CS 573 Project: Predict Wine Quality

Lucía Ostolaza
lostolaz@purdue.edu
Department of Computer Science, Purdue
West Lafayette, Indiana, USA

Elena Gómez
gomez233@purdue.edu
Department of Computer Science, Purdue
West Lafayette, Indiana, USA

Vinitha Marupeddi
vmaruped@purdue.edu
Department of Computer Science, Purdue
West Lafayette, Indiana, USA

Danyang Wang
wang4589@purdue.edu
Department of Statistics, Purdue
West Lafayette, Indiana, USA

ABSTRACT

A consumer is always guided by the quality of the product when making purchase. However, product quality certification in the wine industry is a time-consuming and cost-intensive process for manufacturers. Therefore, machine learning (ML) has become an essential tool for replacing human tasks in modern wine production.[2] By automating the process of wine quality prediction, ML saves both the resources and time for wine-making businesses. [3]

Predicting wine quality using machine learning techniques is becoming increasingly popular today.[1] People build algorithms to predict the difference between cheap and expensive wines. There are many educational step-by-step guides using open-source wine quality prediction datasets that show how to use ML for wine quality prediction. In this project, we decided to do a detailed overview of techniques people used and choose our own set of algorithms to tackle this issue.

Figure 1: Wine

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

KEYWORDS

classification, wine quality, non-parametric algorithms, Bayes, Logistic regression

1 INTRODUCTION

Wine has always had a large impact on society, showing up in religious rituals and cultural festivities. Polytheistic religions had different gods, such as Dionysus, to honor this drink while monotheistic religions, such as Christianity, treat wine as an important symbol for Jesus' blood. Since its discovery thousands of years ago, humans have been changing the way they make this alcoholic beverage, but data science has helped the wine industry improve the quality of their wine and better understand why some are better than others.

There are three main characteristics that differentiate wine: smell, flavor, and color. These are affected by many factors such as the grape itself and the fermentation time, which lead to a differences in properties like pH, citric acid levels, and density. By analyzing them we could quantify the different wines on a scale of 1 to 10 to use in classification. The dataset we use is found on Kaggle: https://www.kaggle.com/datasets/yasserh/wine-quality-dataset.

The objective of our project is to find the best model to predict the wine quality. To achieve it, we applied different algorithms to datasets to create models.

Before starting to apply the machine learning algorithms to predict wine quality, we conducted an exploratory data analysis and data preprocessing. We conducted the first task with visualization techniques. They will allow us to have more information about our data and understand each factor better.

To train the models, we are going to create different datasets. The objective is to develop more efficient algorithms by training each machine learning algorithm with all the datasets. In order to do that we use preprocessing techniques such as the following:

- Remove outliers and/or missing values, create dummies, remove correlated variables, transform variables.
- PCA, SVD and Factor Analysis: descriptive tools to reduce dimensions.

After the preprocessing, we have different variations of the dataset after applying the different preprocessing techniques. Based on the algorithm, we compare the variations to see which best fits the algorithm.

2 EXPLORATORY DATA ANALYSIS

Before we go into any of the preprocessing or models, we go over what we are expecting to see in the original data.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Figure 2: Sample values

Figure 2 is a sample of the data with the different values for each variable. we can see that the explanatory variables in this dataset are fixed acidity, volatile acidity, citric acid content, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, ph, sulfates, and alcohol content, which are 11 variables in total. The quality is what we are trying to predict with our model. There are 1143 observations in out dataset.

	Type
fixed acidity	float64
volatile acidity	float64
citric acid	float64
residual sugar	float64
chlorides	float64
free sulfur dioxide	float64
total sulfur dioxide	float64
density	float64
pH	float64
sulphates	float64
alcohol	float64
quality	int64

Figure 3: Data types

All of the data is numerical with the quality being represented by integers and the other variables represented by floats, as Figure 3 shows.

Figure 4, 5 and 6 help us understand the data better. We can see the distribution of the data and these plots help us better notice any existence of extreme variables and how the data may be skewed. We notice that there are several x variables which are right skewed (i.e. have long right tail with high values). The x variables that have long right tails are: residual sugar, chlorides, total sulfur dioxide, and sulfates, which are 4 out of 11 variables. But this doesn't necessarily make the data points outliers, as outliers are decided by both x and y variables. Therefore, we decided to keep all of the observations in our analysis. Figure 5 also shows us what different values that each variable can take on the x-axis.

