Classify Wine Variety by Description/Review Text

Qihan Guan
A92413483
qiguan@ucsd.edu

Chao Huang
A13136300
chh115@ucsd.edu

Bocheng Huang
A92420614
boh010@ucsd.edu

ABSTRACT

Some professional sommeliers and wine enthusiasts have the special ability to identify a wine's type/variety through blind tasting. In fact, a predictive model can be derived from this ability. In this project, our goal is to develop a predictive model that predicts a wine's variety (Pinot Noir, White Blend, Riesling, etc.) based on its description/review given by a taster. Such predictive model is developed through the 130k wine review dataset on Kaggle scraped from a famous wine review website *WineEnthusiast.* This model can also be useful for other wine review websites to better analyze and process customers' reviews. Sometimes many reviews do not contain any information about the wine's variety. If we can predict the variety thorough the review, then we will be able to process large unlabeled review sets and assign each review a label (wine variety). This will help us extract useful information from customers' reviews better and make more appropriate recommendations of products within the same variety. Further, such review categorization model can be applied to many other fields such as restaurant review, movie review, Amazon product review, etc.

Keywords

prediction; categorization; multi-label classification; Naïve Bayes; Logistic Regression; Multi-Class Support Vector Machine; text-mining

1. Dataset

1.1 Dataset description

The dataset used for this study is a Wine Review dataset from Kaggle.com[1]; the review data was originally scraped from WineEnthusiast during the week of June 15th 2017 and updated on November 24th 2017. The dataset contains 129970 wine reviews which provides information including a wine's designation, price, variety, winery, user's rating score, user's description on the wine, and other features (Table 1). As the data provider stated, with the information given in the dataset, it offers great opportunities for sentiment analysis and other text related predictive models.

1.2 Exploratory Analysis on Dataset

Table 1: Dataset features

Feature	Feature Type	Info
Country	Textual	country the wine is from
Designation	Textual	the grapes' vineyard
Points	Numerical	user rating on wine
Price	Numerical	cost for a bottle of wine
Province	Textual	province/state wine from
Region_1	Textual	wine growing area
Region_2	Textual	more specific region
Taster_name	Textual	name of the person who tasted and reviewed the wine
Taster_twitter_handle	Textual	Taster's twitter account
Title	Textual	Title of the wine review, often contains vintage
variety	Categorical	Type of grapes used
winery	Textual	Winery that made the wine

As the goal of our study is to predict the variety of a wine by reading a user's description of the wine, an observation of the wine's distribution is needed for determining optimal models and approaches for the prediction.

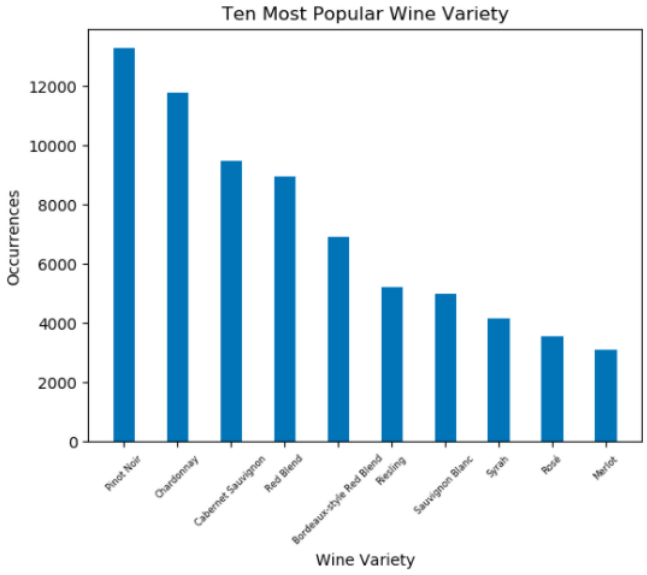

Figure 1. Most Popular Wine Variety

From Figure 1, we found that Pinot Noir, Chardonnay, Cabernet Sauvignon, Red Blend are the wine varieties that received most reviews. To get a more intuitive view of how the wine varieties are distributed, the fractions of the top wine varieties is drawn below as well.

