k-Means Clustering

The objective of this report is to describe the implementation of a k-Means algorithm and discuss the design decisions. We test our k-Means algorithm on the TwoDim dataset, comparing the results of our clustering algorithm with the true cluster membership of records. Next, we run our clustering algorithm on the Wine dataset and discuss the performance of our algorithm. Finally, we use off-the-shelf algorithms to cluster the datasets mentioned above and compare the performance of the models.

1. k-Means Algorithm

1.1. Introduction

The k-means algorithm is a popular clustering algorithm that aims to partition all observations into k clusters. Each cluster is identified by its centroid. Each observation belongs to one cluster and its membership is determined by the least distance to the centroid of a cluster, compared with the distance to the centroids of all the other clusters. During clustering we want our model to create clusters such that cluster cohesion is low and cluster separation is high.

Evaluation Metrics:
Cluster Cohesion is measured by 'within-cluster sum of squares' (SSE).

$$Cluster\,Cohesion = \Sigma_{x \in C_i} dist(c_i, x)^2$$

Cluster Separation is measured by how well a cluster is separated from other clusters.

$$Cluster\,Separation = \Sigma_{i=1}^{k} m_i dist(c_i, c)^2$$

Where m_i is the size of the cluster.

1.2. Implementation

We take data and k as parameters to the function, where k is the number of centroids. The coordinates of the centroids is an optional parameter. We use this parameter to test various centroid initialization techniques.

We then follow the steps below:

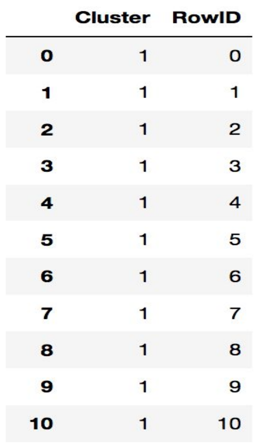
Step 1: If centroids are not passed to the function using the optional parameter, we randomly initialize k centroids in the same dimensional space as records in the data. Each centroid is associated with a cluster class (in our case 1, 2, … K classes).

Step 2: Assign each record in the data a cluster class, which is the class of the centroid nearest to it. We do so by comparing the distance of each observation from all the centroids and choosing the centroid with the least euclidean distance from the observation.

Step 3: After all the observations have been assigned a cluster class, we update each centroid to be the mean of all the observations in its cluster.

Step 4: We repeat Step 2 and Step 3 until the centroids do not change.

Additionally we output the coordinates of the centroids, number of runs of step 2 and 3 and the cluster member of all the points.



	Cluster	RowID
0	1	0
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10

Table 1. Sample Output

1.3. Design Decisions

1.3.1. Initializing Centroids

We try three different methods for initializing the centroids.

Random Initialization

We initialize the centroids by randomly choosing values, such that each dimension's value lies between the minimum and maximum values of that corresponding dimension in the dataset. This method does not ensure that every cluster class will have at least one record.

Random Sampling

We initialize the centroids by randomly choosing k samples from the dataset. This method ensures that each cluster class will at least have one record.

Sampling with Conditions

We start by choosing k random samples from the dataset as centroids. We then proceed to check if other observations from the dataset are better candidates as initial centroids.

For each observation 'j' in the dataset we do the following:

- Condition 1: If the distance between observation j and its closest centroid (c1) is greater than the distance between the two closest centroids (c2 & c3), j substitutes either c2 or c3 as an initial centroid depending on which one it is closer to.

- Condition 2: If the distance between observation j and its second closest centroid (say c4) is greater than the distance between the centroid closest to j (c1) and centriod closest to c4 (c5), then j substitutes the centroid closest to it, c1, as one of the initial centroids.

If condition 1 is not satisfied then condition 2 is checked. If both fail, the jth observation does not become an initial centroid. This approach diminishes the chance of getting trapped into a "local optimum", and speeds up the process of convergence.