In Figure 7 we can see the correlation matrix. It shows the correlations between the variables and we can see that some of these

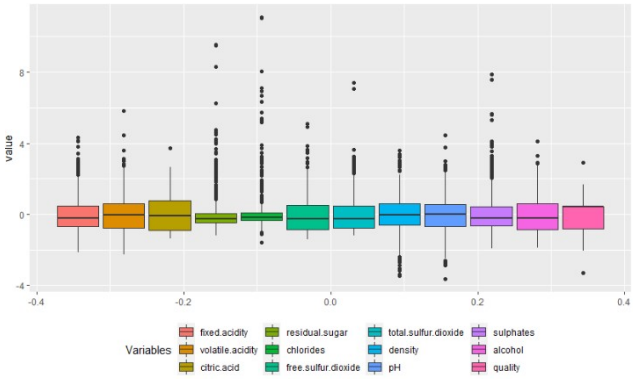


Figure 4: Inspection of x variable (1)

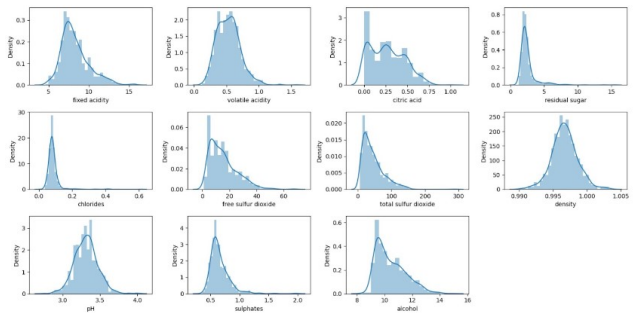


Figure 5: Inspection of x variable (2)

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015	0.657708	10.442111
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.170399	1.082196
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.550000	9.500000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.620000	10.200000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.000000	14.900000

Figure 6: Inspection of x variable (3)

variables do have some sort of correlation with each other. For example: (fixed acidity, pH) and (fixed acidity, density), etc.

The y variable, or quality, which is what we are trying to predict takes on 6 unique values ranging from 3 to 8, as Figure 8 shows. The percentage of wine records having a quality of 3, 4, 7, or 8 accounted for less than 5 percents of the dataset and the wine samples having a quality of 5 or 6 accounted for over 80 per cent of the dataset, so we can see that this is imbalanced.

After understanding our data better, we have seen that we do not have explicit outliers, NULL values nor factor attributes. We have seen that there are some correlated variables. However, instead of trying to remove one of them, we have applied dimensionality reduction techniques. This allows us to reduce the dimensions without losing very useful information.

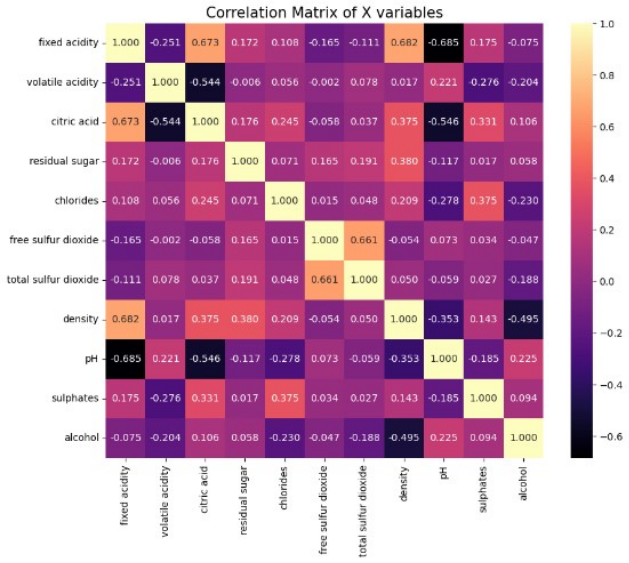
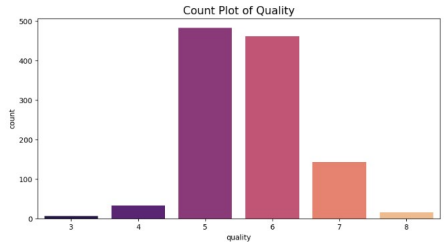


Figure 7: Correlations of x variable



	quality	%
5	483	0.422572
6	462	0.404199
7	143	0.125109
4	33	0.028871
8	16	0.013998
3	6	0.005249