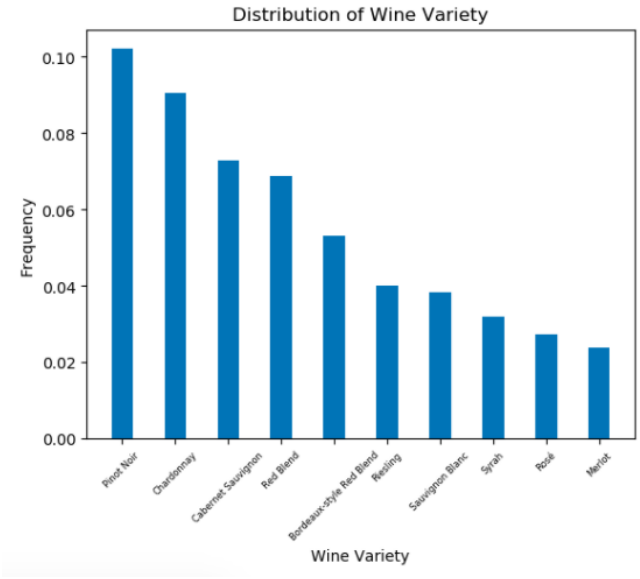

Figure 2. Distribution of Wine Variety

As shown in Figure 2, about 55% reviews comes from the top ten varieties and 45% reviews comes from the rest (697 varieties). Knowing that there is a total of 707 wine varieties in the dataset, we can conclude that the reviews are heavily distributed among the top ten wine varieties.

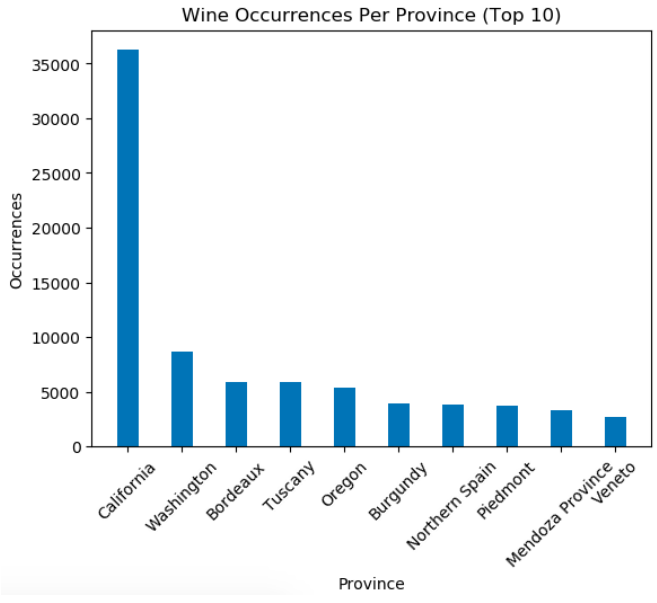

Figure 3. Most Popular Province for Wine Production

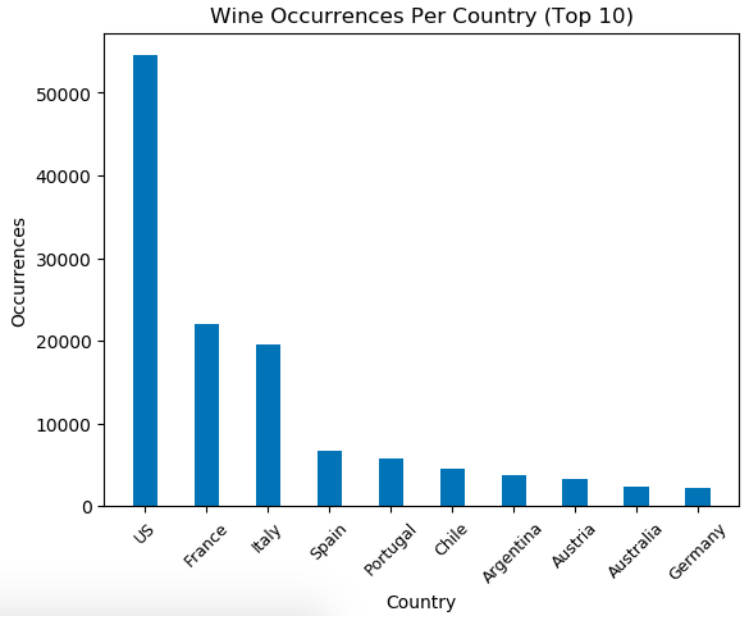

Figure 4. Most Popular Country for Wine Production

While a further exploration is performed on the data, US and California are the most popular country and the most popular state for wine production. Being the major wine supplier, nearly 42% wines in the dataset is found to be the US’ s wine production; California, on the other hand, produces almost one third of wines.