2. Dataset 1 : TwoDimHard

The dataset has 400 records and 3 features. For this dataset the feature ‘cluster’ represents the true cluster membership of the record.

Features

- **X.1**
- **X.2**

True Class Membership

- **Cluster**

2.1. Data Processing and Transformations

We do not perform data preprocessing and transformations on the TwoDimHard dataset. As the data is of the similar scale seen in the box plot in Fig 1.

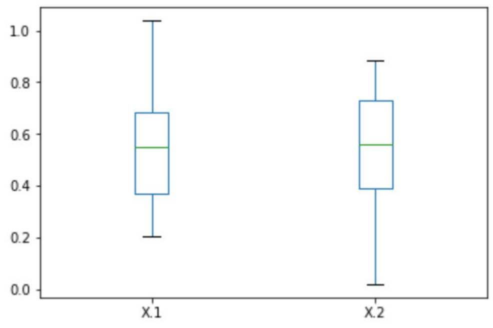


Fig. 1 - Boxplot of the independent variables

2.2. True Cluster Membership

Since we have the cluster labels for the TwoDimHard dataset, we calculate the cluster cohesion and cluster separation.

Cluster evaluation:

- Cluster Separation: 95.4
- Cluster Cohesion: 39.6

- Cluster 1: 4.5
- Cluster 2: 8.41
- Cluster 3: 13.73
- Cluster 4: 12.95

Below in Fig 2., we can see the plot of the TwoDimHard dataset and each point's respective cluster membership.

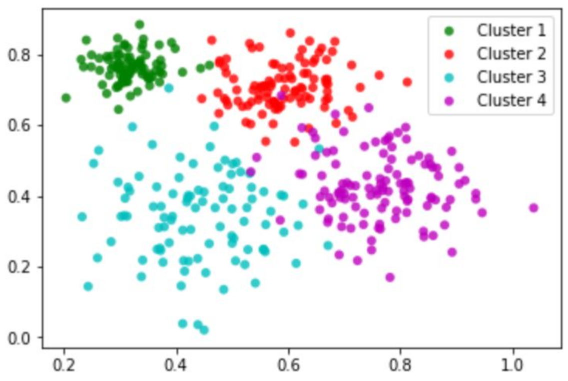


Fig. 2 - True Cluster Membership of TwoDimHard dataset

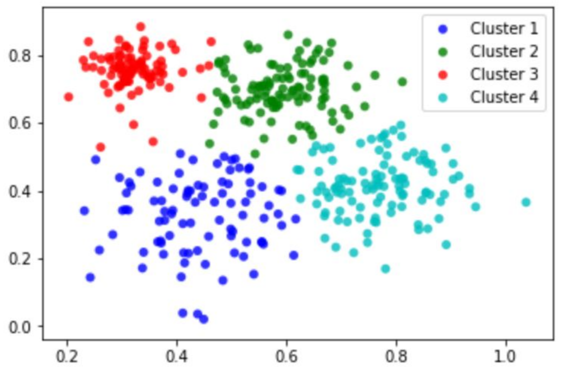


Fig. 3 - Cluster Membership of TwoDimHard dataset based on our algorithm k = 4

2.3. K-Means Algorithm

For K = 4

We ran K means using each of our centroid initialization techniques for 100 runs. We found that on average our random initialization technique outperformed other methods in both having lower cluster cohesion (Mean Cluster Cohesion: 41.21) and higher cluster separation (Mean Cluster Separation: 95.02). We feel that this method of initialization works better due to the nature of the data, that is, the range and spread of the two features are similar.

The model's best cluster performance is achieved 75% of the time.
Best cluster performance

- Cluster Separation: 96.34
- Cluster Cohesion: 37.97
    - Cluster 1: 5.53
    - Cluster 2: 9.48
    - Cluster 3: 11.77
    - Cluster 4: 11.19

*Note: Cluster numbers have been changed from the plot, such that same clusters from Fig 2 and Fig 3 have the same cluster numbers as it is easier for evaluation.

