Analysis of Wine Data Using KMeans Clustering

William Duquette

June 22, 2023

Introduction

This paper analyzes and clusters wines based on various features commonly used to measure the quality of wines. To do this clustering, we use KMeans, an unsupervised algorithm. Although we typically use this algorithm when we do not necessarily have "targets," in this paper, we will see how pure our clusters are. What is meant by purity, in this case, is whether the clusters our algorithm form have all one kind of wine. We seek to understand how well the clusters formed using the explanatory variables correlate with the response variables. Using Gini Impurity as our measure of the algorithm's performance, we find that the KMeans algorithm works exceptionally well in clustering wines in a way that maintains purity.

Wine Data

This data set contains data about 6947 Portuguese wines. It is important to note that the data is not balanced as there are 1599 red and 4898 white wines. There are 13 features in the data set, including wine labels, fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulfates, alcohol, and quality.

Data Preparation

We first check if any cleaning or transformation is needed. Of course, these things should not be done without thought as it increases the risk of adding human bias to the problem. However, feature scaling is necessary because many features have very different ranges. This can be seen in the boxplot below:

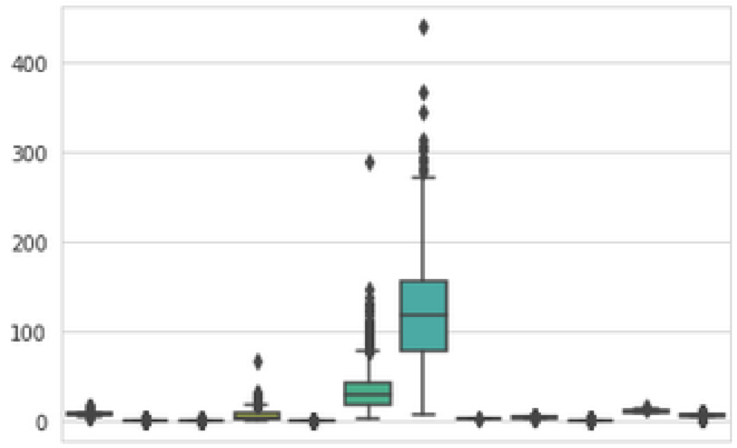

Figure 1: Boxplot showing ranges of different features

Either standardization or normalization is appropriate in this context. However, we use standardization because this method is less susceptible to outliers. To standardize a value, we subtract by the mean and divide by the standard deviation. With the newly scaled data, we can continue with our experiment.

Preliminary Data Analysis

Analyzing data when working in an unsupervised environment can sometimes be challenging. However, creating basic data visualizations is still helpful to see how variables relate. Visualizations are always helpful in understanding the data. In this case, we create a correlation matrix. A correlation matrix is helpful because it allows us to see the relationship between values more concretely. Below, we can see the correlation matrix for the wine data set:

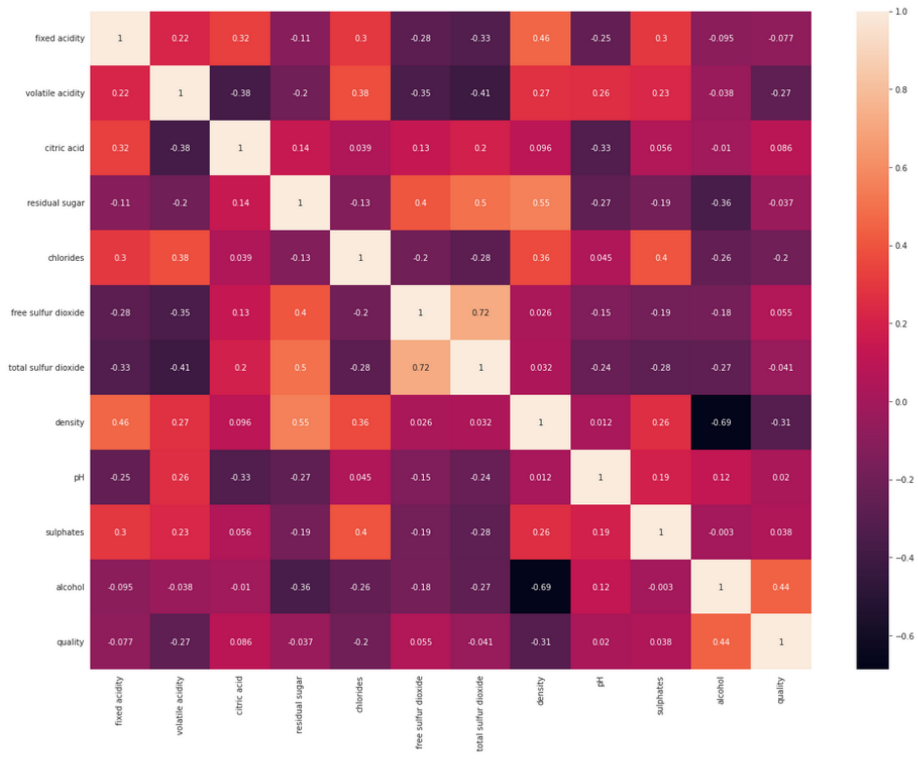

Figure 2: Correlation matrix for wine data set

Clustering

Plotting to visually determine how many clusters are needed is almost impossible if the data is in high dimensions. This is when making a plot of the MSE v. K can be extremely useful. With this plot we find the elbow which signifies a natural clustering of our data. Below is the elbow plot for the wine data.

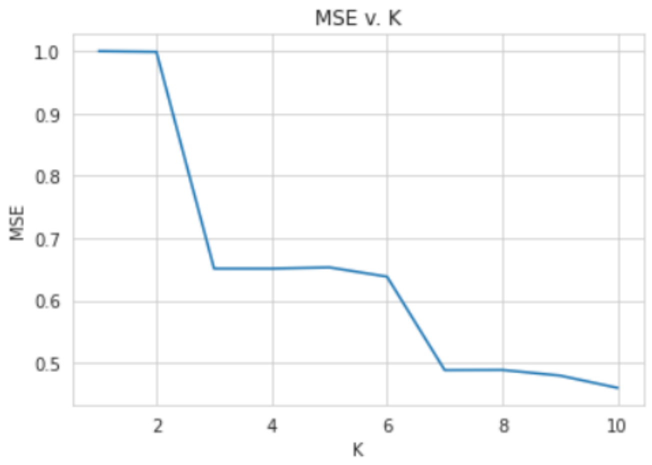

Figure 3: Elbow plot for wine data set

We see that three and seven are the elbows of the graph; we select three as the number of clusters used. It may seem odd that the optimal value of k is not two, given we have red and white wines in our data set. Although we only have two wine types (red and white), we could imagine a situation with multiple subgroups within each wine type. Although all the wines in the data set are within an overarching wine-type group, it could be challenging to group all of the wines in two distinct groups, given how different subgroups may be. This is not necessarily the case, but it is a crucial thought experiment to help illustrate why more than two clusters may be necessary.

Looking at a clustering where $K = 3$, we find a peculiar result. One of our clusters contains only one data point, suggesting that this point might be an outlier. One particular white wine is a massive outlier and cannot feasibly be clustered well with the other white wines. It can be helpful to remove extreme outliers to see the algorithm's performance. However, outliers should only be permanently removed from the experiment if the outlier does not represent the data, which does not appear true in this situation. Removing outliers should be done sparingly, as it can introduce bias. But, it can be helpful to remove the outlier temporarily to see the algorithm's performance while still noting the results we have seen with the outlier present, as we have.

In this case, we see performance differences after removing the extreme white wine in our data set. We first see that only two clusters are recommended by our elbow plot.

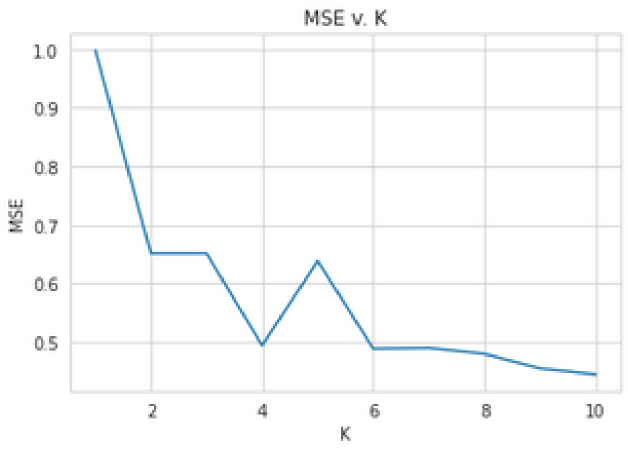

Figure 4: Elbow plot for wine data set (outlier removed)

Additionally, we see that the clusters appear to group our data naturally. Most white wines group together, while most red wines group together. These clusters are not perfect, as seen by the table below, but over 98.1% of white wines are within the same cluster, and 98.4% of red wines are within the same cluster.