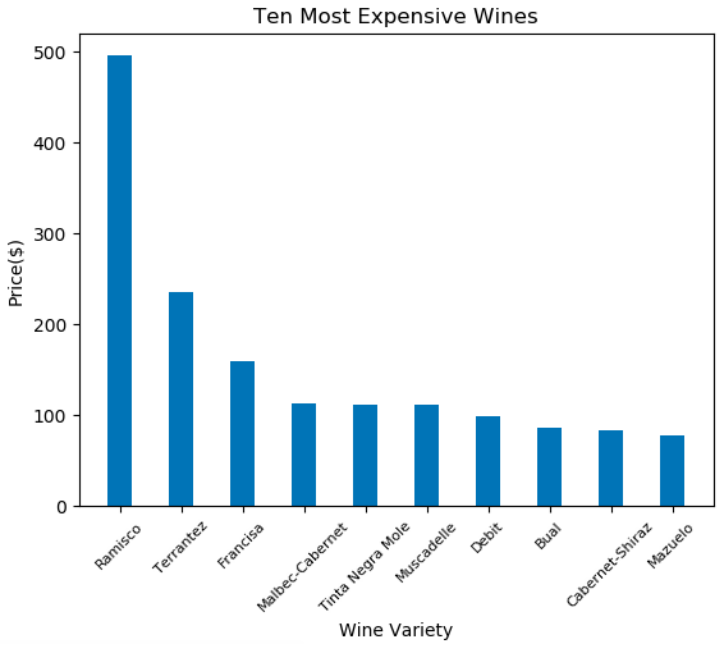

Figure 5. Ten Most Expensive Wine Varieties

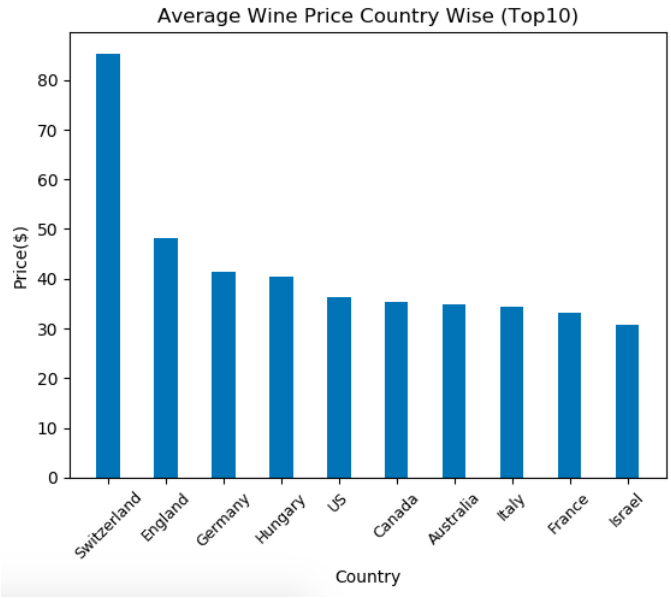

Figure 6. Average Wine Price Country Wise

In Figure 5, the bar charts show how the wine prices differ from one wine variety to another. An interesting finding is that the two most expensive wine variety have prices that are much higher than the other top ten most expensive varieties. Ramisco has a price of $495 and Terrantez for $236 while the rest of top ten averaging about $100.

In Figure 6, Switzerland produces the most expensive wine ($85.3) and the rest of the top ten countries produces wines about $40.

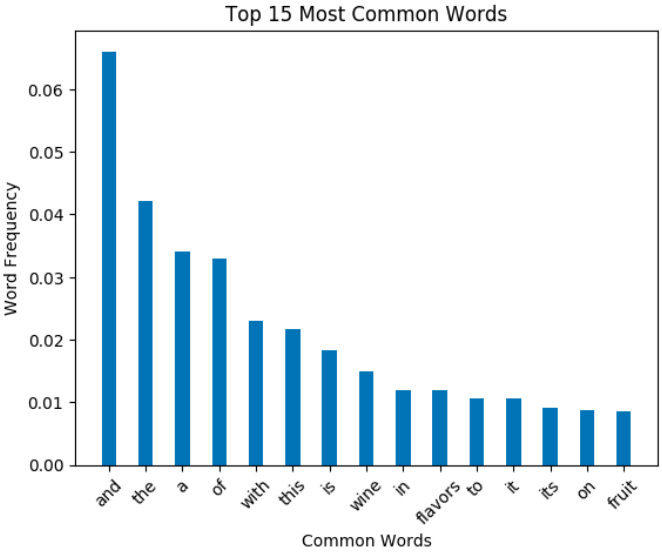

Figure 7. Most Common Words in Description

Since the only feature used for our prediction is the user's description in review. It is essential to count and pick the most common words a description consists of. As shown in Figure 7, lots of meaningless words such as "and", "the", "a", "of", "this" are counted as most common words. However, these meaningless words will affect our model's prediction accuracy negatively.

