Predicting Wine Review Rating Using Sentiment Analysis and Spark

Zheda Mai

University of Toronto | Zheda.mai@mail.utoronto.ca

Table of Contents

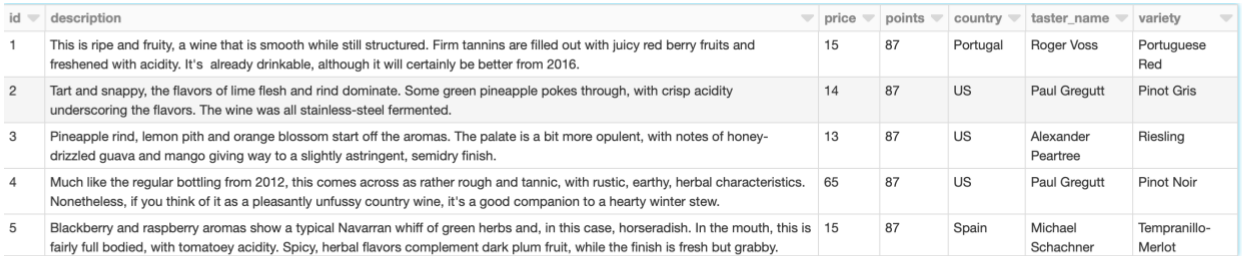
Abstract

There is immense research being done in Data mining in the field of sentiment analysis to predict the rating given to an item. In this project, we will focus on predicting the ratings of wines from experts based on the information of wines. We explored different data embedding techniques including Word2Vec and TF-IDF as well as different machine learning methods including Logistic Regression, Multi-Layers Perceptron Classifier, etc. We were able to get 83.3% F1 score and 69% R2 score on testing data.

Introduction

Wine rating and review have become more and more important to both customers and wineries. Since it has been common for customers to check the rating and review of wine before purchasing, ratings and reviews are becoming crucial metrics for wineries and a wine rating predicting model can help wineries to better adjust their production and marketing strategy.

In this project, we are going to use the Wine Reviews dataset from Kaggle [1]. The data was scraped from WineEnthusiast, world's leading source for wine, on November 22nd 2017 and have 130k reviews. Each data entry contains 14 columns including points, price, description and other 11 different information about the wine and the taster.



id	description	price	points	country	taster_name	variety
1	This is ripe and fruity, a wine that is smooth while still structured. Firm tannins are filled out with juicy red berry fruits and freshened with acidity. It's already drinkable, although it will certainly be better from 2016.	15	87	Portugal	Roger Voss	Portuguese Red
2	Tart and snappy, the flavors of lime flesh and rind dominate. Some green pineapple pokes through, with crisp acidity underscoring the flavors. The wine was all stainless-steel fermented.	14	87	US	Paul Gregutt	Pinot Gris
3	Pineapple rind, lemon pith and orange blossom start off the aromas. The palate is a bit more opulent, with notes of honey-drizzled guava and mango giving way to a slightly astringent, semidry finish.	13	87	US	Alexander Peartree	Riesling
4	Much like the regular bottling from 2012, this comes across as rather rough and tannic, with rustic, earthy, herbal characteristics. Nonetheless, if you think of it as a pleasantly unfussy country wine, it's a good companion to a hearty winter stew.	65	87	US	Paul Gregutt	Pinot Noir
5	Blackberry and raspberry aromas show a typical Navarran whiff of green herbs and, in this case, horseradish. In the mouth, this is fairly full bodied, with tomatoey acidity. Spicy, herbal flavors complement dark plum fruit, while the finish is fresh but grabby.	15	87	Spain	Michael Schachner	Tempranillo-Merlot

Figure 1: Snapshot of wine dataset 130k

The reason we use this data is the diversity of feature types in this dataset. As we can see in Figure1, for example, the description is a **textual** type feature, points and price are **numerical** features, country and tester are **categorical** features. Using hybrid features has been very popular in machine learning. So, picking this dataset not only can help us be familiar with different feature extraction methods in Spark but also allows us to experience more complex machine learning pitfalls we have not met before.

