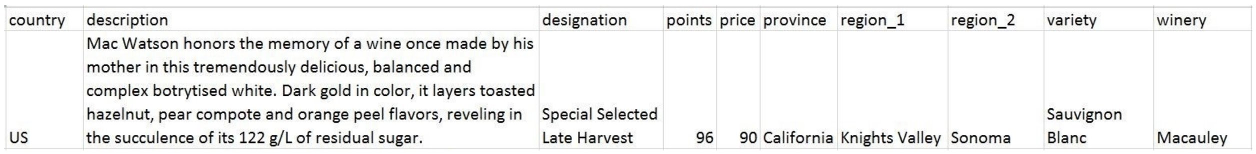
CS 221 Project Progress:
Wine Review Keyword Prediction

Noam Habot, Mackenzie Pearson, and Shalini Ranmuthu

The primary goal of our project is to be able to take information about various wines such as the region, price, and variety, and more, and to be able to predict keywords about the wine without looking at the wine's description or review.

1 Model

We will be developing our analysis based on an online dataset from Kaggle that contains wine reviews scraped from the WineEnthusiast website. The image below shows an example from this dataset that used as a training input. There is a field for the detailed description itself, as well as other fields for related information.



country	description	designation	points	price	province	region_1	region_2	variety	winery
US	Mac Watson honors the memory of a wine once made by his mother in this tremendously delicious, balanced and complex botrytised white. Dark gold in color, it layers toasted hazelnut, pear compote and orange peel flavors, reveling in the succulence of its 122 g/L of residual sugar.	Special Selected Late Harvest	96	90	California	Knights Valley	Sonoma	Sauvignon Blanc	Macauley

While the Kaggle dataset has already been cleaned, we will need to do some pre-processing on this dataset before we can actually use it in our model.

1.1 Building Dictionary of Wine Keywords

First, we will need to extract the most important keywords from all of the wine reviews and build a set, i.e. "dictionary," of relevant key terms that describe wines. To do this, we first concatenate all of the wine reviews into a wine description corpus. We then run a script that takes an input n (the number of unique words that we want to be outputted), normalizes all the words in the corpus by converting them to all lower case and removing any punctuation, and then counts the frequencies of each unique word and stores them in a mapping. This dictionary is then sorted by decreasing frequencies, such that $freq_i > freq_{i+1}$, and written into a file named "wineWordFrequencies.txt", in the format: $word_i : freq_i$.

One particular issue that we ran into is the computational inefficiency of high level programming languages, such as R, when attempting to perform these calculations involving large-scale text (an average of 100 words/description for about 150,000 wines, resulting in around 15,000,000 words). To mitigate this issue, and to be able to run the script with quickly with a modified n, we wrote the script in $C++$ and its runtime is around 5 seconds.

This technique leaves us with the top n frequented words from the wine descriptions, but the top 5 keywords we see in this list are "and", "the", "a", "of", and "with". The next step is to remove such words that are not satisfactory keywords for wine reviews. To do this, we run the same script on a text corpus from a book; we used "The Adventures of Sherlock Homes" by Sir Arthur Conan Doyle. We saved the dictionary including the words and corresponding frequencies in a file named "bookWordFrequencies.txt". The idea is that since the book contains a broader set of English language words that are not particular to a specific subject, we can use the top m frequently used words and subtract them from the top n words we found from the wine description keywords in order to prune out keywords that are not wine-specific.

Finally, we take the ”wineWordFrequencies.txt” and remove the intersection between its words and the words in ”bookWordFrequencies.txt” to generate the wine keywords that predominantly govern the final keywords that we will use to continue with our analysis. The top 5 keywords in the resulting list are “flavors”, “aroma”, “acidity”, “tannins”, and “ripe”. Note that both n and m are hyperparameters that can be tuned as we continue the project and as we see fit. Currently we have $n = m = 10,000$ which results in $k = 6,342$ final keywords.

1.2 Adding Wine Keyword Indicators to Dataset

The file with the list of the final k keywords can now be inputted into a Python script that will add keyword indicator features to every wine in the dataset. The objective is that each wine will have the all the features from the original dataset concatenated with k new response variables (one per keyword). The script first reads in the original “wines.csv” dataset file and iterates over each row (containing the original features for an individual wine). For each wine, the value of the new response variable will be a 1 if the corresponding keyword is present in the wine description feature, and 0 if the keyword is not present. The new response variables are appended to the old feature values and then stored in a new “dataset.csv” file.

It is important to note that the searching for the keyword in the description should be case-insensitive and that superstrings of keywords are also detected as the presence of the keyword (since aroma and aromatic indicate similar properties of the wine). Since this process loops through all approximately 150,000 wines in the dataset and adds k features, the script takes around 30 minutes to run, but has to be run only once in the pre-processing pipeline. However, speeding up this process and reducing the size of the resulting dataset would be valuable, and we discuss some ideas to address this in the *Future Work* section.