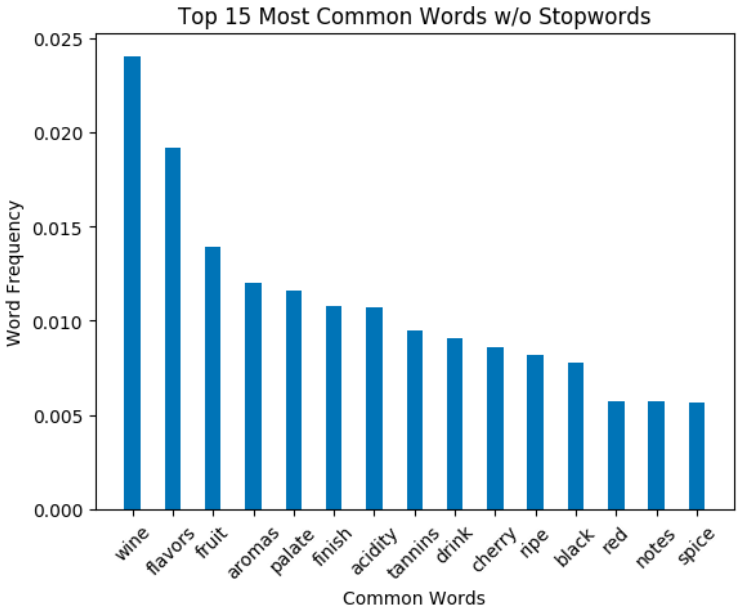

Figure 8. Most Common Words in Description w/o Stop words

To eliminate meaningless words in the most common words, Python's Natural Language Toolkit is applied to ignore the stop words. As shown in Figure 8, a more meaningful set of common words is obtained. These common words would make better candidates for our model's parameters.

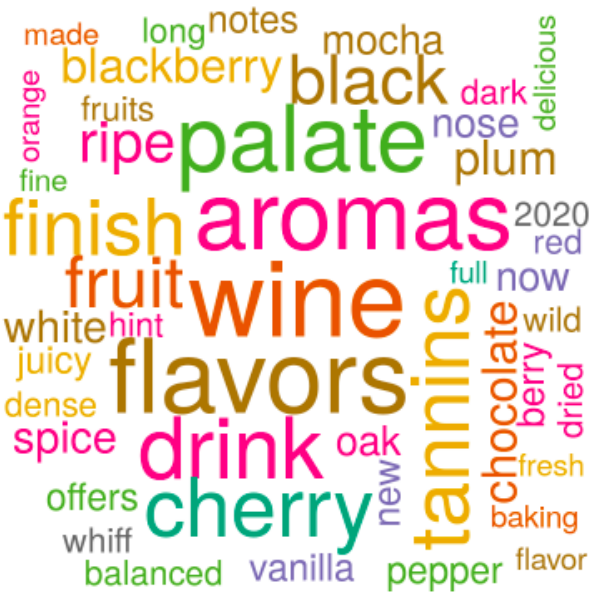

Figure 9. Word Cloud for Most Common Words

2. PREDICTIVE TASK

2.1 Task Description

The predictive task for our study on this wine review dataset is to predict a wine's variety (Pinot Noir, White Blend, Riesling, etc.). Upon a successful prediction, our model will be able to identify the wine's variety based on the description/review of certain wine. By preprocessing the dataset, we find that there are 707 unique wine varieties in the dataset. Essentially, our task is a multi-label text classification with a large number of labels.