Figure 8: Inspection of y variable

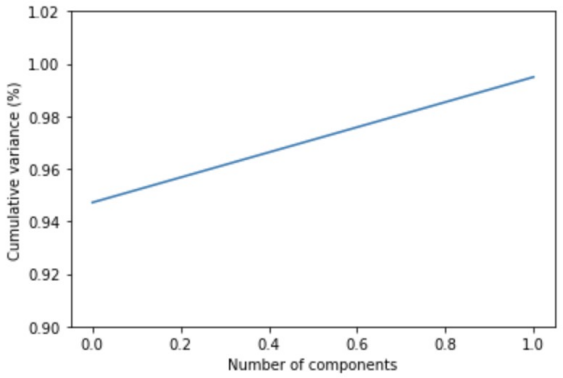


Figure 9: Scree Plot of variance based off of percentage of components

	PCA1	PCA2
0	-12.617220	-1.961333
1	22.607329	4.672012
2	7.772828	-2.322476
3	14.036956	-1.750228
4	-12.617220	-1.961333
...	...	...
1138	-2.917915	14.395525
1139	-5.082249	13.803305
1140	1.630621	16.464205
1141	9.960748	21.828683
1142	1.634233	16.468070

Figure 10

3 DIMENSIONALITY REDUCTION

3.1 Principal Component Analysis

PCA or principal component analysis is a common data preprocessing technique used to reduce the dimensionality of data by consolidating the data from the different variables. In order to determine whether PCA can be used on a specific dataset, we need to ensure that it satisfies the assumptions. The assumptions are the existence of linearity in the dataset and that the principal components with low variance can be eliminated from the dataset. In our dataset, the correlation matrix gives us an idea as to how correlated with one another the variables are and satisfies the assumption. The steps for applying PCA are normalizing the data, calculating the covariance matrix, and selecting a specific number of eigenvalues from a sorted list. This wouldn't modify the data at all, it would only reduce the dimensionality. With our data, we can apply PCA to reduce the dimensionality of our data from 10 columns with different variables to two columns with principal components.

3.2 Factor Analysis

The idea of factor analysis is to model observed, correlated variables as linear combinations of potentially lower number of unobserved variables called factors, plus "error" terms. Since from the correlation matrix of X variables, we observe that some pairs of them have high collinearity, we decided to try factor analysis to address potential multicollinearity issue, and reduce number of parameters.

First we check if the data is suitable to perform factor analysis. There are 2 types of tests we use, Bartlett's test and Kaiser-Meyer-Olkin (KMO) test.

Bartlett's test checks whether the correlation is present in the given data. It tests the null hypothesis (H_0) which assumes that the correlation matrix is an identical matrix, meaning that no correlation is present among the variables. We want to reject this null hypothesis because factor analysis aims at explaining the common variance i.e. the variation due to correlation among the variables. Here we use the significant level 0.05. The p-value we obtained is 0, which means the correlation matrix is not identity, which is the same conclusion as we had in Section 2.

KMO test measures the proportion of variance that might be a common variance among the variables. Larger proportions are expected as it represents more correlation is present among the variables therefore it's proper to use factor analysis. KMO score is

always between 0 to 1 and value more than 0.6 is often considered that the dataset is appropriate to do factor analysis. Our KMO test statistic is 0.440, which means the dataset is not suitable to do factor analysis. We futher observe patterns of amount of variance the factor explains (i.e. eigenvalues in factor analysis)

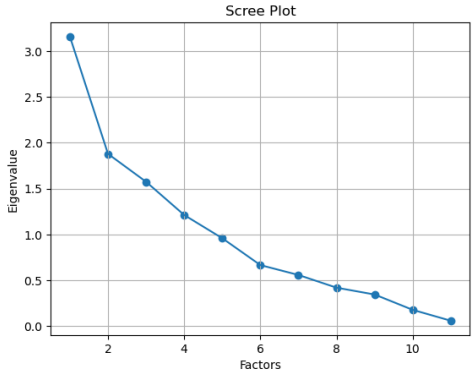


Figure 11: Eigenvalues in Factor Analysis

From Figure 11 we see that adding each factor results in some reduction in eigenvalue, but there is no flat patterns in the curve, which means adding each factor makes a difference in explaining the variances. With the support of both KMO test and this eigenvalue graph, we draw our conclusion that this dataset is not suitable to perform factor analysis.

4 ALGORITHMS

The algorithms that we applied are:

- K nearest neighbors, simple method with good performance for highly non-linear data.
- Decision trees + random forest, supervised learning tools to make decisions based on simple rules.
- Bayes classifiers (LDA, QDA, Naive Bayes), classification algorithms based on Bayes' Theorem.
- Logistic regression (one-vs-all), Ordinal logistic regression.

We applied each algorithm to each variation of the dataset and select the most effective model by comparing the accuracy and complexity. After finding the best dataset variation for each algorithm, we compared the algorithms against one another in terms of accuracy and complexity as well.

