CSE 5243 Intro To Data Mining
Wine Quality Data Analysis - Homework 2

Amanpreet Singh & Sarav Rajavelu

The objective of this report is to analyze read wine samples, build a kNN classification algorithm and evaluate the model on the dataset. The inputs include objective tests (e.g. PH values) and the output is based on sensory data (median of at least 3 evaluations made by wine experts). Each expert graded the wine quality between 0 (very bad) and 10 (very excellent).

This is a dataset of 1599 wine samples and their attributes. Following attributes are available in the dataset:

Wine Quality Dataset

Independent Variable

- **Fixed acidity**
- **Volatile acidity**
- **Citric acid**
- **Residual sugar**
- **Chlorides**
- **Free sulfur dioxide**
- **Total sulphur dioxide**
- **Density**
- **pH**
- **Sulphates**
- **Alcohol**

Dependant Variable

- **Quality**

Data Preprocessing

We replace the Quality variable with a binary class variable for the classification task according to the following rule :

Class : High when *Quality* > 5
Class : Low when *Quality* <= 5

1. Preliminary Data Analysis

1.1. Summary Statistics

We take a look at the summary statistics in *Table 1* to observe the spread of the data. We see that there are no missing values in the dataset.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
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	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

Table 1. Summary statistics

Looking at *Table.1*, the summary statistics of all the variables in the dataset, we notice that a few of the features are on a different scale compared to the rest. Therefore during in the model evaluation section, we test our kNN model on data that has been normalized to make the model less sensitive to the scale of certain features.

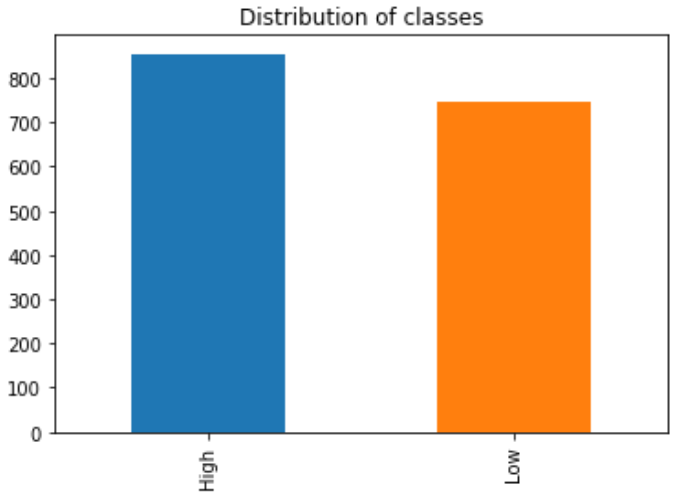

Fig 1. Distribution of Class

In Fig. 1, we see distribution of the dependent class. Both the classes have a fairly even representation (53% High/ 47% Low), therefore we do not have to an issue of only class being under-represented.

1.2. Correlation Matrix

We then look at the correlation matrix as seen in *Table. 2*, to check if any columns are highly correlated.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978	0.183006	-0.061668
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937	-0.260987	-0.202288
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904	0.312770	0.109903
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652	0.005527	0.042075
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026	0.371260	-0.221141
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	0.070377	0.051658	-0.069408
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-0.066495	0.042947	-0.205654
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-0.341699	0.148506	-0.496180
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1.000000	-0.196648	0.205633
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-0.196648	1.000000	0.093595
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	0.205633	0.093595	1.000000

Table 2. Correlation matrix

We see that there are a few columns with a correlation coefficient greater/lesser than 0.5/-0.5. Below in *Table. 3*, we have a list of the highly correlated attributes.