2.2 Evaluation

To evaluate the performance of our models and make model selection, we first process and divide our data into three parts—training set, validation set and test set. When observing the dataset, we find that the entries in our input .json file is somehow grouped by rating points. Therefore, we randomly shuffle the entire dataset after we read the file, and then we divide the dataset via 60%, 20%, 20% split. The first 60% is for training, the second 20% is for validation, and the last 20% is for testing. It is also important to note that each entry in the dataset has exactly one label (i.e. variety). Since we have total of 707 wine varieties, we assign each variety an id in integer format (0-706), and create a dictionary called varietyDict to store each variety and its id. This makes it easier to evaluate the predictions.

Prediction results are evaluated by accuracy, i.e. 1 minus the Hamming Loss [2]:

$$HammingLoss(x_i, y_i) = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{xor(x_i, y_i)}{|L|}$$

Where |D| is the number of samples, |L| is the number of labels. x_i is the prediction and y_i is the ground truth. Since we look at 1 minus Hamming Loss, we want the value to be high to represent high accuracy.

2.3 Proper Models

Since our task is a multi-label classification task, the following three models might be suitable for this task—Naïve Bayes, Logistic Regression, and Multi-Class Support Vector Machine. It is important to address the advantages and disadvantages of each model here. Naïve Bayes classifiers are simple probabilistic classifiers based on applying Bayes' theorem with assumption that features are conditionally independent given the label. It is fast and simple. However, it can perform badly when the features are not conditionally independent. Logistic Regression is a regression model that converts the real-valued expressions into a probability value in [0,1]. It is a bit slower than Naïve Bayes but still relatively fast. It fits well with linear features. Multi-Class SVM is a more complex model. It tends to give higher accuracy but the computation time can be very long when the dataset is very large. In this project, we will compare all three models and find out which one suits the dataset best.

2.4 Baseline Model

The very first step of developing good models is to design a baseline model for future comparison. Since Naïve Bayes is the fastest and simplest out of the three models, our baseline model was implemented based on Naïve Bayes. For this baseline model, we will just use a simple *bag-of-words* feature with binary values. While all the reviews from the dataset are being processed, the number of times a word appears for each unique word will be counted. We then select the top 500 most common word and build a feature vector of 500 dimensions with binary values; i.e. [commonword0 in review, commonword1 in review ⋯ commonword499 in review]. Since our feature vector are made of binary values, we will use Bernoulli Naive Bayes (*sklearn.naive_bayes.BernoulliNB*). After training the baseline model on the training set, it gives a validation accuracy of 0.42636 (round to 5 significant digits).

2.5 Additional Features

Since our model in this predictive task makes predictions based only on review text, we choose to extract useful features only through the description/review text in the dataset.

The feature vector in our baseline model is just a vector of size 500 with binary values indicating whether a certain popular word is found in the input review text. Binary values may not give a good overview of the word's relevance. To further develop this feature vector, we decide to replace the binary values with TF-IDF (Term Frequency-Inverse Document Frequency) of each popular word in the given input review text. The TF-IDF is calculated as follow:

$$tfidf(word, review) = tf(word, review) \times \log(\frac{N}{df(word)})$$

Where $tf(word, review)$ is number of occurrences of $word$ in $review$. $df(word)$ is the number of reviews that contain $word$. N is the total number of reviews. Note that for calculating tfidf, we only consider the reviews across the training set.

Now our feature vector looks like this:

[tfidf(commonWord0, review), tfidf(commonWord1, review),⋯
tfidf(commonWord499, review)]

So far, we have been only looking at unigram only, i.e. we only look at a single word instead of word phrases. It will be useful to look at bigrams and trigrams as well. Take the review text of the first entry of the dataset as an example; the review text is:

"Aromas include tropical fruit, broom, brimstone and dried herb. The palate isn't overly expressive, offering unripened apple, citrus and dried sage alongside brisk acidity." Initially, we only look at the text by splitting it into single words, i.e. 'Aromas', 'tropical', 'fruit', etc. Now we want to include bigrams such as 'Aromas include', 'include tropical' and trigrams such as 'Aromas include tropical', and 'isn't overly expressive'. For bigrams and trigrams, we will use the same approach for feature vector design as for unigrams. We extract most common unigrams/bigrams/trigrams by using *ngrams* imported from *nltk.util*. When extracting popular unigrams, we find that removing stop words can be useful. This is because stop words such as 'the' 'is' 'and' are very common across the reviews, and if we only extract unigrams, these stop words are not representative at all. On the other hand, we choose not to remove stop words when extracting bigrams and unigrams, because word phrases such as 'is good' 'isn't overly expressive' can be very representative and we do not want to remove stop words such as 'is'.