Moreover, we use PySpark, a Python tool of Apache Spark, in this project for all the data processing and analysis. We use the Notebook and clusters provided by Databricks community edition for code running, visualization and documentation.

Problem Definition

We are trying to use other information about the wine to predict its rating. As we have the rating data, this will be a supervised learning problem. The scoring scale of the WineEnthusiast [2] is shown below.

- 98–100 – Classic
- 94–97 – Superb
- 90–93 – Excellent
- 87–89 – Very good
- 83–86 – Good
- 80–82 – Acceptable

We are going to conduct both classification and regression of the rating prediction. In order the simplify the problem, we decided to have three categories, (Fair: 80-86, Good: 87-93, Excellent: 94-100). Since this is a multi-class classification problem, we will use four multi-class metrics provided by Spark to evaluate the performance of the classification model.

- F1
- Accuracy
- WeightedRecall
- WeightedPrecision

In terms of regression, we will use Mean Square Error(MSE) and R2 for evaluation.

Related Work

The Wine Review dataset on Kaggle has been widely used in a lot of data science projects. Most of the projects were trying to do the sentiment analysis using the description to predict the price, variety or rating [3][5]. Some project focused on using numerical and categorical features such as country, variety and price to predict rating [4].

Our project is different from others in the following ways

- We used textual, numerical and categorical features rather than just one or two of them.
- We did both regression and classification and used more metrics for evaluations
- We used feature selector such as Chi-Square to select more informative features
- We used Word2Vec to embed the textual feature while most projects only use term frequency or TF-IDF
- All the projects we can find online used local machines for processing and we care using Spark on clusters.

Data Exploration & Feature Selection

Aimed to obtain good features for our model we try to explore further on the properties on the dataset.

Country:

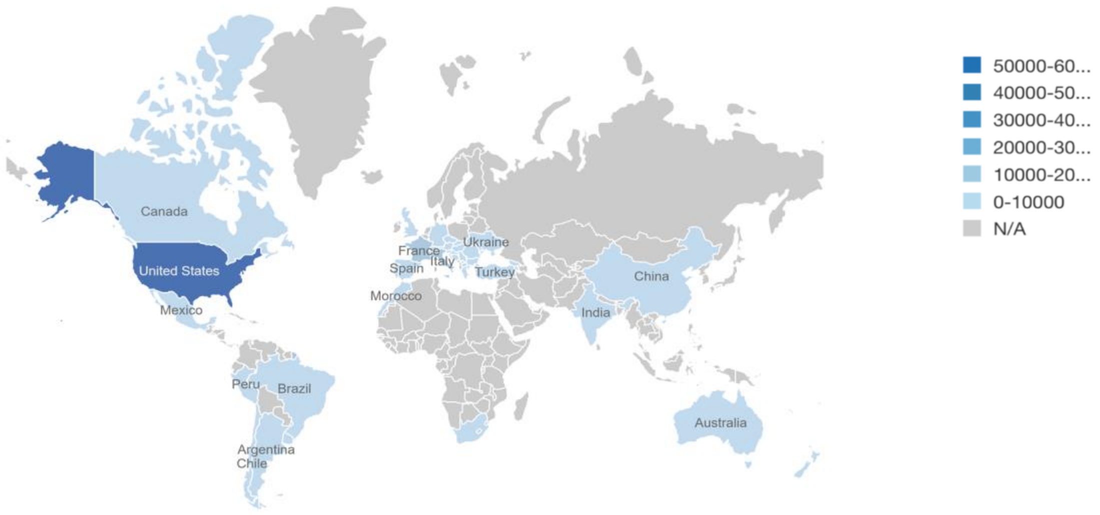


Figure2: Distribution of wine based on country

As we can see from Figure2, most of the wines in this dataset come from US. Other major countries and areas include Europe, Canada, South American, Australia and China.