From data description session, we see that the number of observations in each category are not balanced, where most of the observations are either in quality category 5 or 6. Hence, we use stratified cross validation to retain the ratio of categories to make sure we include the minority categories in each of the K fold samples when we do cross validation. Also our PCA results show it's sufficient to keep 2 variables, so except for the raw dataset, we also used PCA processed dataset, which keeps only 2 variables. In summary, all the methods we used are performed on both raw dataset, and the dataset containing 2 variables from PCA, where all the methods used 5 fold stratified cross validation to estimate the parameters, the different methods are all measured by their Mean Squared Error (MSE) and Accuracy, keeping the final comparation fair between different methods.

4.1 K-Nearest Neighbor

The first machine learning algorithm that we have trained is the K-Nearest Neighbor. It is a discriminative classification mode which falls under the supervised learning subset of machine learning. It does not make any assumptions on underlying data because it is non-parametric. Moreover, it is an instance-based method, which means that it will classify the new instance into the class where the most similar points belong. It will also assume that all points are represented in p-dimensional space. The training process consists of calculating the distance from the new observation to all of the points in the dataset. Then, the k closest neighbors are selected. The algorithm analyzes the class each "neighbor" belongs to and the most common one is the one that is assigned to the new observation.

If we focus on the mathematical and statistical point of view, we have a training set composed of vectors X and y (x_1, y_1), $(x_2, y_2)...(x_n, y_n)$. X could be a matrix with p features to predict y, and therefore x_i would be the vector $x_i = [x_{i1}, x_{i2}, \ldots, x_{ip}]$. Once we have all of our data, we calculate the distances. We want to minimize $d(x_i, x_j)$ where d is a certain distance metric, "minkowski" by default. When we have the closest neighbors selected, we apply the majority vote technique and assign that class value to the y of the new observation.

The computational cost is high because we are calculating the distance between the data points for all of the training samples. Therefore, using dimensionality reduction techniques helps KNNS become more efficient and redues the time complexity of the model.

Inside the knn, we have different hyperparameters that we can tune. Without tuning the hyperparameters, we had the following results: in the original dataset, the accuracy was 54.3% , while in the PCA dataset we obtained an accuracy of 50.5%. The first hyperparameter is the number of neighbors that we choose to apply for the majority vote. On one hand, when this number is very low we tend to have overfitting. On the other hand, if we choose a high number of neighbors the sample will probably not be very representative. Moreover, if we choose an even number, we could have the same number of points belonging to different classes. To avoid these problems, we have picked the odd numbers between 3 and 49 inclusive.

Another hyperparameter that could be tuned is the metric. This refers to the method that we use to calculate the distance among the observations. We have tried the "minkowski", "manhattan", and "cosine".

In figures 12 and 13 we can see that the accuracy increases with the number of neighbors until 20 approximately. Then it starts varying a lot. In fact, the metric that changes the most within the number of neighbors is the cosine.

The last one is the weights. It can take two values: "uniform" and "distance". As its name says, when we set weights = "uniform", the algorithm treats all the neighbors the same. However, when we use the "distance" weights, the closer neighbors have more importance than the farther ones.

Figures 14 and 15 show that until 20 neighbors the accuracy increases with the number of neighbors (as well as in the metric

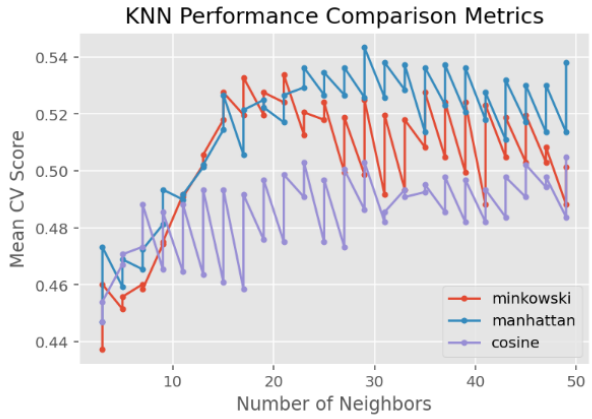


Figure 12: Knn performance comparison in the original dataset

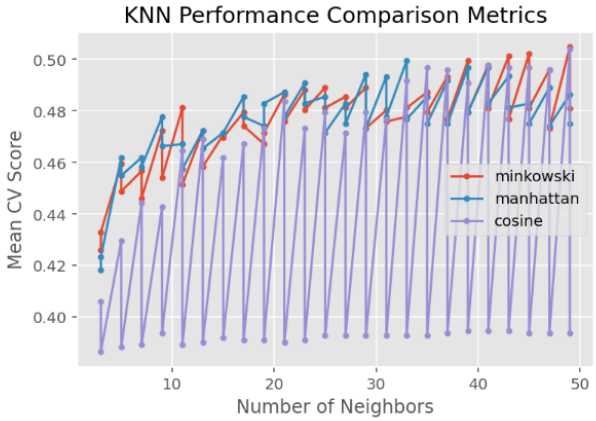


Figure 13: Knn performance comparison in the PCA dataset

plots). In the original dataset the distance weights perform better in general tahtn the uniform. In the PCA is the oposite but there is less difference.