Cluster	Reds	Whites	Total
1	25	4804	4829
2	1574	93	1667

Cluster Impurity

We need a more formal measure to see how well KMeans clustered our data. In this case, we will calculate the purity of our clusters. The Gini Impurity will tell us how different our clusters are from the pure clustering (where all points in each cluster are the same wine type).

The Gini Impurity can be defined as

$$G_k = 1 - \Sigma_{i=1}^{c} N(i)^2 \tag{1}$$

where c is the number of classes and $N(i)$ is the fraction of data points at the cluster that belong to class i.

Let's walk through two examples to understand what a good Gini value is. In both cases, we will assume that a cluster has 100 points and two possible classes (red and white).

1. If the cluster contains fifty red and fifty white points (so 50/50), we would have a Gini Impurity of 0.5.

2. If the cluster contains 90 red and ten white, we would have a Gini Impurity of 0.18.

To conclude our example, the closer to zero, the more pure the cluster.

In our case, we see low impurity values, which is encouraging. Below is the same table that held our results previously, but now with information on the Gini Impurity. The table below suggests that our clustering algorithm successfully groups our data points in clusters with other truly similar points, in this case, of the same type (red or white).

Cluster	Reds	Whites	Total	Gini Impurity
1	25	4804	4829	0.0103
2	1574	93	1667	0.105

Conclusion

In this paper, we analyzed a wine data set using KMeans clustering and sought to determine whether the clusters created were pure. As we have seen, it is clear that the KMeans algorithm works well for clustering wine data points in such a way that keeps the clusters relatively pure, especially once the outlier is removed. This is an important finding as it allows us to use the KMeans algorithm more confidently because we can have faith that this algorithm will cluster well.

Appendix

KMeans v. KMeans++

KMeans is a classic algorithm in unsupervised machine learning. Like most areas of computer science, there have been improvements to the algorithm. One such improvement is the KMeans++ algorithm (we will go into more detail below). Using an example set, we will show how KMeans++ improves upon the original KMeans.

The data consists of (x,y) pairs to help us see how the basic algorithm works. Looking at this data, there are clearly four clusters.

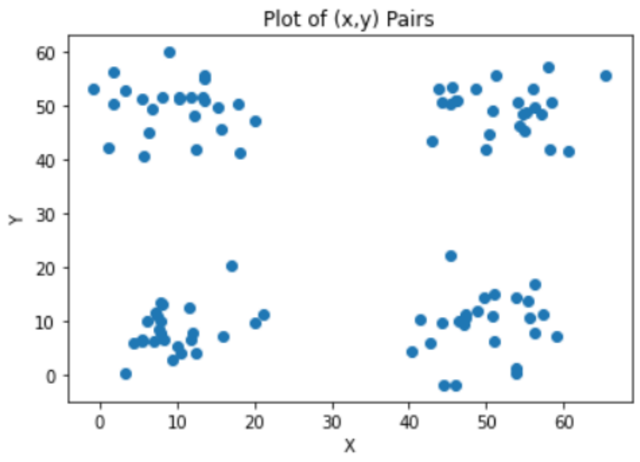

Figure 5: Scatter plot for example data set

Before we use KMeans on this example set, we have to better understand what this algorithm looks like.

KMeans

This algorithm will group points with other points that are similar to them. This is done in a relatively simple algorithm. The general approach of the algorithm is as follows:

1. Initialization: randomly place k centers
2. Repeat:
   (a) Assignment phase: map each sample (point) to its closest center
   (b) Update phase: move each center to the mean of its assigned samples

This algorithm will iterate until no changes are made to the center locations.

KMeans++

KMeans and KMeans++ algorithms are similar. The only difference between the two is in the initialization step. In KMeans++, we select our first center randomly (random point), but instead of picking the following centers randomly, we try to pick points for our centers far away from our previous centers. Although this takes more time in the initialization step, it reduces the number of iterations the algorithm has to run to find the center's location.