Points:

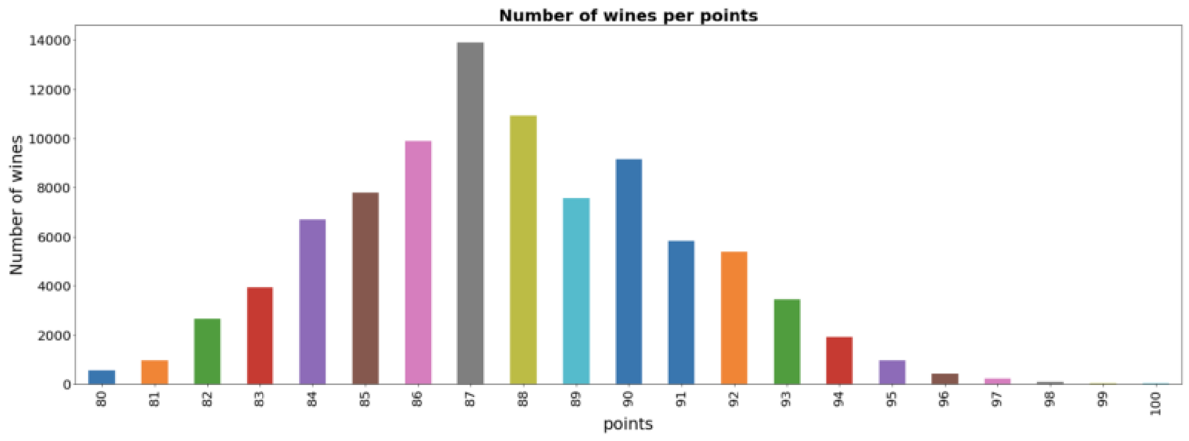


Figure3: Distribution of Points of all reviews

Figure3 shows the distribution of points, it mostly follows Gaussian Distribution and most wines are between 83-93.

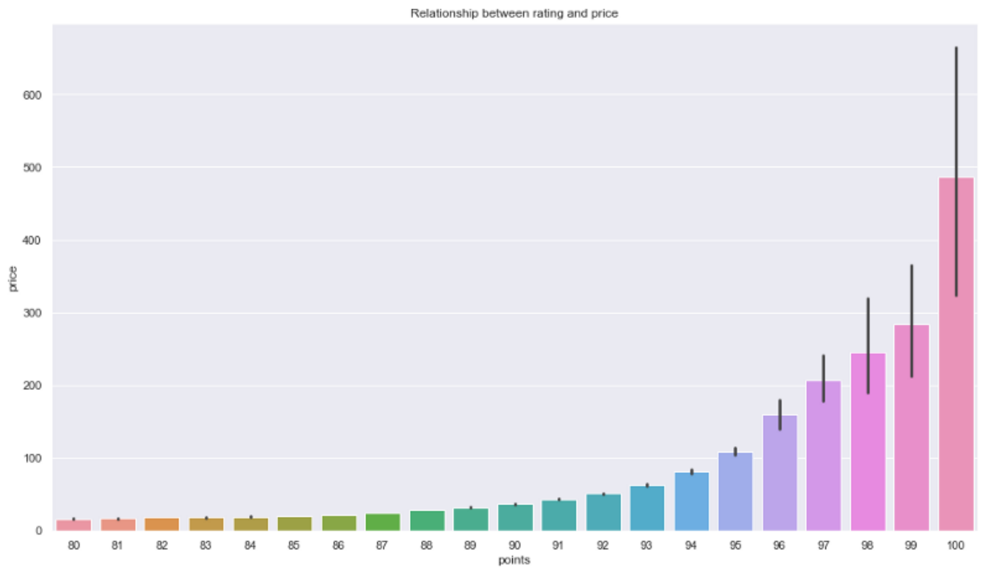


Figure4: Relationship between Price and Rating

Figure4 is a plot of the average price for each rating, it shows a clear relationship between the price and the rating. This is a clear evidence that we should use price as one of the features

Variety:

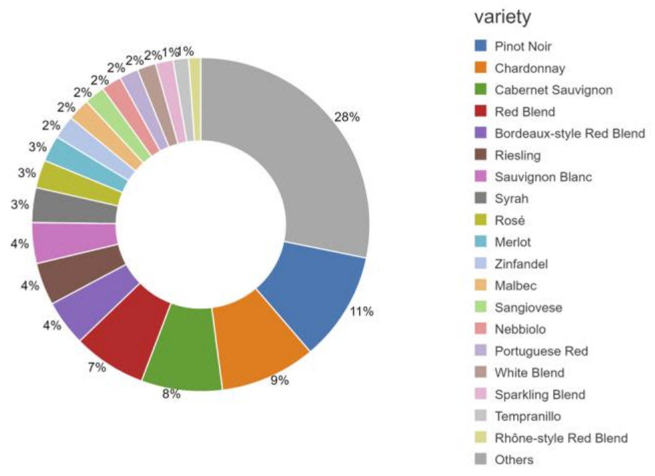


Figure5: Distribution of variety

As we can see in Figure5, there are 4 varieties exceed 5% with the largest 11%, other varieties are distributed evenly.