Once we have tuned all the hyperparameters, we have obtained good results with different hyperparameters depending on whether the dataset has been preprocessed and the metric used to compare. We can compare them in the following table:

	Simple	Tunning hyperparameters
MSE (w/ PCA)	0.788 (0.884)	0.652 (0.761)
Zero-One loss (w/ PCA)	0.549 (0.541)	0.457 (0.495)
Accuracy (w/ PCA)	0.451 (0.459)	0.543 (0.504)

Table 1: Knn Summary

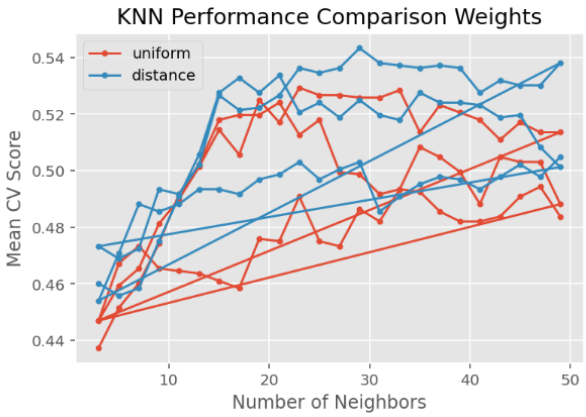


Figure 14: Knn performance comparison in the original dataset

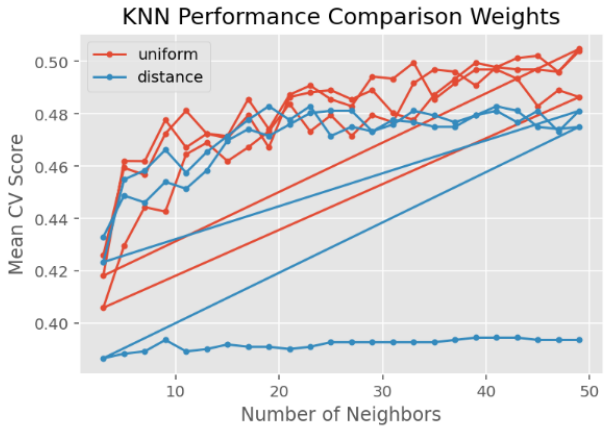


Figure 15: Knn performance comparison in the PCA dataset

4.2 Decision Trees and Random Forests

These models divide the data into more homogeneous subsets and then repeatedly into even smaller subsets. Until the subsets are sufficiently homogeneous. We computed these models firstly using the parameters by default and then we studied with which parameters we obtain better results. And we trained the models with these parameters that improve the accuracy. We optimized the parameters using a grid search. The hyperparameters that are optimized are the maximum depth and the function used to measure the quality of a split.

- Maximum depth of any node of the final tree is a limit to stop the further splitting of nodes when the specified tree depth has been reached during the building of the initial decision tree. We trained the model with max depths 2,5,10, and 20. A large depth of the tree can produce overfitting while a small depth can produce underfitting.
- Functions used to measure the quality of a split.We trained the model with gini and entropy. The gini impurity measures

the frequency at which any element of the dataset will be mislabelled when it is randomly labeled. Thus, the optimum split is chosen by the features with less Gini Index. The gini impurity is calculated using the formula $GiniIndex = 1 - \sum_j p_j^2$ where p_j is the probability of class j. And entropy is a measure of information that indicates the disorder of the features with the target. It is calculating using the formula $Entropy = -\sum_j p_j \cdot log_2 \cdot p_j$ where p_j is the probability of class j, as before. Similar to the Gini Index, the optimum split is chosen by the feature with less entropy. It gets its maximum value when the probability of the two classes is the same and a node is pure when the entropy has its minimum value, which is 0. One of the differences between these two formulas is that entropy is more complex since it makes use of logarithms. So, the calculation of the Gini index will be faster.