		Coefficient
pH	**fixed acidity**	-0.704507
fixed acidity	**pH**	-0.704507
volatile acidity	**citric acid**	-0.589286
citric acid	**volatile acidity**	-0.589286
pH	**citric acid**	-0.521270
citric acid	**pH**	-0.521270
free sulfur dioxide	**total sulfur dioxide**	0.660348
total sulfur dioxide	**free sulfur dioxide**	0.660348
density	**fixed acidity**	0.660497
fixed acidity	**density**	0.660497
citric acid	**fixed acidity**	0.699747
fixed acidity	**citric acid**	0.699747

Table 3. Highly correlated pairs

2. Data Transformations

2.1. Outlier Analysis

We plot a boxplot of all the independent features in the dataset to take a look at spread of the data and look out for outliers

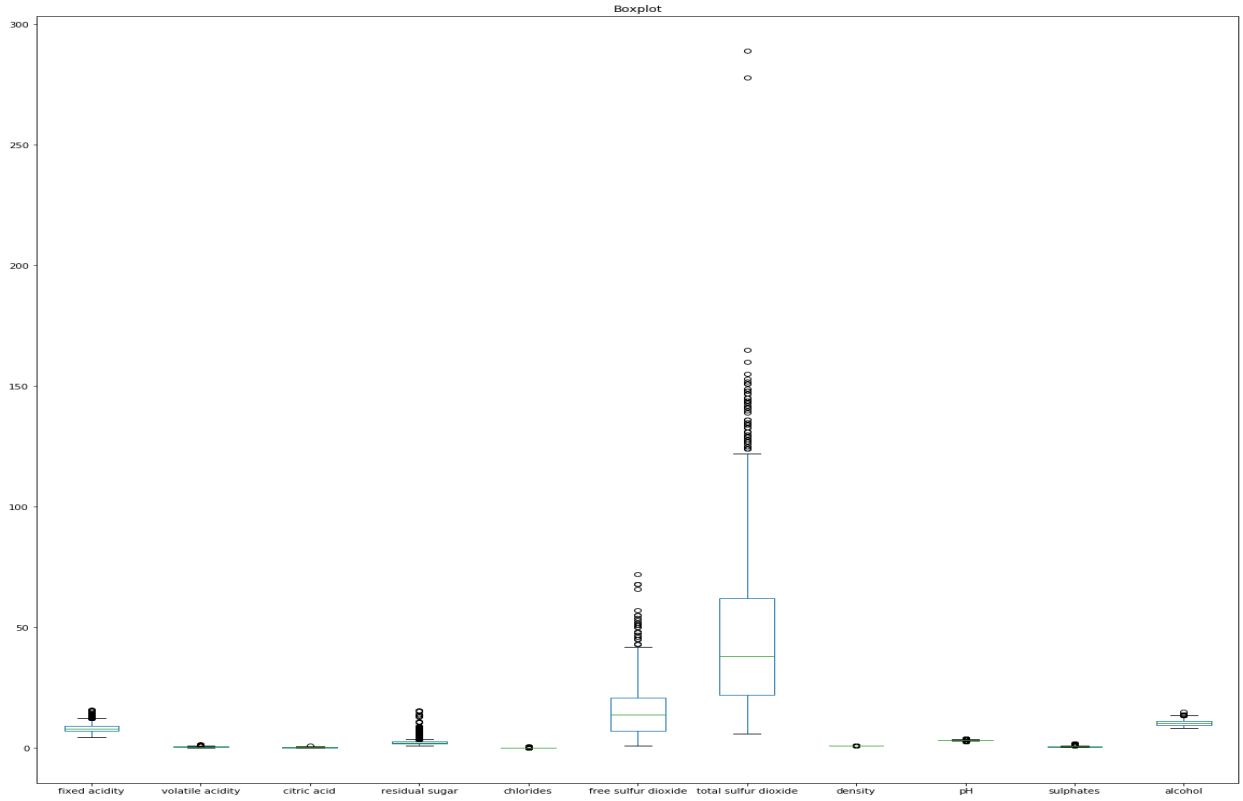

Fig. 2 - Boxplot of the independent variables

We see in Fig. 2, that there are outlier present in our dataset. We remove those records from our dataset where at least one attribute's value for that record is an outlier. Since some outlier convey valuable information to model, the threshold that we set to decide an outlier is if the value is greater than 3 standard deviations from the mean of the attribute.

2.2. Data Normalization

Looking at Fig.2, a boxplot of all the variables in the dataset, we notice that a few of the features are on a different scale compared to the rest. Therefore we need to normalize the data to make the model less sensitive to the scale of certain features.

For normalization we use a Min-Max normalization technique. We linearly transform a series X to Y, where Y = (X - min(X) / (max(X) - min(X)). Therefore the entire range of X is mapped to values between 0 and 1, both inclusive.

