Wine Classification Based on Color Using Unsupervised ML Models

Kanishk Kumar

July 07, 2022

Table of Contents

Main Objective

This project will be focused on **classification.** The main objectives of this project are as follows:

- To apply data preprocessing and preparation techniques in order to obtain clean data.
- To build at least three unsupervised machine learning classification models that are able to classify wines into two types: red and white.
- To analyze and compare performance of each model in order to choose the best model.

Description of the Data

These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. The initial data set had around 30 variables, but here I only have the 13-dimensional version. The attributes are:

1) Fixed Acidity
2) Volatile Acidity
3) Citric Acid
4) Residual Suga
5) Chlorides
6) Free Sulfur Dioxide
7) Total Sulfur Dioxide
8) Density
9) pH
10) Sulphates
11) Alcohol
12) Quality
13) Color

	0	1	2	3
fixed_acidity	7.4	7.8	7.8	11.2
volatile_acidity	0.7	0.88	0.76	0.28
citric_acid	0.0	0.0	0.04	0.56
residual_sugar	1.9	2.6	2.3	1.9
chlorides	0.076	0.098	0.092	0.075
free_sulfur_dioxide	11.0	25.0	15.0	17.0
total_sulfur_dioxide	34.0	67.0	54.0	60.0
density	0.9978	0.9968	0.997	0.998
pH	3.51	3.2	3.26	3.16
sulphates	0.56	0.68	0.65	0.58
alcohol	9.4	9.8	9.8	9.8
quality	5	5	5	6
color	red	red	red	red

Our dataset has 6497 rows and 13 columns.

```
data.shape
```

```
(6497, 13)
```

EDA and Feature Engineering

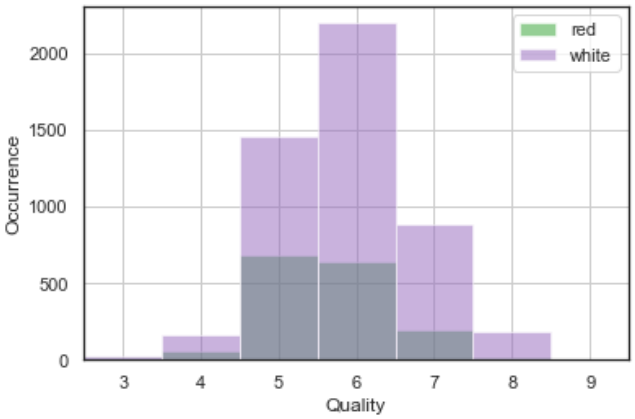
In this section we will explore the dataset in depth through several EDA techniques such as checking for data skewness, data visualization, furthermore, showing the correlation between the features for the sake of feature engineering implementation and data cleaning.

Let's make sure all the columns have continuous values as you can see on the right.

Now let's check the Quality column using the code below:

```
# seaborn styles
sns.set_context('notebook')
sns.set_style('white')

# custom colors
red = sns.color_palette()[2]
white = sns.color_palette()[4]

# set bins for histogram
bin_range = np.array([3, 4, 5, 6, 7, 8, 9])

# plot histogram of quality counts for red and white wines
ax = plt.axes()
for color, plot_color in zip(['red', 'white'], [red, white]):
    q_data = data.loc[data.color==color, 'quality']
    q_data.hist(bins=bin_range,
                alpha=0.5, ax=ax,
                color=plot_color, label=color)


ax.legend()
ax.set(xlabel='Quality', ylabel='Occurrence')

# force tick labels to be in middle of region
ax.set_xlim(3,10)
ax.set_xticks(bin_range+0.5)
ax.set_xticklabels(bin_range);
ax.grid('off')
```

```
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
color                   object
```



We clearly won't be needing this column and drop it from our training data along with Color column.