Examples of uni-gram, bi-gram, tri-gram:
unigram (one word): 'is'
bigram (two words): 'is good'
trigram (three words): 'is very good'

In addition, how many most common unigrams/bigrams/trigrams we want to extract is a parameter we need to tune. Suppose we

want to keep top N1 most common unigrams, top N2 most common bigrams, and top N3 most common trigrams, in the end, we will have three important feature vectors as presented in Table 2. In the model selection process, we can freely use any of these 3 vectors, and we can also concatenate them together as a large feature vector. Our hypothesis is that concatenating any two or all of them together should give higher accuracy than using them individually.

Table 2. Summary of Features Vectors in Hand

Feature Vector	Format	Size
unigram	[tfidf(commonUnigram0, review), …, tfidf(commonUnigramN1-1, review)]	N1
bigram	[tfidf(commonBigram0, review), …, tf-idf(commonBigramN2-1, review)]	N2
trigram	[tfidf(commonTrigram0, review), …, tfidf(commonTrigramN3-1, review)]	N3

3. MODEL SELECTION AND OPTIMIZATION

3.1 Model Selection

First, we want to specify the models we want to select from. Naïve Bayes has three models, Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes. Since our feature vectors are all based on TF-IDF, which are fractional counts, Multinomial Naïve Bayes model is the appropriate Naïve Bayes model we want to consider. The second model we want to consider is the Logistic Regression model, and the third model we want to consider is the Multi-Class SVM. In this task, we will consider OneVsRest Classifier with LinearSVC.

Now we have three models in hand— Multinomial Naïve Bayes, Logistic Regression, and Multi-Class SVM (OneVsRest). We decide to do a simple evaluation on these three models using the same feature vector. The format of the unigram feature vector used to train the models is shown above in Table 2, and their corresponding accuracy on validation set is shown in Table 3. (accuracies are rounded to 5 significant digits):

Table 3. First-Round Evaluation

Model	Parameter	Validation Accuracy	Run time
Multinomial Naïve Bayes	default	0.51276	< 1min
Logistic Regression	default	0.55348	< 2min
Multi-Class SVM (OneVsRest)	default	0.54909	> 10min

As mentioned in 2.3, Naïve Bayes and Logistic Regression are relatively simple and fast models comparing to Multi-Class SVM. In our case, the OneVsRest Multi-Class SVM takes very long time to compute, and its accuracy performance is similar to Logistic Regression. Therefore, we choose to discard the OneVsRest Multi-Class SVM model and keep the other two models. Later we will evaluate the Multinomial Naïve Bayes model and the Logistic Regression models. We will also optimize them with testing different parameters and feature vector combinations.

3.2 Multinomial Naïve Bayes Model

3.2.1 Implementation

To implement the Multinomial Naïve Bayes Model, we will import the *MultinomialNB* from scikit-learn library. We will use different feature vector combinations to build out X_train and y_train. Then we will train our model by fitting X_train and y _train. Our model is then ready for validation.

3.2.2 Optimization

To optimize our multinomial Naïve Bayes model, we want to create a parameter space for each parameter we want to tune. All candidates for each parameter is presented in Table-3.

Table 4. Parameter Candidates for Naïve Bayes

Parameter	Candidates
Additive smoothing parameter alpha: α	[0.00001, 0.0001, 0.001,0.01,0.1,1.0,10.0,100.0]
Unigram/bigram/trigram feature vector size N1, N2, N3 in tuple (N1, N2, N3)	[(500,100,100), (500,200,200), (500,300,300), (500,400,400), (500,500,500)]
Feature vector combination	[unigram, unigram+bigram, unigram+trigram, unigram+bigram+trigram]

All parameters are evaluated in the optimization process by following steps:

1) Set additive smoothing parameter alpha as default
2) Set N1= 500 N2=500 N3=500 as default value
3) Train the model with all possible feature vector combinations and use the combination that returns the best validation accuracy
4) Adjust the feature vector size N1/N2/N3 of the features presented in the best feature vector combination returned by 3) and evaluate the validation accuracy. Return the model with the highest validation accuracy.
5) Tune the additive smoothing parameter alpha on the model returned by 4) to get the final optimized model.

Results and issues with overfitting will be discussed later in Section 5.