4.2.1 Decision Trees. Decision trees capture non-linearities and they provide ease of interpretation. They are basis of popular and powerful tools, for instance, random forests. we trained this model in the original dataset and in the dataset after applying PCA. The results are the following ones:

	Simple	Tunning hyperparameters
MSE (w/ PCA)	0.790 (1.142)	0.656 (0.701)
Zero-One loss (w/ PCA)	0.503 (0.470)	0.470 (0.520)
Accuracy (w/ PCA)	0.496 (0.434)	0.529 (0.479)

Table 2: Decision Trees Summary

We only focus on the accuracy to compare the performance of the models.From table 2, we can see that the accuracy is higher when we tune hyperparameters than when we use parameters by default. And that when we use the original dataset the results are better than when we use the dataset after applying PCA. This means that it is better to do the predictions with the original dataset because we have more information, the problem is that the computations are more complex.

The accuracy varies when different values of the parameters are combined as follows:

From Figure **??** we see that the highest value of the accuracy in the original dataset is obtained when max depth equal to 5 is combined with entropy.

From Figure 17 we see that the highest value of the accuracy in the dataset after applying PCA is obtained when max depth equal to 2 is combined with Gini.

4.2.2 Random Forests. Random Forest is the most-known bagging-learning tool, it uses random subspace methods to reduce correlations between the trees. It improves the predictive performance of DTs by averaging them. The advantages of this model are that is more accurate, reduce variance and overfitting. We follow the same procedure than when we trained decision trees. And we obtain the following results.

From table 3, we can see that, as with happened when we trained decision trees, the accuracy is higher when we tune hyperparameters. As before, we obtain better results when we use the original dataset. Because, as it is explained above, it provides more information than the dataset after applying PCA.

From Figures 18 and 19 we can see that the highest accuracy is obtained when entropy is combined with max depth equal to 5 in the original dataset. And with Gini combined with max depth equal to 2 in the dataset after applying PCA.

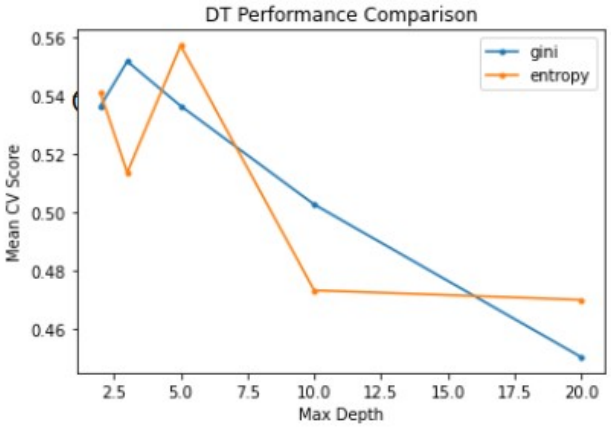


Figure 16: Decision trees performance comparison in the original dataset

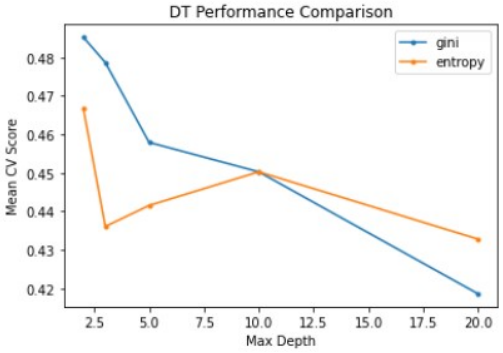


Figure 17: Decision trees performance comparison in the dataset after applying PCA

	Simple	Tunning hyperparameters
MSE (w/ PCA)	0.547 (0.879)	0.656 (0.684)
Zero-One loss (w/ PCA)	0.423 (0.530)	0.407 (0.498)
Accuracy (w/ PCA)	0.576 (0.469)	0.595 (0.501)

Table 3: Randon Forests Summary

4.3 Naive Bayes

Naive Bayes uses the Bayes Theorem for conditional independence in order to classify data. There are different variants of the Naive Bayes classification model that we implemented: Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes, and

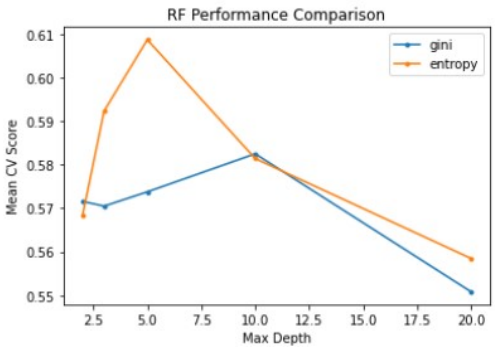


Figure 18: Random Forests performance comparison in the original dataset

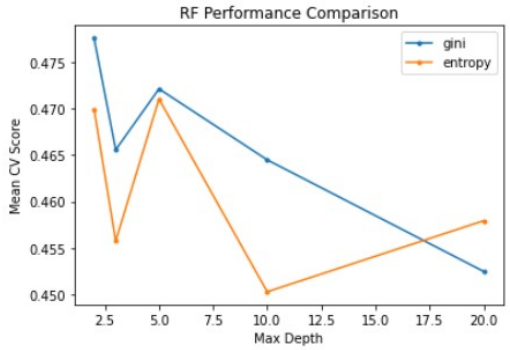


Figure 19: Random Forests performance comparison in the dataset after applying pca