Now let's check the correlation between the columns using this code:

```
float_columns = [x for x in data.columns if x not in ['color', 'quality']]

# The correlation matrix
corr_mat = data[float_columns].corr()

# Strip out the diagonal values for the next step
for x in range(len(float_columns)):
    corr_mat.iloc[x,x] = 0.0

corr_mat
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol
fixed_acidity	0.000000	0.219008	0.324436	-0.111981	0.298195	-0.282735	-0.329054	0.458910	-0.252700	0.299568	-0.095452
volatile_acidity	0.219008	0.000000	-0.377981	-0.196011	0.377124	-0.352557	-0.414476	0.271296	0.261454	0.225984	-0.037640
citric_acid	0.324436	-0.377981	0.000000	0.142451	0.038998	0.133126	0.195242	0.096154	-0.329808	0.056197	-0.010493
residual_sugar	-0.111981	-0.196011	0.142451	0.000000	-0.128940	0.402871	0.495482	0.552517	-0.267320	-0.185927	-0.359415
chlorides	0.298195	0.377124	0.038998	-0.128940	0.000000	-0.195045	-0.279630	0.362615	0.044708	0.395593	-0.256916
free_sulfur_dioxide	-0.282735	-0.352557	0.133126	0.402871	-0.195045	0.000000	0.720934	0.025717	-0.145854	-0.188457	-0.179838
total_sulfur_dioxide	-0.329054	-0.414476	0.195242	0.495482	-0.279630	0.720934	0.000000	0.032395	-0.238413	-0.275727	-0.265740
density	0.458910	0.271296	0.096154	0.552517	0.362615	0.025717	0.032395	0.000000	0.011686	0.259478	-0.686745
pH	-0.252700	0.261454	-0.329808	-0.267320	0.044708	-0.145854	-0.238413	0.011686	0.000000	0.192123	0.121248
sulphates	0.299568	0.225984	0.056197	-0.185927	0.395593	-0.188457	-0.275727	0.259478	0.192123	0.000000	-0.003029
alcohol	-0.095452	-0.037640	-0.010493	-0.359415	-0.256916	-0.179838	-0.265740	-0.686745	0.121248	-0.003029	0.000000

Let's see the pairwise maximal correlations:

```
corr_mat.abs().idxmax()
```

```
fixed_acidity                density
volatile_acidity      total_sulfur_dioxide
citric_acid               volatile_acidity
residual_sugar                    density
chlorides                       sulphates
free_sulfur_dioxide   total_sulfur_dioxide
total_sulfur_dioxide   free_sulfur_dioxide
density                           alcohol
pH                            citric_acid
sulphates                       chlorides
alcohol                           density
dtype: object
```

Now let's check which feature has a skew value of more than 0.75 and perform log transformation on them:

```
skew_columns = (data[float_columns]
                .skew()
                .sort_values(ascending=False))

skew_columns = skew_columns.loc[skew_columns > 0.75]
skew_columns
```

```
chlorides             5.399828
sulphates             1.797270
fixed_acidity         1.723290
volatile_acidity      1.495097
residual_sugar        1.435404
free_sulfur_dioxide   1.220066
dtype: float64
```

```
# Performing log transform on skewed columns.
for col in skew_columns.index.tolist():
    data[col] = np.log1p(data[col])
```

Now we scale all our training features:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
data[float_columns] = sc.fit_transform(data[float_columns])

data.head(4)
```

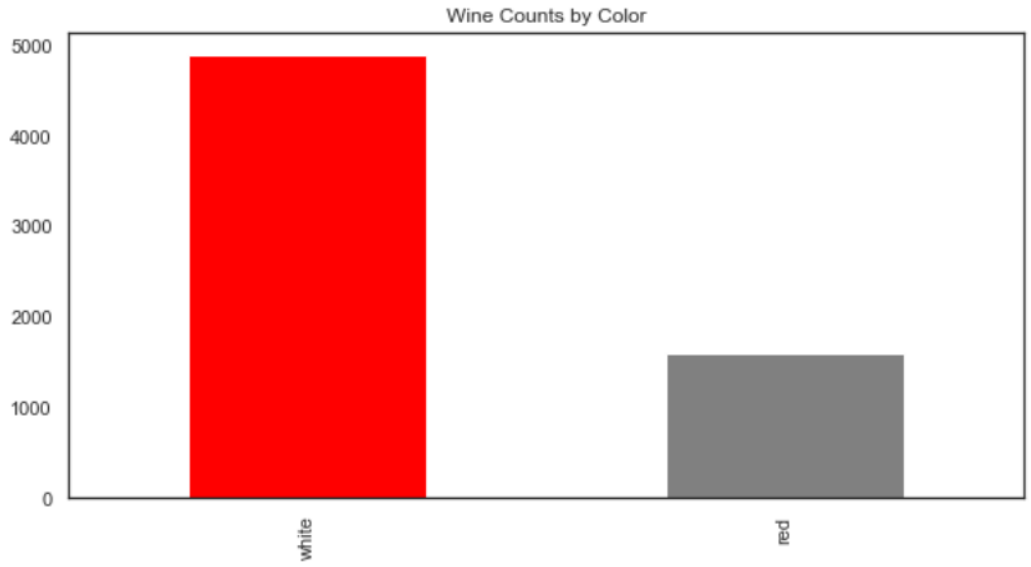
	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	0.229509	2.135767	-2.192833	-0.815173	0.624554	-1.193601	-1.446359	1.034993	1.813090	0.250355	-0.915464	5	red
1	0.550261	3.012817	-2.192833	-0.498175	1.281999	-0.013944	-0.862469	0.701486	-0.115073	1.059213	-0.580068	5	red
2	0.550261	2.438032	-1.917553	-0.625740	1.104012	-0.754684	-1.092486	0.768188	0.258120	0.862549	-0.580068	5	red
3	2.802728	-0.337109	1.661085	-0.815173	0.594352	-0.574982	-0.986324	1.101694	-0.363868	0.389396	-0.580068	6	red