3.3 Logistic Regression Model

3.3.1 Implementation

To implement the Logistic Regression Model, we will import the *LogisticRegression* from scikit-learn library. We will use different feature vector combinations to build X_train and y_train. Then we will train our model by fitting X_train and y _train. We use l2 norm as the norm used in penalization because it is more stable. All other parameters are set to default except C, the inverse of regularization strength, is being optimized during validation. Our Logistic Regression model is then ready for validation.

3.3.2 Optimization

To optimize our Logistic Regression model, we want to create a parameter space for each parameter we want to tune. All candidates for each parameter is presented in Table-4.

Table 4. Parameter Candidates for Logistic Regression

Parameter	Candidates
Inverse of regularization strength: C	[0.0001, 0.0001, 0.001,0.01,0.1,1.0,10.0,100.0]
Unigram/bigram/trigram feature vector size N1, N2, N3 in tuple (N1, N2, N3)	[(500,100,100), (500,200,200), (500,300,300), (500,400,400), (500,500,500)]
Feature vector combination	[unigram, unigram+bigram, unigram+trigram, unigram+bigram+trigram]

All parameters are evaluated in the optimization process by following steps:

1) Set regularization parameter C as default
2) Set N1= 500 N2=500 N3=500 as default value
3) Train the model with all possible feature vector combinations and use the combination that returns the best validation accuracy
4) Adjust the feature vector size N1/N2/N3 of the features presented in the best feature vector combination returned by 3) and evaluate the validation accuracy. Return the model with the highest validation accuracy.
5) Tune regularization parameter C on the model returned by 4) to get the final optimized model.

Results and issues with overfitting will be discussed later in section 5.

4. RELATED LITERATURE

Our study is essentially a multi-label text classification problem. The wine review dataset we selected is an existing dataset from Kaggle as mentioned above. Text classification is a very popular topic in the academic field. There are many other datasets similar to the one we study. Similar datasets that have been studied in the past and now including Beer reviews form BeerAdvocate[3], wine reviews from CellarTracker[4], films ratings from The Netflix Prize Dataset[5] and product reviews from Amazon[6] etc. Some text processing methods are widely used by researchers on these datasets to predict ratings. Their feature extraction methods on this type of data are inspiring and helpful for our feature selection as well as feature design in the future text classification task.

Aside from data processing, we want to discuss some similar prediction tasks related to our study. An article published by New York University addresses some traditional and state-of-the-art methods currently employed to study this type data [7]. They discuss a number of datasets including Yelp Review Full Dataset, Amazon Review Full Dataset, Yahoo Answers Dataset, etc. These datasets all promote a text classification task. The traditional methods they mentioned align with our approach. They suggest using Bag-of-words and its TF-IDF and Bag-of-ngrams and its TF-IDF as well. One additional traditional method they mention is to use Bag-of-means on word embedding. It is an experimental model that uses k-means on word2vec learnt from the training subset of each dataset, and then uses these learnt means as representatives of the clustered words. They take into consideration all the words that appeared more than 5 times in the training subset. They also introduce some deep learning methods. One of them is the Word-based ConvNets. The experimental Bag-of-means method perform very well on the selected datasets. It performs better than traditional methods by several percent.

Another project conducted by Stanford University also focuses on text classification on multi-label datasets [8]. They study four datasets, which are all similar to the dataset we are studying. One of them is the Bookmark Dataset. It contains metadata for bookmark items such as the URL of the web page, a description of the web page, and the labels assigned to it. They use similar models to ours, including Naïve Bayes and SVM. They also include methods such as k-Nearest Neighbor and Random Forests. They study datasets with both low feature to label ratio and high feature to label ratio. Their SVM seems to work well and efficient. This is different from our findings. Our SVM takes too long to compute. We may want to improve our SVM model in the future. The Naïve Bayes model they implemented have relatively low accuracy compared to KNN model and SVM model, which aligns with our observation. KNN seems to give very good result on datasets with many labels (low feature to label ratio). We may want to implement KNN in future text classification projects.

5. RESULTS AND CONCLUSIONS

5.1 Optimization Results

The results we get by following the optimizations steps from 1) to 3) for Multinomial Naïve Bayes and Logistic Regression are presented in Table-5.