Comparison of KMeans and KMeans++

Now that we have established both approaches to KMeans that we will be testing, we can begin to run our algorithms on the data. Below we can see an example of how the two algorithms compare. The centers that KMeans and KMeans++ produce will likely be similar, if not visually identical. However, over multiple iterations, it becomes clear that the KMeans++ algorithm is far superior to the original KMeans++ algorithm.

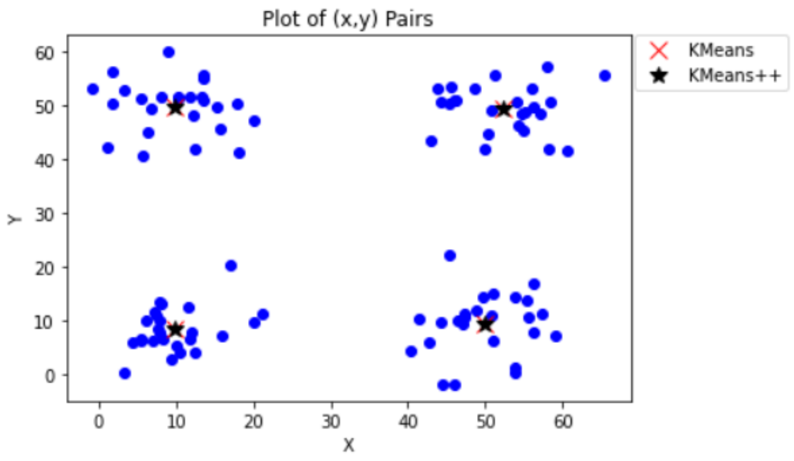

Figure 6: Scatter plot with the placement of centers for KMeans and KMeans++

As previously mentioned, it is likely that across one iteration KMeans and KMeans++ appear the same. However, as shown below, the KMeans++ distinguishes itself over multiple iterations as the superior algorithm. The average Mean Squared Error for the KMeans algorithm is 17939, while the average Mean Squared Error for KMeans++ is 8955. This may be a surprising result, but it will become clear when looking at the data graphically. Over ten possible iterations of KMeans and KMeans++ being run on our example data, two plots had sub-optimal center placements for the KMeans algorithm, while KMeans++ always had optimal center placement. For example, one such plot looked like this:

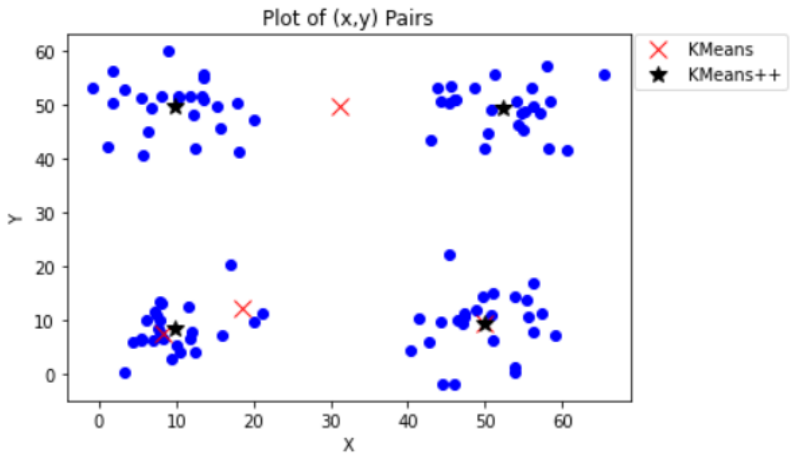

Figure 7: Scatter plot for example data set with the sub-optimal placement of centers for KMeans

From our analysis, we can make some conclusions about KMeans clustering and how well it works as a clustering algorithm on our data. We can say that the KMeans++ algorithm is superior to the KMeans algorithm. This is clear from our experiment, where we compared the average MSE and the center locations. For a good reason, the KMeans algorithm is one of the most commonly used unsupervised learning algorithms. The elegancy and efficiency of this algorithm is impressive. Additionally, we see an even more outstanding performance when the algorithm is improved (KMeans++). The results affirm the algorithm's abilities, and we recommend that this algorithm continue to be used as frequently as possible.