine which color from the template color palette is the closest. However, simple distance in RGB colorspace doesn't correspond to human perception. A change of one parameter results in much more different color than the same change distributed between multiple parameters. It was possible to overcome this by converting the colors to **YUV colorspace** (Fig. 4), which takes human perception into account, making the response more or less linear.

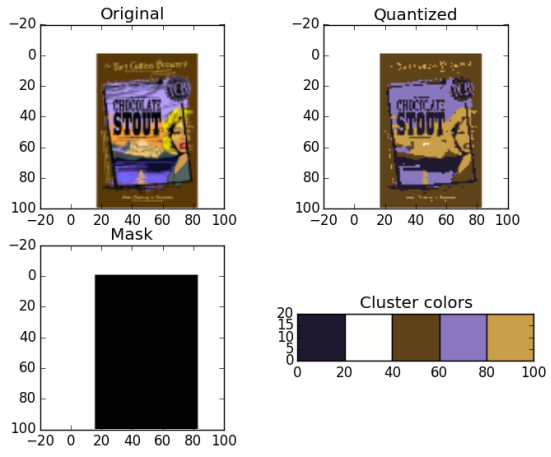

Fig. 2. Image color clustering

Fig. 3. YUV linear colorspace sample for Y = 0

IV. Web Application

The web application uses one of the most popular Python frameworks - Django. This allowed us to create elegant web application while still being able to focus on the server-side python scripts rather than web-design.

The web-design and structure of the web application is pretty straight forward: A landing page that serves as a portal and first encounter information of the web applications functionality. A prediction page where the core of PBA is preformed. The user can fill in data that would reflect their beer and check where it would do well with a histogram. A Description page where a dynamically created ranking of the processed words are presented. Here it is also possible to search for a specific word. The result will be a list of similar words and their ranking starting form the highest rating. A color page where the user can see the rankings off colors used in beer labels. An about page where we tell the story and information about Predictive Beer Analytics. And last a map page where we showcase some predefined results from the predictor.

Django uses the MVC architecture. So the project has a data model, representing our processed data. A single view file that contains all (as the size of the Web application was not large enough to warrant separation of concerns in our eyes) application level functionality. And templates which uses CSS and JavaScript to make a smoother appearance. Bootstrap was used to speed up the styling in this regard. All templates extends the base.html template which acts as a master page.

The data that the web application uses are stored in a MySQL database. The data itself is generated from the data mining scripts and saved to the webapps database by importing them as a csv file. The web application uses **Django's database-abstraction API** to perform the basic CRUD functionality to the data model.

When it comes to security, Django has some great features that will bring about basic **security** with minimum effort. Django's database-abstraction API automatically sanitizes any input so **SQL-injection is not a problem**. Cross-site-scripting is dealt with by using **cross-site request forgery (CSFR) tokens**. Those are automatically attached to the server responses

Fig. 4. Database data model

so we need to add it only while doing POST.

Django also has some **predefined user authentication and authorization functionality**. In the web application we do not use any access control so this part is unnecessary until a time arrives when we do need it.

The decision to use these technologies is pretty straight forward: The technologies were familiar, widely used, well documented and easy to use. This lead to quite a fast development of the Predictive beer analytics GUI.

V. Documentation

The documentation has been created with Sphinx tool. It allows us to generate the documentation in various formats including html, pdf, epub etc. from reStructuredText and python sources. For this to work, the `doctstring` convention is required. Apart from this, we also performed a brief code-style checking according to PEP conventions. The documentation is hosted on http://predictive-beer-analytics.readthedocs.org/, which integrates directly to the GitHub account.

VI. Discussion

There is usually no simple way to determine the relevance of the results in these kind of projects. However, we tried to validate it as much as possible.