Table 5. Results from Different Feature Vector Combinations

Feature Vector Combinations	**Multinomial Naïve Bayes validation** accuracy ($\alpha = 1.0$) N1=500 N2=500 N3=500	**Logistic Regression validation accuracy** ($C = 1.0$) N1=500 N2=500 N3=500
unigram	0.51276	0.55348
unigram+bigram	0.56305	0.59090
unigram+trigram	0.54610	0.58178
unigram+bigram +trigram	0.58842	0.61951

So far, we observe that the best feature vector combination is to concatenate unigram vector, bigram vector and trigram vector all together. We decide to use this unigram+bigram+trigram feature vector as our final feature vector to tune our final model's parameter. Results of the rest two optimization steps are presented in Table 6.

Table 6. Results from Different Feature Vector Size

N1, N2, N3 value	Multinomial Naïve Bayes validation accuracy with best $\alpha = 0.01$	Logistic Regression validation accuracy with best $C = 0.1$
500, 100, 100	0.57029	0.60743
500, 200, 200	0.58164	0.61022
500, 300, 300	0.62410	0.65192
500, 400, 400	0.60844	0.62351
500, 500, 500	0.59707	0.62138

We observe that Logistic Regression always perform better than multinomial Naïve Bayes. The final optimized model is the Logistic Regression model with regularization parameter C=0.1. The final feature vector combination is the top 500 most common unigrams TF-IDF vector + top 300 most common bigrams TF-IDF vector + top 300 most common trigrams TF-IDF vector. The final test result of our optimized model is presented in Table 7.

Table 7. Test Result of Final Optimized Model

Model	Feature Vector Combination	Validation Accuracy	Test Accuracy
Logistic Regression C=0.1	unigram+bigram+trigram N1=500, N2=300, N3=300	0.65192	0.65311

5.2 Conclusion and Additional Notes

Our predictive task is a multi-label text classification task dealing with a large number of labels (wine variety). Considering there are 707 different wine varieties appeared in the dataset, using the OneVsRest Multi-Class SVM model is rather complex and requires unreasonable long computation time. Since there are hundreds of classes to be classified, it will be an extremely time consuming process to get the result from a Multi-Class SVM model. Therefore, the Multi-Class SVM failed the task efficiency wise. Looking at the results of the other two models shown in Table 6, we observed that Logistic Regression always performs better than Naïve Bayes on our dataset. This implies that our features are linear, as logistic regression works very well with linear features. Speaking of feature selection, our hypothesis is that including unigrams, bigrams, and trigrams all together in out feature vector will give the best performance. Our optimization result validates our hypothesis. When combining TF-IDF vectors of unigrams, bigrams and trigrams all together, a highest validation accuracy among all approaches is obtained. This also shows that all three of our feature representations work well with our task; the key is to select the best combination of them. In addition, the size of the unigram TF-IDF vector, N1, the size of the bigram TF-IDF vector, N2, and the size of the trigram TF-IDF vector N3, are tuned to 500, 300, 300 to give the most suitable feature vector. After building the most suitable feature vector, we tune the Logistic Regression model's hyper parameter C, the inverse of regularization strength. In this case, a smaller value means stronger regularization. When C = 0.1, we achieve the best validation accuracy of 0.65192. Then this model was sued to predict the test set (20% of the whole dataset) which we split earlier, and resulted in a test accuracy of 0.65311 (Higher accuracy on test set than validation set). So, we can conclude that our final model does not over fit.

6. REFERENCES

[1] Thoutt, Z., 2017. *130k Wine Reviews.* Kaggle. https://www.kaggle.com/zynicide/wine-reviews

[2] Kaggle. 2017. *Hamming Loss.* Kaggle. https://www.kaggle.com/wiki/HammingLoss

[3] Leskovec, J. 2014. *Web data: BeerAdvocate reviews*. Stanford University. http://snap.stanford.edu/data/web-BeerAdvocate.html

[4] Leskovec, J. 2014. *Web data: CellarTracker.* Stanford University. http://snap.stanford.edu/data/web-CellarTracker.html

[5] Netflix. 2017. *Netflix Prize data*. Kaggle. https://www.kaggle.com/netflix-inc/netflix-prize-data

[6] McAuley, J., 2016. *Amazon Product Data*. UCSD. jmcauley.ucsd.edu/data/amazon/

[7] Zhang, X., Zhao, J., LeCun, Y. 2015. *Character-level Convolutional Networks for Text Classification*. Cornell University Library. https://arxiv.org/abs/1509.01626

[8] Nayak S., Ramesh R., Shah S. 2013. *A Study of multilabel text classification and the effect of label hierarchy.* Stanford University. https://nlp.stanford.edu/courses/cs224n/2013/reports/nayak.pdf