Above in Fig 3., we can see the plot of the TwoDimHard dataset and each point's respective cluster membership as calculated by our clustering algorithm.

For K = 3

We ran K means using each of our centroid initialization techniques for 100 runs. We find again that on average our random initialization technique outperformed other methods in both having lower cluster cohesion (Mean Cluster Cohesion: 53.87) and higher cluster separation (Mean Cluster Separation: 88.8). As stated above, we feel that this method of initialization works well better due to the nature of the data, that is, the range and spread of the two features are similar.

The model's best cluster performance is achieved 39% of the time.

- Cluster Separation: 90.63
- Cluster Cohesion: 53.88
    - Cluster 1: 27.85
    - Cluster 3: 12.22
    - Cluster 4: 13.81

*Note: Cluster numbers have been changed from the plot. We've skipped Cluster 2 (as Cluster 1 for K = 3, covers Cluster 1 & 2 for K=4) to get a better like-for-like comparison.

Below in Fig 4., we can see the plot of the TwoDimHard dataset and each point's respective cluster membership as calculated by our clustering algorithm for K = 3.

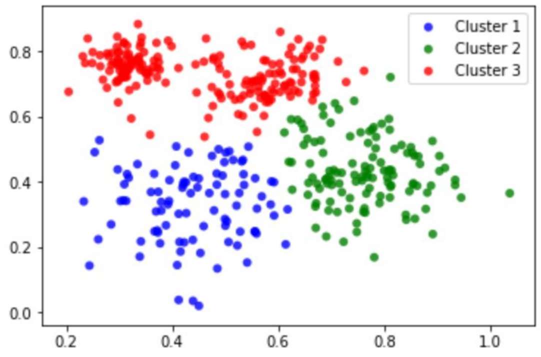


Fig. 4 - Cluster Membership of TwoDimHard dataset based on our algorithm k = 3

2.4. Model Comparison

K-Means (K = 4) and True Cluster Class

K-Means generates clusters with greater cluster separation and lower cluster cohesion than the true class labels.

K-Means (K = 4)

- Cluster Separation: **96.34**
- Cluster Cohesion: **37.97**

True Cluster Class

- Cluster Separation: 95.4
- Cluster Cohesion: 39.6

To the right in Table 2., we can see the cross tabulation matrix comparing the actual and assigned clusters. From the table we can see that the K-Means clustering algorithm got the cluster predictions correct **94.75%** times (379 out of 400). Looking at Fig 2., we can see a few cases where dots of one color are engulfed by another cluster. For instance a few pink dots are inside red and blue clusters. Using the K-Means algorithm, we will never have stray pink dots inside red and blue clusters, as those stray pink dots will be assigned to red and blue clusters respectively. Therefore, the algorithm will not be able to assign the correct clusters for those observations.

True Clusters	**1**	**2**	**3**	**4**
Assigned Cluster				
1	89	2	4	0
2	0	98	2	8
3	0	0	88	2
4	0	0	3	104

Table 2 - Cross tabulation comparing the actual and assigned clusters

K-Means (K = 4 and K = 3)

Below we have the evaluation characteristics for best models of K = 4 and K = 3.

K-Means (k = 4)

- Cluster Separation: **96.34**
- Cluster Cohesion: **37.97**

K-Means (k = 3)

- Cluster Separation: 90.63
- Cluster Cohesion: 53.88

True Clusters	**1**	**2**	**3**	**4**
Assigned Cluster				
1	89	94	5	1
3	0	0	89	3
4	0	6	3	110

Table 3 - Cross tabulation comparing the actual and assigned clusters for K = 3, cluster number changes to match true cluster membership

Looking at the cross tabulation results in Table 3, we see that the cluster in red (in Fig 4) covers two clusters from the true cluster membership, cluster 1 and cluster 2. This is expected as we have used one less cluster than the number of true clusters. K-Means with K = 4 generates clusters with lower cluster separation and greater cluster cohesion compared to the true class labels.