Word frequency:

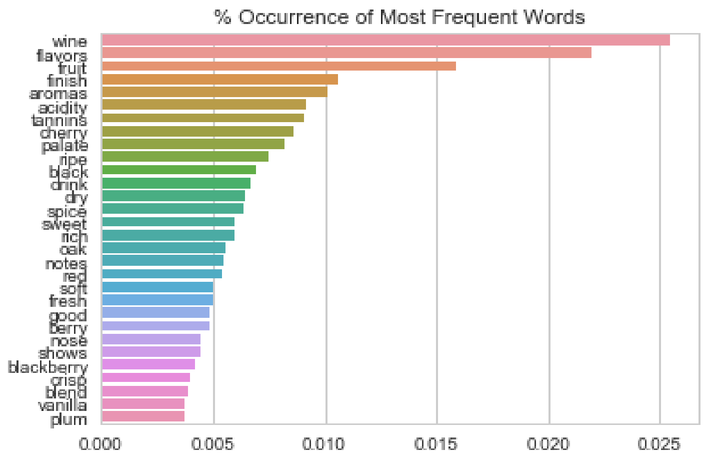


Figure6: Occurrence of most frequent words

As we can see in Figure6, the top 3 frequent words are wine (0.026), flavors (0.022) and fruit (0.0155).

Based on the exploration, we decided to use the following features in our project.

- Textual feature
    - Description
- Categorical feature
    - Country, Variety
- Numerical Feature
    - Price

Design

Our development pipeline contains 5 major steps:

1. Data Preprocessing
    - Remove all the data entries which contain Null value in columns of ['description', 'price', 'points', 'country']. After this step, our data size is reduced from 129971 to 120916.
    - Lowercase text in 'description'
    - Select the top 20 countries and group other countries as 'Other'

2. Features Extraction
    - Textual feature
        - Tokenization
        - Remove Stop-words
        - Feature Extraction
        - TF-IDF
        - Unigram-Word2Vec
        - Bigram-Word2Vec

    Instead of using simple Tokenizer, we use RegexTokenizer with the pattern '\\W', which helped us remove all the non-word tokens including punctuations. Also, we used three different word embedding methods. TF-IDF measures the importance of a term to a document using Term Frequency and Document Frequency. Unigram-Word2Vec takes sequences of words representing documents and trains a Word2VecModel. The model maps each word to a unique fixed-size vector. The difference between Unigram-Word2Vec and Bigram-Word2Vec is Bigram-word2Vec maps every two consecutive words to a unique fixed-size vector.

- Categorical Feature
    - Encode string label to label indices
    - One-hot encode the indices generated above

  We firstly used StringIndexer to encodes all the categorical variables (country and variety) in to label indices. Then we used One-Hot-Encoder Estimator to convert the label indices to one-hot format without assuming a natural order by assigning a binary matrix to represent the presence of each variable.
- Numerical feature

  Price is the only numerical feature and we simply normalize it
- Target

  Since the rating is numerical and we will use Bucketizer to turn the numerical number to label.

3. Feature combination and selection

   Since we have a lot of features and we want to select features that are informative about our target labels. We used Chi-Squared test of independence to decide which features to choose. The reduction of dimension not only allows us to train the model with less data but also can combat overfitting in the model.

   Also, we will use VectorAssembler to concatenate different types of feature together.