Then we plot a pairplot to see pair-wise relationships in our dataset using this code on the right here:

```
sns.set_context('notebook')
sns.pairplot(data[float_columns + ['color']],
             hue='color',
             hue_order=['white', 'red'],
             palette={'red':'red', 'white':'gray'});
```

Let's see what the actual number and percentage of red and white wine is in our dataset:

```
data_color_counts = data.color.value_counts().to_frame()
data.color.value_counts().plot(kind="bar", figsize=(10,5), title = 'Wine Counts by Color', color={"grey", "red"})
data_color_perc = round(data.color.value_counts(normalize=True).to_frame(), 4)
data_color_perc["color"] = data_color_perc["color"] * 100
data_color_perc.rename(columns = {"color": "Color of Wine (%)"},inplace = True)
```




```
wine_colors = pd.merge(data_color_counts, data_color_perc, on = data_color_counts.index)
wine_colors.rename(columns={'key_0':'color_name'}, inplace=True)
wine_colors.rename(columns={'color':'Wine Counts by Color'}, inplace=True)
wine_colors
```

	color_name	Wine Counts by Color	Color of Wine (%)
0	white	4898	75.39
1	red	1599	24.61

A perfect classification would look something like the dataframe above.

Machine Learning Analysis

1. k-means

Here we have Implemented K means algorithm, with a range of different K values 0-20 to see if it can find the appropriate number of clusters (that is two), and to measure the entropy in the model. We've selected the inertia metric and elbow method to find the appropriate value of K.

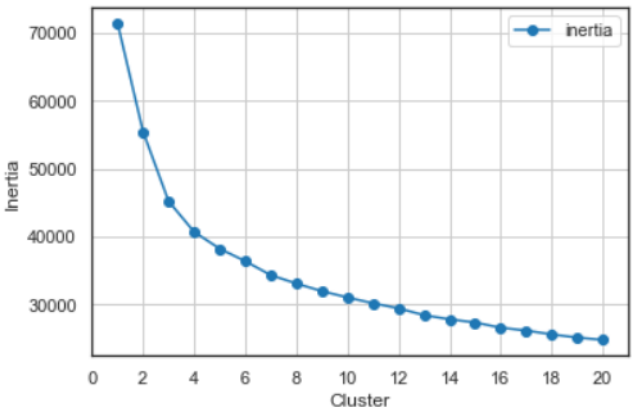
Inertia: is defined as the sum of squared distance from each point (X_i) to its cluster C_k.

$$\sum_{i=1}^{n} (X_i - C_k)^2$$

Using the code here, we'll try to see if the algorithm can see the correct number of clusters.

```
from sklearn.cluster import KMeans

# Create and fit a range of models
km_list = list()

for clust in range(1,21):
    km = KMeans(n_clusters=clust, random_state=42)
    km = km.fit(data[float_columns])

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))
plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker = 'o',ls = '-', grid = True)
ax.set_xticks(range(0,21,2))
ax.set_xlim(0,21)
ax.set(xlabel='Cluster', ylabel='Inertia');
```

As shown in the graph, if we follow elbow method approach, we can't clearly make sense of the correct number of k but the biggest drop is at still at 2, which aligns with our dataset which contains two different clusters (colors).



After applying K means algorithm with number of clusters = 2 to this dataset, it will classify each observation to the two clusters which we assigned in the model.

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=2, random_state=42)
km = km.fit(data[float_columns])

data['kmeans'] = km.predict(data[float_columns])
k_means_df = (data[['color','kmeans']]
              .groupby(['kmeans','color'])
              .size()
              .to_frame()
              .rename(columns={0:'number'}))
display(k_means_df)
print('Red Wine classified accurately: '
      + str(round((k_means_df.number[0]['red']
      / (k_means_df.number[0]['red']
      + k_means_df.number[1]['red']))
      * 100, 2)) + '%')
print('White Wine classified accurately: '
      + str(round((k_means_df.number[1]['white']
      / (k_means_df.number[1]['white']
      + k_means_df.number[0]['white']))
      * 100, 2)) + '%')
```

We can see how many of these observations are classified correctly here:

		number
kmeans	**color**	
0	**red**	1576
	white	87
1	**red**	23
	white	4811

```
Red Wine classified accurately: 98.56%
White Wine classified accurately: 98.22%
```

Red Wine classified accurately: 98.56%
White Wine classified accurately: 98.22%

The outcomes were very satisfactory for the two clusters.

2. Agglomerative Hierarchical Clustering

Applying agglomerative hierarchical clustering with no. clusters = 2. This algorithm works best when the number of clusters in already known, which is true in our case.