Bernoulli Naive Bayes. Gaussian Naive Bayes follows a normal distribution, whereas the multinomial naive bayes is used for data that is multinomially distributed. The complement naive bayes algorithm is a variant of the multinomial naive bayes algorithm which are generally used for text classification. Bernoulli naive bayes follows a multivariate bernoulli distribution. We've tested both the normal dataset as well as the dataset that has been transformed by PCA on each of these models and found that the Gaussian naive bayes. In order to find the best model, we tuned the hyperparameters used Nested Cross Validation. By using GridSearch, we were able to train the model multiple times with different hyperparameter values from a prespecified parameter grid and increased the accuracy by approximately 2% per model. The PCA dataset being used on the datasets has marginally higher accuracy on both the Multinomial and Gaussian datasets, whereas on the others, it reduces the accuracy or keeps it the same.

Model	MSE	Zero-One Loss
Gaussian Naive Bayes	0.7070	0.4905
Multinomial Naive Bayes	1.0804	0.5756
Bernoulli Naive Bayes	1.0805	0.5275
Complement Naive Bayes	0.7288	0.5101

Table 4: MSE and Zero-One loss of the different models without PCA

Model	Accuracy	Accuracy with PCA
Gaussian Naive Bayes	0.5095	0.5004
Multinomial Naive Bayes	0.4400	0.4243
Bernoulli Naive Bayes	0.4243	0.4724
Complement Naive Bayes	0.4917	0.4917

Table 5: Accuracy of the different models with the different datasets

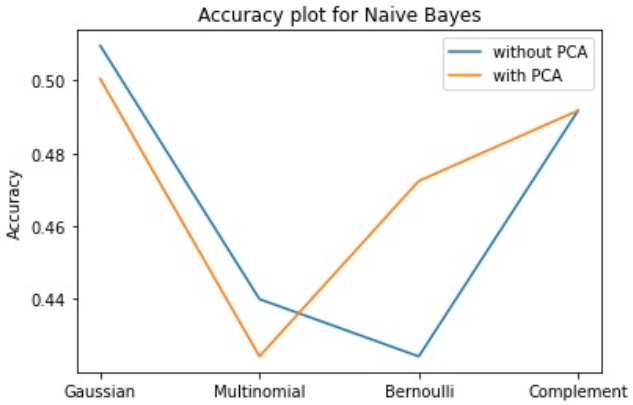


Figure 20: Compariosn of PCA and non PCA datasets on the Naive Bayes models

4.4 Logistic Regression Models

Logistic regression is a popular statistical model used for binary classification, utilizing sigmoid function. Logistic regression can be extended to multi class classification too. The logistic regression methods we applied here includes: One vs rest logistic regression, Multinomial logistic regression, and Ordinal logistic regression. We started with the simplest one-vs-all logistic regression, and generalizes logistic regression to multi class without order, since we are having more than 2 categories here. Lastly, since qualities of wine are categorized from 3 to 8, where higher number indicates that the wine quality is better, it naturally has an order in it. Therefore we considered ordinal logistic regression at the end. The MSE, zero-one loss and accuracy of different logistic models are as follows:

	One vs Rest	Multinomial	Ordinal
MSE (w/ PCA)	0.549 (0.663)	0.528 (0.652)	0.515 (0.658)
Zero-One loss (w/ PCA)	0.417 (0.502)	0.412 (0.491)	0.403 (0.490)
Accuracy (w/ PCA)	0.583 (0.498)	0.588 (0.509)	0.597 (0.510)

Table 6: Logistic Regression Summary

Since accuracy equals one minus zero-one loss, we only use accuracy to describe the performance of models, instead of both accuracy and zero-one loss. From Table 6, we see the performance

is as follows: One vs Rest < Multinomial < Ordinal logistic regression. This coincides with our intuition that the ordinal logistic model fits the nature of data best, hence has best performance. The performance of PCA processed data is worse than using raw data, as expected since we are reducing some dimension in features and losing information comapred to raw data. (Figure 21) But it is achieving a relatively good result with just 2 variables compared to original 11 variables. However, the accuracy overall is just around 60% when we use raw data, and is around 50% when we use PCA processed data (keeps only 2 features among 11 features). Which indicates both are not well performed.