Below is the list of all the combinations of features

- Textual Feature only: TF-IDF
- Textual Feature only: Unigram-Word2Vec
- Textual Feature only: Bigram-Word2Vec
- Categorical & Numerical Feature only
- TF-IDF_Num_Cate: TFIDF, Categorical & Numerical Together
- TF-IDF_Num_Cate_Chi TFIDF, Categorical & Numerical, used Chi-Square to select Top-K best feature
- UnigramWord2Vec_Num_Cate: UnigramWord2Vec, Categorical & Numerical
- BigramWord2Vec_Num_Cate: BigramWord2Vec, Categorical & Numerical

We use all combinations individually on Logistic Regression to do the classification to see which combination performs better

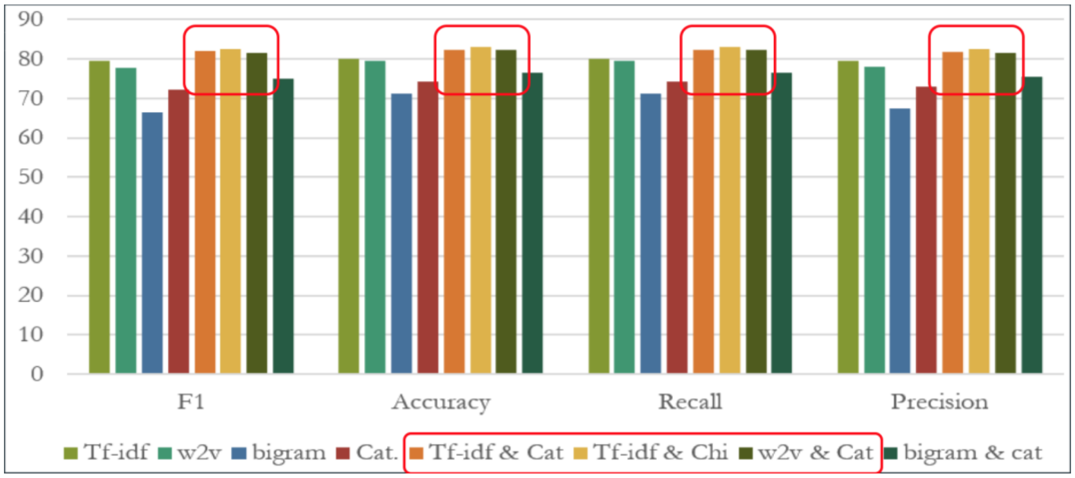


Figure7: Performance of different combinations of features on Logistic Regression

As shown in Figure7, three combinations always outperformed others. They are TF-IDF_Num_Cate, TF-IDF_Num_Cate_Chi and UnigramWord2Vec_Num_Cate. These are will features will be used in the model experiments.

Model Experiment:

Classification

We use four different models in our classification experiments including:

- Logistic Regression
- Naive Bayes
- Random Forests
- Multi-layers Fully Connected Neural Network