We normalize the training and testing data for all attributes X using the below formula.

X_Train` = (X_Train - min(X_Train) / (max(X_Train) - min(X_Train))
X_Test` = (X_Test - min(X_Train) / (max(X_Train) - min(X_Train))

In the model evaluation section we compare this normalized data again a dataset which has not been normalized.

2.3. Feature Selection

From Table 3, we see the list of highest correlated pairs. The following pairs are highly correlated.

- Fixed acidity and Citric acid: 0.699
- Fixed acidity and density: 0.66
- Free sulphur dioxide and total sulphur dioxide: 0.66

Since we don't want to highly correlated variables to skew the model, by counting their influence twice over, we remove two of these attributes.

We choose to remove **Fixed acidity** as it correlated to both citric acid and density. We also remove the feature **total sulphur dioxide**, we chose this over free sulphur dioxide because total sulphur dioxide is on a larger scale compared to the rest of the models as can be see from Fig 1.

3. Model Development

3.1. Data Partition

Partition the data of 1599 observations into train set with 75% of the data and a test set with 25% of the data.

3.2. Proximity Measure

We use the following two proximity measures to calculate the distance between two observations.

- **Euclidean Distance**: The Euclidean distance between points p and q is the length of the line segment connecting them. In Cartesian coordinates, if p = (p1, p2,..., pn) and q =

(q1, q2,..., qn) are two points in Euclidean n-space, then the distance (d) from p to q, or from q to p is given by the Pythagorean formula.[1]

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Fig 3. Euclidean Distance Formula[1]

- **Cosine Similarity**: Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Fig 4. Cosine Similarity Formula[2] (where A_i and B_i are components of vector A and B respectively)

3.3. kNN Implementation

Our implementation of kNN follows the algorithm description. Each test record and its attributes are compared with all other records in the training set using a specified distance or dissimilarity metric. The k records with the least distance to the test record constitute its neighborhood.

To predict a class and its posterior probability from this neighborhood we used two methods:

- **Majority Class**: This method simply picks the majority class from the neighborhood as the predicted class.
- **Distance Weighted Probabilities**: The contribution of each class in the neighborhood to the predicted class is weighted by the inverse square of its distance from the test record. This gives a higher weight to classes closer to the test record

	Actual	Predicted	Posterior
0	High	Low	0.673469
1	Low	High	0.510204
2	High	Low	0.673469
3	Low	Low	0.653061
4	Low	Low	0.591837
5	Low	Low	0.571429
6	Low	High	0.510204
7	High	Low	0.551020
8	High	Low	0.673469
9	Low	Low	0.632653
10	Low	Low	0.612245
11	Low	Low	0.591837
12	Low	Low	0.632653
13	Low	Low	0.734694
14	Low	High	0.653061
15	Low	High	0.571429
16	High	High	0.591837
17	High	High	0.775510
18	High	Low	0.530612
19	Low	High	0.673469
20	Low	Low	0.571429

Fig 2.2 kNN Output

4. Model Evaluation and Comparisons

4.1. Proximity Measure - Euclidean Distance vs Cosine Similarity

We start the model evaluation by comparing the accuracy and f-measure of kNN algorithm using euclidean distance versus cosine similarity for all odd valued K between 1 and 50.

In table 4 below we see the values of K with the top 10 highest f-measure.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
1	0.756098	0.243902	0.779412	0.807107	0.753555	0.753555	0.759494
23	0.658537	0.341463	0.698565	0.705314	0.691943	0.691943	0.613924
35	0.674797	0.325203	0.695431	0.748634	0.649289	0.649289	0.708861
31	0.666667	0.333333	0.694789	0.729167	0.663507	0.663507	0.670886
39	0.669377	0.330623	0.693467	0.737968	0.654028	0.654028	0.689873
29	0.663957	0.336043	0.693069	0.725389	0.663507	0.663507	0.664557
27	0.661247	0.338753	0.692875	0.719388	0.668246	0.668246	0.651899
3	0.658537	0.341463	0.692683	0.713568	0.672986	0.672986	0.639241
21	0.653117	0.346883	0.692308	0.702439	0.682464	0.682464	0.613924
37	0.672087	0.327913	0.692112	0.747253	0.644550	0.644550	0.708861