```
from sklearn.cluster import AgglomerativeClustering
ag = AgglomerativeClustering(n_clusters=2,
                             linkage='ward',
                             compute_full_tree=True)
ag = ag.fit(data[float_columns])
data['agglom'] = ag.fit_predict(data[float_columns])
agglom_df = (data[['color','agglom']]
             .groupby(['agglom','color'])
             .size()
             .to_frame()
             .rename(columns={0:'number'}))
display(agglom_df)
print('Red Wine classified accurately: '
      + str(round((agglom_df.number[1]['red']
      / (agglom_df.number[1]['red']
      + agglom_df.number[0]['red']))
      * 100, 2)) + '%')
print('White Wine classified accurately: '
      + str(round((agglom_df.number[0]['white']
      / (agglom_df.number[0]['white']
      + agglom_df.number[1]['white']))
      * 100, 2)) + '%')
```

After applying Agglomerative Hierarchical Clustering algorithm with number of clusters = 2 to this dataset, it will classify each observation to the 2 clusters which we assigned in the model.

We can see how many of these observations are classified correctly here:

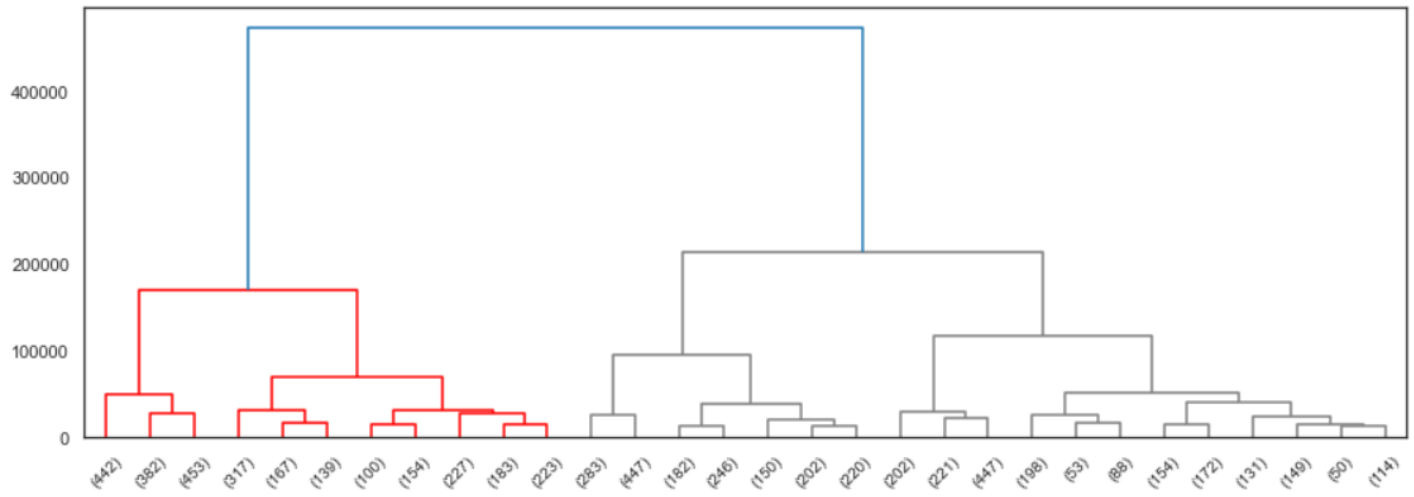
Red Wine classified accurately: 98.06%
White Wine classified accurately: 97.08%

The outcomes were very satisfactory for the two clusters but k-means still outperformed this model.

		number
agglom	**color**	
0	**red**	31
	white	4755
1	**red**	1568
	white	143

```
Red Wine classified accurately: 98.06%
White Wine classified accurately: 97.08%
```