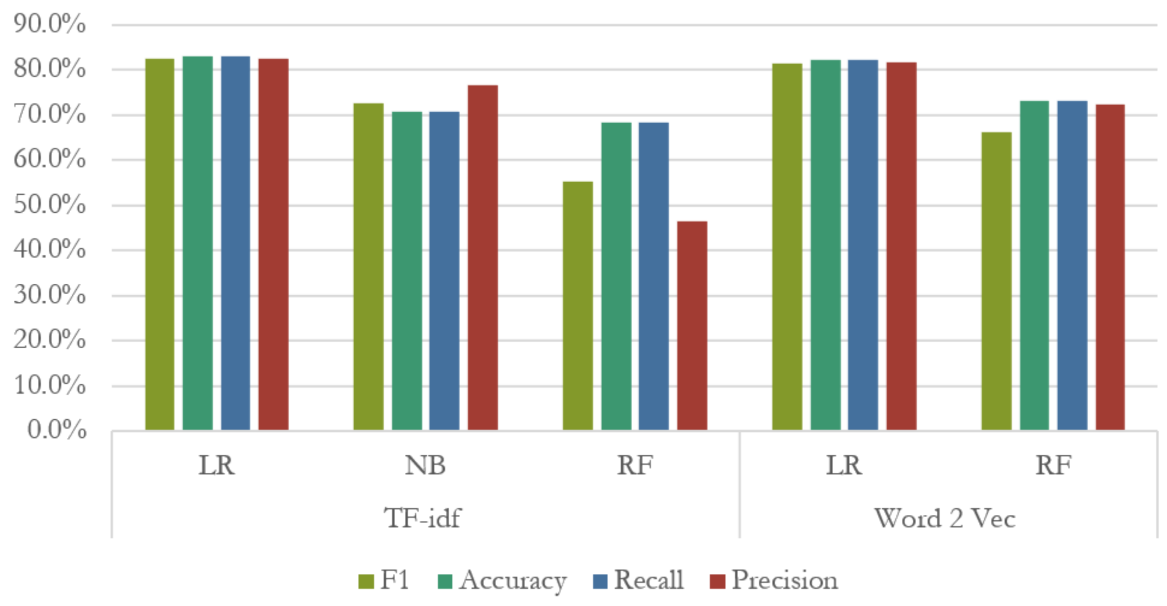


Figure8: TF-IDF and Word2Vec performance on different models

It's worth mentioning that Naïve Bayes cannot be easily applied on word2vec metrics without constructing clusters for each word. Thus, only logistic regression and random forest were implemented for Word2Vec. As we can see in Figure8, Logistic Regression outperformed other models for both TF-IDF and Word2Vec. If we compared the performance of TF-IDF and Word2Vec on Logistic Regression, they are very similar for all four metrics. They are all around 81% -83%.

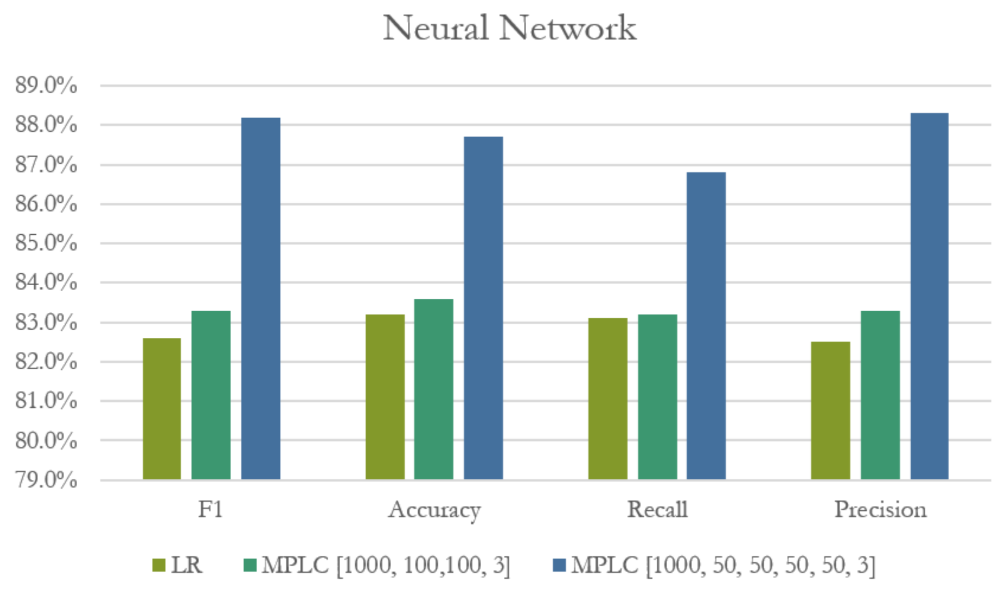


Figure9: Comparison between MLPC and Logistic Regression

We also implemented MLPC(MultilayerPerceptronClassifier). As we can see in Figure9, a MLPC with two hidden layers with 100 neurons in each layer has already outperformed our best model Logistic Regression. If we used the same number of neurons deepen architecture, the performance is much better. But since there is no GPU in the free Databricks clusters, for MLPC with 2 hidden layers, it took 2 hours to run and with 4 hidden layers, it took 5 hours to run. So, we still used Logistic Regression for hyper-parameters tuning as it only took around 13 seconds to run.