1.3 Logistic Regressions

Next, we will train a separate model for each of these k keywords in our “dictionary,” where the response variable y will be a binary feature indicating whether or not the keyword is in the wine’s description, and where our predictor/feature variables are the rest of the information we have on each wine (country, points, price...). To do this, we will make the assumption that the log-odds of a response y can be expressed as a linear function of our k feature variables, i.e.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k$$

where p is the probability of the keyword being in the description. Here, we can interpret the value of e^{β_j} as how the odds of the keyword being in the description increase or decrease for a one unit increase of x_j, holding all else equal. Rewriting the above, we find

$$\hat{p}(x) = \frac{1}{1 + e^{-(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_k x_k)}}$$

where maximum likelihood estimation is used to solve for the parameters that best fit the data. See the *Algorithm* section for more details.

2 Algorithm

To find the parameters for logistic regression we maximize the likelihood function:

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$$

Now, taking logs and subsituting $p(x_i)$ into the above we find:

$$\begin{aligned} l(\beta_0, \beta) &= \sum_{i=1}^{n} -\log(1 - p(x_i)) + y_i \log\left(\frac{p(x_i)}{1 - p(x_i)}\right) \\ &= \sum_{i=1}^{n} -\log(1 + e^{\beta_0 + x_i\beta}) + y_i(\beta_0 + x_i\beta) \end{aligned}$$

and differentiating with respect to all β_j and equating to 0 we derive our system of equations:

$$\begin{aligned} \frac{\partial l(\beta_0, \beta)}{\partial \beta_j} &= \sum_{i=1}^{n} y_i x_{ij} - \frac{1}{e^{\beta_0 + x_i\beta}} e^{\beta_0 + x_i\beta} x_{ij} \\ &= \sum_{i=1}^{n} (y_i - p(x_i)) x_{ij} \\ &= 0 \end{aligned}$$

which can finally be solved numerically via Newton's Method.

It is important to note that certain feature vectors (such as "region_2") are quite sparse and therefore not useful for prediction. The final dataset passed through this algorithm will thus be a subset of the original data set with such features removed. Our algorithm will also be tested on a reserved subset of the Kaggle dataset (not used for training). During testing, the inputs will be in the same format as the training inputs with the wine review field removed, and the outputs will be a list of keywords for each input.

3 Performance Evaluation

3.1 Preliminary Evaluation Metrics

For each keyword, after running the logistic regression on the training set, we calculate the test error on that corresponding keyword and report it as our initial evaluation metric for

each keyword's model. We can take the average of these test errors to have an ultimate test error for all of the logistic regressions.
The mean preliminary **Test Error** over the top 25 keywords: **11.94%**.

Table 1: Error Rates per Keyword (16 most frequent keywords)

keyword	flavors	aromas	acidity	tannins	ripe	spice	berry	blackberry
error rate	33.5%	15.3%	18.1%	18.8%	20.6%	17.4%	29.1%	11.7%
keyword	crisp	blend	vanilla	plum	citrus	chocolate	cabernet	texture
error rate	10.0%	8.7%	12.0%	9.3%	7.5%	9.7%	8.8%	7.2%

Highlights: The keyword "flavors" recieved a test error of 33.48% which could be considered quite poor; however, is not surprising as the word "flavors" is not specific to a particular set of wines. This can be contrasted with, for example, the keyword "citrus", which obtained a test error of just 7.49%!

3.2 Future Evaluation Metrics

We will have 2 evaluation metrics of performance once we finish the entire analysis on a larger subset of the keywords. The first will be what percentage of the "dictionary" keywords present in the review did our model include in the output prediction (in the range [0,1]). The second will be a negative percentage of how many predicted words in the output were not present in the review (in the range [-1,0]). The separate metrics will help us determine if our model is too verbose or not accurate enough, and combining the metrics could also produce an overarching indicator of performance (in the range [-1,1]).

4 Future Work

A problem that we are currently having with stages 2 and 3 of the model creation (adding wine keyword indicators to the dataset and logistic regressions, respectively) is the sheer size of the dataset. After adding the wine indicators in stage 2, the dataset expands to include an additional $|observations| * |keywords| \approx 900,000,000$ binary values. These binary values will indicate if $keyword_j$ exists in obs_i. Since most of these end up being 0, perhaps we need to figure out a way of representing the columns as sparse vectors and incorporating this method into processing the algorithm to improve loading and computational times. We can speed up the creation of the dataset by optimizing the Python script in C++ instead.

Additionally, we would like to delve deeper into the different factors that each feature variable has. For example, $region_1$ and $region_2$ have many missing values, and we might be better off if we don't include them as features in the model. Also, they will most likely have high multicollinearity with other features such as region and country, since they are subsets of each other, which is something we should be avoiding.

We can also separate out a validation subset in order to tune the hyperparameters mentioned in Section 1.1 (the n and m number of highest frequency words to extract).