Increasing the number of clusters changes the results for the better as we can see from Fig 5., on the right. For low values of K, more general clusters are formed. For higher values, more granular clusters are formed. Therefore, while increasing the number of clusters from 3 to 4 improves the model, we have to carefully evaluate the model before increasing the K value.

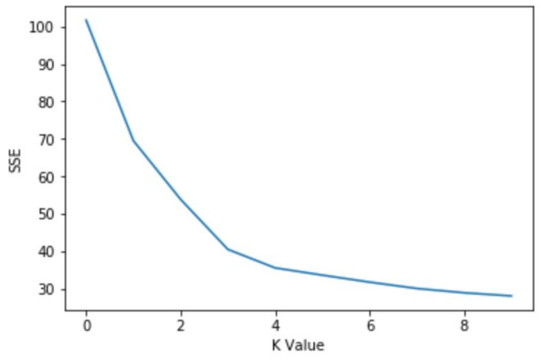


Fig 5. SSE vs K Value Graph

2. Dataset 2 : Wine Data

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

Features

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

True Class Membership

- **Quality**

3.1. Data Processing and Transformations

Feature Selection

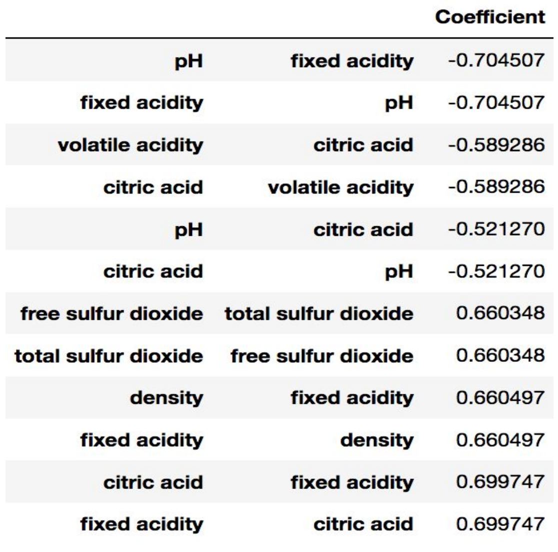
In Table 4 we see a list of highly correlated pairs. The following pairs are most highly correlated.

- **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 highly correlated variables to skew the model by counting their influence twice over, we remove three of these attributes.

We choose to remove both citric acid and density as they correlate with fixed acidity. We also remove the feature total sulphur dioxide. We removed this over free sulphur dioxide because total sulphur dioxide is on a larger scale compared to the rest of the features.

We plot a boxplot of all the features in the dataset to take a look at spread of the data. Looking at Fig.6, we notice that a few 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.



		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 4. Highly correlated pairs

Min-Max normalization

We linearly transform a feature X to Y, where Y = (X - min(X) / (max(X) - min(X)). Therefore the entire range of feature X is transformed 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))

Softmax normalization.

Softmax normalization is a way of reducing the influence of extreme values or outliers in the data without removing them from the dataset. The function limits the range of the **normalized** data to values between 0 and 1.