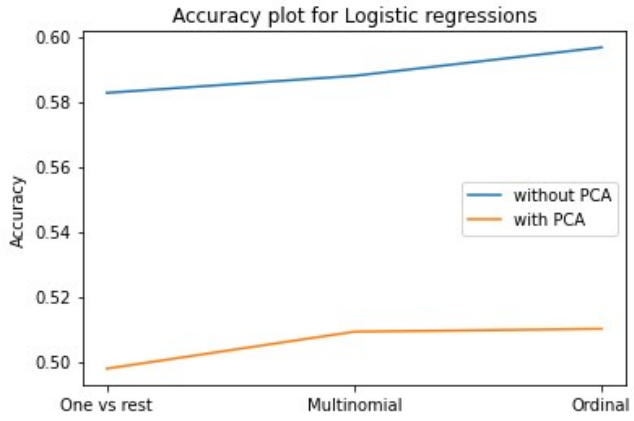


Figure 21: Logistic regression accuracy

5 MODEL COMPARISON AND DISCUSSION

Once we have applied all the algorithms, we have compared their results.

Regarding the different datasets, there is a difference that can reach a 10% difference in terms of accuracy. However, in most of the models difference is around 5%. If we had more complex and long computation it could be worth it to use the PCA dataset. But in this case we would stay with the original data.

We started by comparing the difference among the original data and the one after applying PCA. In the decision tree, the increment of mean squared error is the biggest one without tunning the hyperparameters. This is also reflected in the random forest, where the accuracy decreased almost a 10%. Another fact when comparing these two algorithms is that even though the results when applying the majority vote are better, it is not a high increment.

Attending the prediction results, we have not taking into account the simple models (without hyperparameter tuning). The best algorithm regarding MSE and the accuracy is the ordinal logistic regression, in both the original and the PCA dataset. Overall, the performance of the models we applied are: Ordinal Logistic Regression > Random Forest > KNN > Decision Trees > Gaussian Naive Bayes. Where we pick the best performed model in each catetory of models we use.

6 OUTCOME EVALUATION

Comparing our main ideas with the outcome of the project, we have been able to achieve all the objectives that we had in the proposal. Nevertheless, we estimated that the data pre-processing was going to be more time-consuming, but it was very clean. We expected to have factor attributes to transform into dummy variables, outliers, and null values. In addition, we thought that the factor analysis could have been a dimensionality technique we could make use of.

Regarding the results of our models, we expected more accuracy in general. We achieved better predictions when we tuned the hyperparameters in every model, but still we had a high mean squared error and relatively low accuracy overall.

7 NEW INSIGHTS

After doing this project we realized the importance of trying different models. Even though the performance of each one could be similar, each algorithm uses different statistical and mathematical approaches. We obtained different results when we trained them. In addition, if we only try one model when predicting, probably the prediction will not be the best one.

Another important part of the process is to study the hyperparameters, and how they affect to accuracy optimization. In most of the courses that we have taken, we do not give much importance to them, but they are very helpful. Understanding them properly allows you to increase your results and avoid overfitting or underfitting. For instance, if inside Knn we use only one neighbor, we will probably get overfitting.

Another helpful tool to avoid overfitting is cross-validation. We are used to using naive K-fold cross-validation. Nevertheless, in this specific dataset, we had to use the stratified K-fold cross-validation because we had unbalanced data.

In addition, the performance of each model reminds us the importance of choosing the appropriate model which describes the data best, For example, in Logistic regression, the ordinal model performs best as it captures the natural underlying order in the response variable.

8 GROUP CONTRIBUTION

- Danyang Wang: Data description, Factor Analysis, Logistic regression.
- Vinitha Marupeddi: Data description, PCA, Naive Bayes
- Elena Gómez: Data description, Data cleaning, Decision trees, Random forest
- Lucía Ostolaza: Data description, Data cleaning, KNN, Conclusions

REFERENCES

[1] IE Frank and Bruce R Kowalski. 1984. Prediction of wine quality and geographic origin from chemical measurements by parital least-squares regression modeling. *Analytica Chimica Acta* 162 (1984), 241–251.

[2] Joanna M Gambetta, Daniel Cozzolino, Susan EP Bastian, and David W Jeffery. 2016. Towards the creation of a wine quality prediction index: Correlation of Chardonnay juice and wine compositions from different regions and quality levels. *Food analytical methods* 9, 10 (2016), 2842–2855.

[3] Yogesh Gupta. 2018. Selection of important features and predicting wine quality using machine learning techniques. *Procedia Computer Science* 125 (2018), 305–312.