As we went through the list of keywords, we found a lot of invalid words which NLTK wasn't able to recognize. For example links, unknown abbreviations, misspelled word, html snippets etc. Those could be easily eliminated by defining usage threshold, that is to filter out words used less times than let's say 20. This is also to make the results more reliable, since the worst ones are used just once. There is still space for improvements here. For instance we can imagine using regular expressions for the rating of English phrases and idioms instead of just separate keywords. Altough after having the results generated, they seem reasonable enough for purposes of this project. In the end, it is only logical to expect words like "inexpensive" and "watery" to be much worse rated than "immeasurable" or "aged". Other language support has not been considered in this project.

It is problematic to assign ratings to the whole color palette. There was an idea of double-clustering the colors to form rated groups, but it made them look dull and actually quite distant from the original ones. This method is too radical. Consequently, we decided to use supervised classification to assign the colors to template palette that covers all the common colors. At this point the palette is hard-coded in the `labels:Image` class. It is shared with the web application and also used as a prediction parameter. To use full color picker on the website (e.g. using jQuery), it is possible to use the same classification function defined in `labels.py`.

VII. Further development

At this early stage, the quality of the results highly increases with getting new data. The scripts to gain information from Untappd are finished, so the only limit is the time. As seen in the figure 5, there are still some areas with little or no user reviews. Those are rendered as zero rating blue. Other sources of beer information and reviews could also be implemented in the future. It would be interesting to relate brewery information to the beer popularity as well. This required brewery data is already available to us.

Depending on the quantity of mined data, the density of the map's grid can be increased. Current grid area varies between 30 000 km^2 and 200 000 km^2. Another idea to consider would be implementing different location distribution methods. Now we use latitude/longitude classification, but Google API offers other ways to lookup the current state or city. This would all lead to more detailed results. Machine learning could be improved by introducing more sophisticated algorithm to extract keywords and phrases.

VIII. Results

Using `PBAMap.py`, `keywordClassifier.py` we are able to graphically visualize the effects of specific words, alcohol content, and beer style on a user's rating. Figure 5 shows the consistent and seemingly global idea that people typically enjoy stronger beers. While there is a clear correlation between user reviews and and a beer's alcohol content, the type of beers an area likes are typically varied and relatively unpredictable. Though it does seem that the United States enjoys IPAs more than ciders and fruitier beers.

It was also interesting to see that certain words used in a beer's description had a large impact on a user's review of the beer. For example, beer descriptions containing words like "inexpensive", "alcohol-free", and "grandma" all had an average rating of below three out of five stars. Whereas "sought-after", "chocolate-covered", and "fabled" all had ratings above four stars.

IX. Conclusion

At the beginning of Predictive Beer Analytics, our goal was to find relationships between the alcohol content or style of a beer and the location of a user's rating to determine what areas enjoy the most. As the project progressed, ambitions rose and we wanted to see if the color of the label or the description of beers made a significant impact on the desirability of a

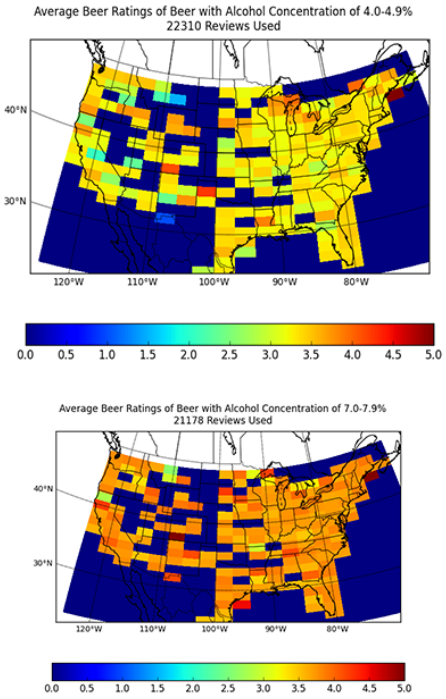

Fig. 5. Comparison of average beer ratings in US based on alcohol content

beer. Using techniques described in the data mining course, it was found that there were relationships between the beer properties and user ratings. There were also findings on the locations of certain preferred types of beer. It can be shown that the challenge of determining what makes a good beer was overcome.