Table 4. Euclidean Distance kNN

In table 5 below we see the values of K with the top 10 highest f-measure.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
1	0.490515	0.509485	0.570776	0.550661	0.592417	0.592417	0.354430
39	0.490515	0.509485	0.562791	0.552511	0.573460	0.573460	0.379747
41	0.490515	0.509485	0.562791	0.552511	0.573460	0.573460	0.379747
47	0.490515	0.509485	0.562791	0.552511	0.573460	0.573460	0.379747
43	0.487805	0.512195	0.561485	0.550000	0.573460	0.573460	0.373418
45	0.487805	0.512195	0.561485	0.550000	0.573460	0.573460	0.373418
37	0.487805	0.512195	0.559441	0.550459	0.568720	0.568720	0.379747
49	0.487805	0.512195	0.559441	0.550459	0.568720	0.568720	0.379747
25	0.487805	0.512195	0.557377	0.550926	0.563981	0.563981	0.386076
27	0.487805	0.512195	0.557377	0.550926	0.563981	0.563981	0.386076
29	0.487805	0.512195	0.557377	0.550926	0.563981	0.563981	0.386076

Table 5. Cosine Similarity kNN

As we can see from Table 4 and Table 5, although the f-measure of the two models are comparable, the accuracy using euclidean distance is significantly higher than when we use cosine similarity as a proximity measure.

4.2. Normalized Data vs Non-Normalized Data

We run our Euclidean kNN algorithm on the min-max normalized dataset and we can see in Table 6, that the model lesser accuracy and f-measure than when we ran the Euclidean kNN algorithm on non-normalized data, Table 4.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
9	0.726287	0.273713	0.758950	0.764423	0.753555	0.753555	0.689873
45	0.728997	0.271003	0.754902	0.781726	0.729858	0.729858	0.727848
47	0.726287	0.273713	0.751843	0.780612	0.725118	0.725118	0.727848
13	0.720867	0.279133	0.751807	0.764706	0.739336	0.739336	0.696203
43	0.723577	0.276423	0.751220	0.773869	0.729858	0.729858	0.715190
27	0.715447	0.284553	0.749403	0.754808	0.744076	0.744076	0.677215
5	0.712737	0.287263	0.748815	0.748815	0.748815	0.748815	0.664557
25	0.718157	0.281843	0.748792	0.763547	0.734597	0.734597	0.696203
49	0.723577	0.276423	0.748768	0.779487	0.720379	0.720379	0.727848
35	0.715447	0.284553	0.748201	0.757282	0.739336	0.739336	0.683544

Table 6. Euclidean kNN - Min-Max Normalized

We also ran the Cosine similarity kNN algorithm on the min-max normalized data. However, the cosine variant performed poorly again, getting a highest accuracy of 41.1% when K = 3.

4.3. Majority Class (default) vs Weighted Distance Approach

We run our Euclidean kNN algorithm using the weighted distance approach on the min-max normalized dataset and we can see in Table 7, that the model achieves marginally lower (<2%) accuracy and f-measure than when we ran the Euclidean kNN algorithm using the majority class approach on the non-normalized dataset, Table 4.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
43	0.731707	0.268293	0.761446	0.774510	0.748815	0.748815	0.708861
17	0.726287	0.273713	0.760095	0.761905	0.758294	0.758294	0.683544
45	0.734417	0.265583	0.759804	0.786802	0.734597	0.734597	0.734177
13	0.726287	0.273713	0.758950	0.764423	0.753555	0.753555	0.689873
33	0.726287	0.273713	0.758950	0.764423	0.753555	0.753555	0.689873
37	0.728997	0.271003	0.758454	0.773399	0.744076	0.744076	0.708861
39	0.728997	0.271003	0.758454	0.773399	0.744076	0.744076	0.708861
27	0.723577	0.276423	0.758294	0.758294	0.758294	0.758294	0.677215
41	0.726287	0.273713	0.756627	0.769608	0.744076	0.744076	0.702532
25	0.723577	0.276423	0.755981	0.763285	0.748815	0.748815	0.689873

Table 7. Euclidean kNN - Min-max normalized - Weighted distance approach

We also ran the Cosine similarity kNN algorithm using the weighted distance approach on the min-max normalized data. The highest accuracy was 41.5% when K = 15.