```
from scipy.cluster import hierarchy

Z = hierarchy.linkage(ag.children_, method='ward')

fig, ax = plt.subplots(figsize=(15,5))

hierarchy.set_link_color_palette(['red', 'gray'])

den = hierarchy.dendrogram(Z, orientation='top',
                           p=30, truncate_mode='lastp',
                           show_leaf_counts=True, ax=ax)
```



3. DBSCAN

After applying DBSCAN algorithm many times with different parameters, I still got poor outcomes. When I validated the clustered observations with the actual classes, I got high error and the algorithm was unable to even predict the appropriate number of clusters. So, I made a function that checks different parameters and saves a list of those parameters which deduce two clusters. You can see the code on the right:

After running several ranges of parameters, I ended up with this very high range of epsilon and minimum samples below:

```
dbscan_clusters = []
cluster_count   = []

dbscan_grid_search(X_data = data[float_columns],
                   lst = dbscan_clusters,
                   clst_count = cluster_count,
                   eps_space = pd.np.arange(1, 11, 1),
                   min_samples_space = pd.np.arange(100, 1001, 100),
                   min_clust = 2,
                   max_clust = 2)
```

```
Search Complete.
Your list is now of length 86.
Hyperparameter combinations checked: 100.
```

So, I ended up running the algorithm with epsilon = 2 and minimum samples value = 200 and got somewhat sensible results.

We can see how many of these observations are classified correctly here:

Red Wine classified accurately: 44.97%
White Wine classified accurately: 91.12%

The outcomes were very poor with only white wine being clustered correctly and even that less accurately than the previous two algorithms.

```
def dbscan_grid_search(X_data, lst, clst_count, eps_space = 0.5,
                       min_samples_space = 5, min_clust = 0, max_clust = 10):

    # Starting a tally of total iterations
    n_iterations = 0


    # Looping over each combination of hyperparameters
    for eps_val in eps_space:
        for samples_val in min_samples_space:

            dbscan_grid = DBSCAN(eps = eps_val,
                                 min_samples = samples_val, n_jobs=-1)

            # fit_transform
            clusters = dbscan_grid.fit(data[float_columns])

            # Saving the number of clusters
            n_clusters = len(np.unique(clusters.labels_))


            # Increasing the iteration tally with each run of the loop
            n_iterations += 1


            # Appending the lst each time n_clusters criteria is reached
            if n_clusters >= min_clust and n_clusters <= max_clust:

                dbscan_clusters.append([eps_val,
                                        samples_val])


    # Printing grid search summary information
    print(f"""Search Complete. \nYour list is now of length {len(lst)}. """)
    print(f"""Hyperparameter combinations checked: {n_iterations}. \n""")
```

As you can see on the left, I ended up with a list of 86 combinations which were giving somewhat sensible results.

```
dbs = DBSCAN(eps=2, min_samples=200, metric='euclidean')
dbs = dbs.fit(data[float_columns])
len(np.unique(dbs.labels_))
data['dbscan'] = dbs.fit_predict(data[float_columns])
dbscan_df =  (data[['color','dbscan']]
              .groupby(['dbscan','color'])
              .size()
              .to_frame()
              .rename(columns={0:'number'}))
display(dbscan_df)
print('Red Wine classified accurately: '
      + str(round((dbscan_df.number[-1]['red']
      / (dbscan_df.number[-1]['red']
      + dbscan_df.number[0]['red']))
      * 100, 2)) + '%')
print('White Wine classified accurately: '
      + str(round((dbscan_df.number[0]['white']
      / (dbscan_df.number[-1]['white']
      + dbscan_df.number[0]['white']))
      * 100, 2)) + '%')
```

		number
dbscan	**color**	
-1	**red**	719
	white	435
0	**red**	880
	white	4463

```
Red Wine classified accurately: 44.97%
White Wine classified accurately: 91.12%
```

Key Findings

k-means and Agglomerative Hierarchical Clustering methods both were fast and accurate in terms of finding the appropriate number of clusters. In addition of that, they clustered majority of observations correctly and in overall both algorithms achieved above 97 % in terms of clustering accuracy.

In contrast, DBSCAN required a lot of time to find the appropriate number of clusters since it identified the number of clusters depending on the parameters that we need to change again and again to find the expected number of clusters. Furthermore, the outcomes of the observation's clustering were very poor.

Advanced Steps

I don't think our K means and AHC models need any further improvements unless there is some change in the dataset features. It'd be interesting to run these models with more datapoints and same number of dimensions.

To enhance the clustering process in DBSCAN, we can use wine images as data instead of chemical characteristics of wines because this algorithm plays a very good role in computer vision applications. Or we can further refine my grid search and hyperparameters tuning function to find the best parameters, but this will require a lot of time as the model is quit CPU intensive. Increasing the number of datapoints while keeping the number of dimensions same might be helpful to this particular model.