Regression

We used three models in regression

- Gradient Boosting,
- Decision Tree
- Linear Regression

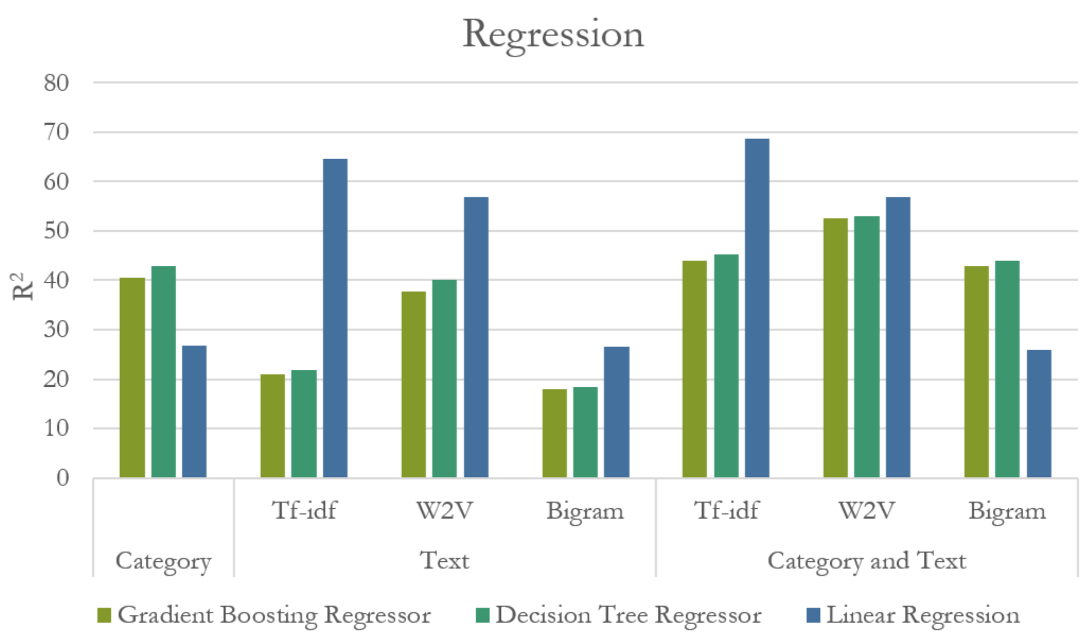


Figure10: Regression performance of features on different models

It's clear that Linear Regression outperformed the others significantly except with the BigramWord2Vec_Num_Cate. The best performance is TF-IDF_Num_Cate with Linear Regression which give us around 0.7 R2 score. Also, we found that when we combined different features together, it always performed better than the individual feature.

Cross Validation and Hyper-parameters tuning

As Logistic Regression and Linear Regression performed the best with TF-IDF_Num_Cate_Chi, we will use these settings for our hyper-paramers tuning. We firstly created the pipeline from preprocessing all the way to the machine learning model. Then we use ParamGridBuilder to construct a grid of parameters to search over. Lastly, we used 5-folds CrossValidator to conduct the tuning to obtain more unbiased result. To simplify the problem and save time, we only test 2 potential parameters for each hyper parameter.

Hyper-Parameters	Logistic Regression		Linear Regression	
HashingTF feature number	5000		5000	
regParam/max iteration	0.001		10	
Top-K Features from Chi-square	2000		2000	
F1 score/ R2 score	Before:0.81	After: 0.83	Before: 0.687	After: 0.693

Table1: Hyper-Parameters tuning result

Table1 shows the hyper-parameters we tuned and the best number for each hyper-parameter. After the tuning, the performance of Logistic Regression model increased by 2% and Linear Regression increased by 0.87%.

Discussion and Findings

1. Metrics selection

We chose F1, Accuracy, WeightedRecall and WeightedPrecision as our metrics. As we can see in Figure3, the distribution of rating is a Gaussian-like distribution, so most of the ratings will be in the middle. Using accuracy only does not reflect the true performance of the model. With four different metrics, each addresses different aspect, we can have a more comprehensive view of the performance of the model.

2. Combine textual, categorical and numerical features together performed better than a single type of features.

As we can see in Figure7, when we combine different types of features together, they outperformed individual type of features. This makes sense, as different types of features focus on different aspects of a problem. With more features, the classifier can pay attention to broader areas.