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

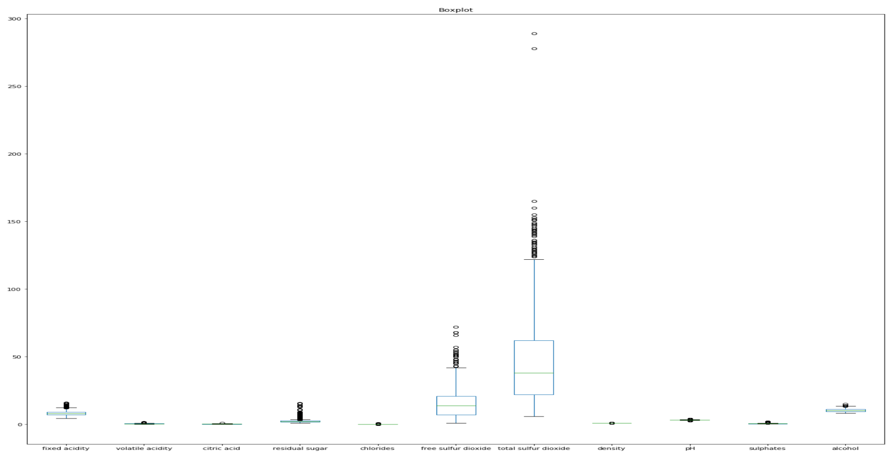


Fig. 6 - Boxplot of the independent variables

3.2. Model Evaluation

For each of the 3 different forms of our dataset: base dataset, min-max normalized and softmax normalized, we run K-Means algorithm using each of our centroid initialization techniques. We see that using random value centroid initialization typically results in only 2 or 3 clusters as compared to sampling with condition, where all clusters are represented at least once. For each value of K, we see from Fig 7. that random initialization is unreliable and is also outperformed by sampling with condition based on cluster cohesion.

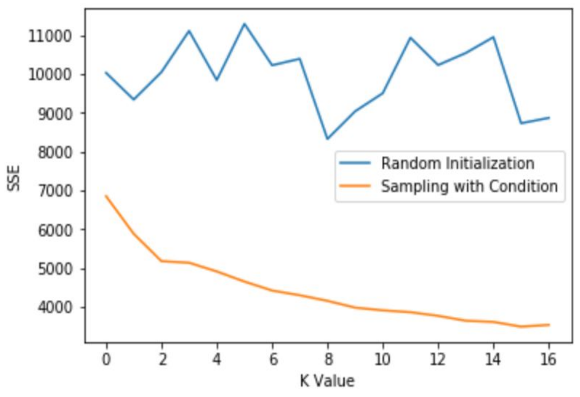


Fig 7. Random Initialization vs Sampling with Condition

For each of the 3 different forms of our dataset, we run the K-Means algorithm across different values of K and compare the results. From Fig 8, we can see that the softmax normalized data has the best cluster cohesion among the 3 for all values of K starting from 3 to 20.

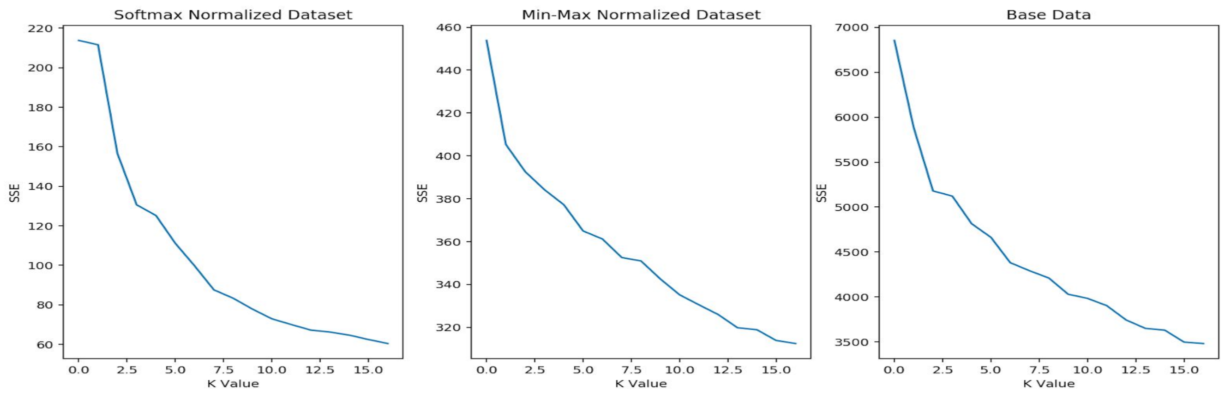


Fig 8. Comparison of Normalization Techniques

We take k = 6 as our ideal value as:

- In spite of the quality feature of the data ranging from 1-10, only 6 of those values (3, 4, 5, 6, 7 & 8) are represented by the observations. This helps us compare our K-Means with that of the quality feature as true cluster class of wine dataset.
- We do not want to choose a higher value of k, as we want to learn more general cluster trends.