In total we ran 8 variants of kNN algorithm, varying the following parameters.

- Proximity Measure: Euclidean distance vs Cosine similarity
- kNN approach: Majority class vs Weighted distance
- Normalized data vs non-normalized data

The two following models had similar highest accuracy

1. Euclidean kNN - Non - Normalized - Majority class approach
    a. Accuracy = 75.6%
    b. K value = 1
2. Euclidean kNN - Min-max normalized - Weighted distance approach
    a. Accuracy = 73.1%
    b. K value = 43

In spite of the majority class approach having a marginally higher accuracy than the weighted distance approach, we conclude that the **weighted distance approach** is the better model because it has K = 43, which makes is more robust than K = 1. K = 1 is more prone to outliers in the data.

4.4. Comparison with off the shelf model

We compare our best model, the **Euclidean kNN - Min-max normalized - Weighted distance approach** with an off the shelf implementation of kNN algorithm.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
1	0.7150	0.2850	0.750000	0.730769	0.770270	0.770270	0.646067
3	0.6900	0.3100	0.726872	0.711207	0.743243	0.743243	0.623596
5	0.6825	0.3175	0.710706	0.718894	0.702703	0.702703	0.657303
7	0.6900	0.3100	0.707547	0.742574	0.675676	0.675676	0.707865
21	0.6900	0.3100	0.707547	0.742574	0.675676	0.675676	0.707865
11	0.6850	0.3150	0.704225	0.735294	0.675676	0.675676	0.696629
9	0.6800	0.3200	0.702326	0.725962	0.680180	0.680180	0.679775
27	0.6750	0.3250	0.696262	0.723301	0.671171	0.671171	0.679775
19	0.6775	0.3225	0.695035	0.731343	0.662162	0.662162	0.696629
17	0.6725	0.3275	0.690307	0.726368	0.657658	0.657658	0.691011
23	0.6725	0.3275	0.690307	0.726368	0.657658	0.657658	0.691011
25	0.6725	0.3275	0.690307	0.726368	0.657658	0.657658	0.691011

Table 8. Scikit Learn Implementation of kNN

We used euclidean distance as a measure of proximity and uniform weights for all points in the neighbourhood. We chose these settings are it represents the parameters of our **Euclidean kNN - Non - Normalized - Majority class approach** model. There it would be an apples to apples comparison.

As we can see from Table 6 and Table 4, our **Euclidean kNN - Non - Normalized - Majority class approach** out performs the off the shelf model.

4.5. Comparison with reducing the dataset to 50%

We run our **Euclidean kNN - Non - Normalized - Majority class approach** approach only using 50% of the data. Table 9, shows the results.

K	accuracy	error_rate	f_measure	precision	recall	sensitivity	specificity
27	0.540	0.460	0.629032	0.545455	0.742857	0.742857	0.315789
35	0.515	0.485	0.622568	0.526316	0.761905	0.761905	0.242105
45	0.500	0.500	0.621212	0.515723	0.780952	0.780952	0.189474
49	0.500	0.500	0.621212	0.515723	0.780952	0.780952	0.189474
25	0.535	0.465	0.617284	0.543478	0.714286	0.714286	0.336842
31	0.515	0.485	0.616601	0.527027	0.742857	0.742857	0.263158
33	0.510	0.490	0.614173	0.523490	0.742857	0.742857	0.252632
13	0.560	0.440	0.614035	0.569106	0.666667	0.666667	0.442105
47	0.485	0.515	0.611321	0.506250	0.771429	0.771429	0.168421
5	0.540	0.460	0.606838	0.550388	0.676190	0.676190	0.389474
43	0.490	0.510	0.604651	0.509804	0.742857	0.742857	0.210526

Table 9. Euclidean kNN - Non - Normalized - Majority class approach with only 50% data

We can clearly see that loss of data results in lower accuracy of the model. A model like kNN that is a lazy learner would perform better if there are more data points for it to use to classify new data.

5. References

[1] https://en.wikipedia.org/wiki/Euclidean_distance
[2] https://en.wikipedia.org/wiki/Cosine_similarity