3. Using Feature Selector

We used Chi-Square test to select top-K best features. The reduction of dimension not only allows us to train the model with less data but also can combat overfitting in the model. So using feature selector in our project always give the better result comparing with combination without feature selector. One drawback of feature selector is it took a fairly long time to run as it needs to calculate the Chi-Square test for each feature.

4. Word Embedding methods performance.

TF-IDF and UnigramWord2Vec performed equally and BigramWord2Vec was worse. As we know, Word2Vec is better than TF-IDF in most NLP problems. I think the potential reasons that Word2Vec did not outperform TF-IDF are:

- In Spark, we need to train our Word2Vec model and our text corpus is fairly small compared with other NLP problems. The average number of tokens in each description, if we remove stop-words and punctuations, is around 35 and the number of distinct words in this corpus is 1000. This amount of data is not enough to train a good Word2Vec model.
- BigramWord2Vec performed poorly is because, if we used two words as a vector, we basically shrunk the corpus by half again. Given the small corpus we have, halving the corpus definitely could not give good result.

5. Models Selection

In the classification problem, MLPC is the best model with 88% F1(has potential to be better), however, it took fairly amount of time to train without GPU. Logistic Regression is the second best (83% F1) with only around 13 seconds to train. In the regression problem, Linear Regression gave the best result (69% R2).

Challenge

Working in PySpark environment introduces some limitation of this project. Firstly, the limitation of API we can use. For example, there is no lemmatization API we can use. Another example is the limitation of word embedding methods. If we work locally, we can use more embedding methods such as GloVe, WordRank, etc. Also, clusters provided by Databricks community edition lack of GPU, which prevents us from using deep learning models in our project.

Future Improvement

1. Using more features, for example, region, province, title and winery, etc. Since there are more features in the dataset, we can leverage those features. But potential problem is increasing the number of features without increasing the dataset size may lead to overfitting. But we can combat overfitting by increasing the regularization parameters or use Chi-Square to select top-K features.
2. Since the scrapping script of this dataset was released by the author [1], we can scrape more data to improve the training.
3. Using lemmatization. There is no lemmatization tool in PySpark, so we did not lemmatize the description data. But I noticed if we use Scala, there is an external NLP package that can be integrated into Apache Spark for more advanced NLP tools [6]
4. Better hyper-parameters tuning. To simplified the problem and saved more time, we only tested 2 values for each hyper-parameter. But theoretically, we should do a completed coarse tuning then find out the best interval and continue to do a fine tuning in that interval and repeat again. So, if we have more time, we can follow this methodology for a better tuning.
5. Use more complex models, we used very basic traditional machine learning models in this project. But as we all know, deep learning is beating most of the conventional models. 5 months ago, Databricks released a new deep learning pipeline [7], which we can take advantage of to yield a better result. For example, we can use LSTM RNN for this problem.

Reference

[1] zackthoutt, "Kaggle," [Online]. Available: https://www.kaggle.com/zynicide/wine-reviews.

[2] W. Enthusiast, "Wine Enthusiast," [Online]. Available: https://www.wine-searcher.com/critics-17-wine+enthusiast.

[3] O. Goutay, "Towards Data Science," 4 6 2018. [Online]. Available: https://towardsdatascience.com/wine-ratings-prediction-using-machine-learning-ce259832b321.

[4] L. E. Bolstad, "Medium," 18 10 2018. [Online]. Available: https://medium.com/@larserikbolstad/predicting-wine-ratings-using-machine-learning-aa64167d25b9.

[5] M. Gu, "Towards Data Science," 12 2 2019. [Online]. Available: https://towardsdatascience.com/predicting-wine-quality-using-text-reviews-8bddaeb5285d .

[6] D. Talby, "Databricks Blog," 19 10 2017. [Online]. Available: https://databricks.com/blog/2017/10/19/introducing-natural-language-processing-library-apache-spark.html.

[7] Databricks, "Github," [Online]. Available: https://github.com/databricks/spark-deep-learning.

Code Access

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8514324022161000/2053482634820665/383463452256204/latest.html?fbclid=IwAR0L0-tBE0-5V__O48rALD4wqudwp8MSL1BTwSf-g_D63lc5xIgapWMs9es