Cluster	fixed acidity	volatile acidity	residual sugar	chlorides	free sulfur dioxide	pH	sulphates	alcohol
1	7.693493	0.597046	2.203425	0.085736	11.517979	3.343647	0.598938	9.727568
2	8.371429	0.542857	1.975000	0.351464	15.464286	3.041786	1.309286	9.446429
3	11.108681	0.425990	2.632465	0.087438	10.333333	3.161701	0.713958	10.558218
4	8.197222	0.530972	9.313889	0.103528	34.638889	3.258056	0.646944	10.002778
5	7.403125	0.498239	2.500710	0.072960	12.818182	3.391506	0.658182	11.831013
6	7.959486	0.523891	2.391479	0.081534	30.512862	3.327781	0.660289	10.146517

Table 5. Cluster Analysis.

Using k = 6 as our ideal value, we analyse the data. In Table 5 above we have the mean values of all features for all clusters. We observe the following trends in the data (Fig 9):

- **Cluster 3** (in **cyan**) stands out with the **highest fixed acidity** and **lowest free sulfur dioxide** amongst all the clusters.
- **Cluster 4** (in **red**) can be identified by its **significantly high value for residual sugar** and **high value for free sulfur dioxide**.
- **Cluster 2** (in **green**) has the **highest chlorides** and **sulphates** amongst all the clusters.
- **Cluster 6** (in **yellow**), like cluster 4 has **high value for free sulfur dioxide** but has a much **lower value of residual sugar**.
- **Cluster 5** (in **pink**) has the highest pH values and the lowest chlorides and fixed acidity values.
- **Cluster 1** (in **blue**) has the **lowest sulphates** and **low alcohol** and **chlorides values**.

As we can see, each cluster has a unique identity that differentiates it from other clusters. Below in Fig 9., we plot a pairwise scatter plot of all the features that were part of the analysis to observe the trends described above.

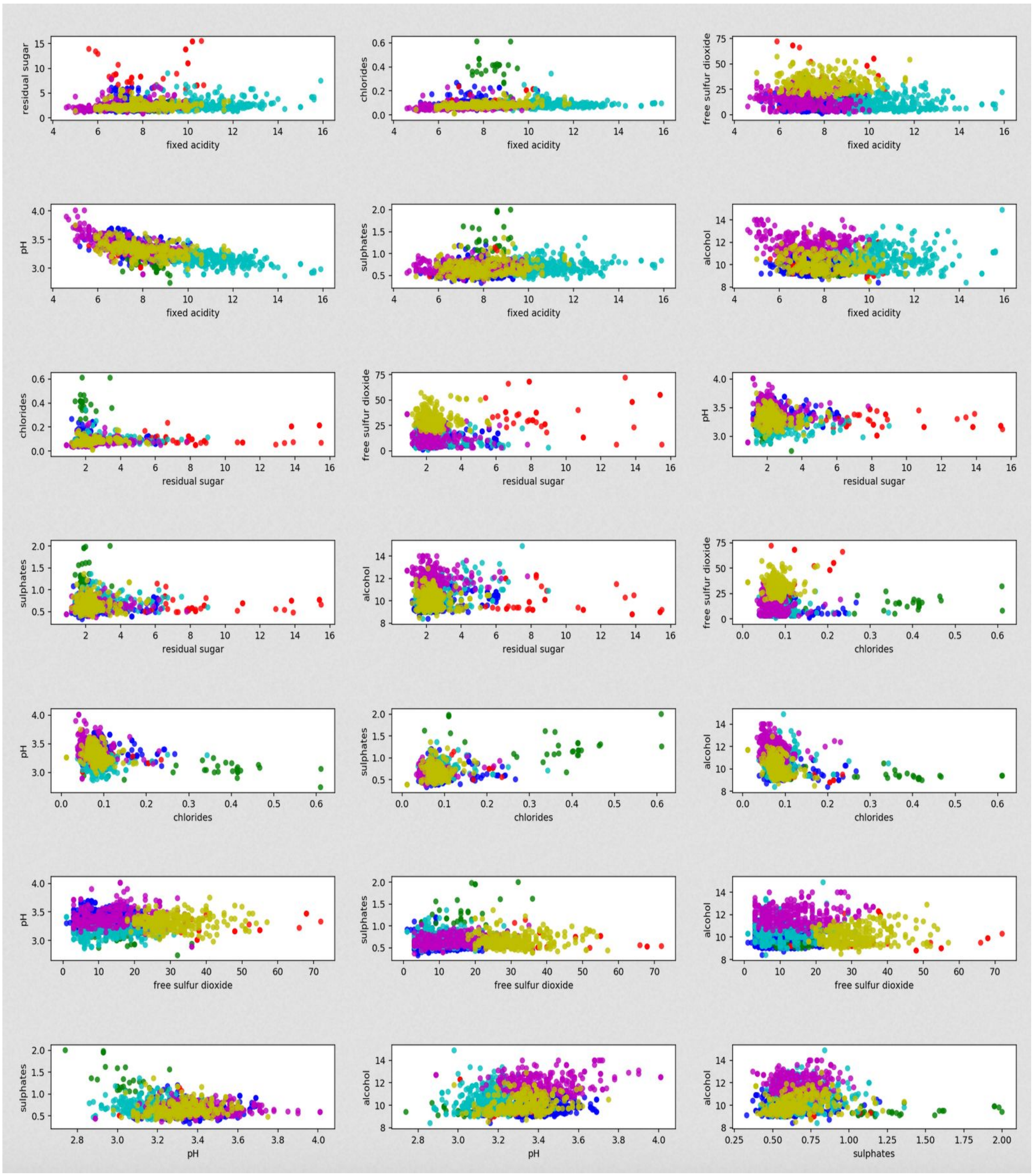


Fig 9. Pairwise Scatter Plot of Features with Clusters

3.3. Model Evaluation using Quality as True Class Membership

Since there are only 6 classes represented in by the quality feature in the wine dataset, we choose K = 6 for the comparison. K-Means generates clusters with greater cluster separation and lower cluster cohesion than the quality class labels.

K-Means (k = 6)

- Cluster Separation: **909.11**
- Cluster Cohesion: **86.52**

True Cluster Class

- Cluster Separation: **117.72**
- Cluster Cohesion: **893.48**

4. Off-the-shelf Algorithms

4.1. TwoDimHard Dataset

We use the K-Means implementation from sklearn package in Python using the parameters given below:

- **n_cluster**: 4 as there 4 true classes in the dataset
- **init:** Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- **n_init**: Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- **max_iter**: Maximum number of iterations of the k-means algorithm for a single run.

Evaluation

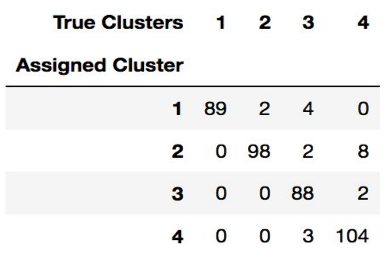
Our implementation of K-Means generates clusters with lower cluster separation and greater cluster cohesion.

K-Means (k = 4)

- Cluster Separation: **96.34**
- Cluster Cohesion: **37.97**

Off the shelf (k = 4)

- Cluster Separation : **83.031905**
- Cluster Cohesion : **137.662537**



True Clusters	1	2	3	4
Assigned Cluster				
1	89	2	4	0
2	0	98	2	8
3	0	0	88	2
4	0	0	3	104

Table 4 - Cross tabulation comparing the actual and assigned clusters

From Table 4 we can see that the K-Means clustering algorithm got the cluster predictions correct **94.75% (**379 times out of 400) of the time. Same as result from our implementation.

4.2. Wine Dataset

4.2.1. K-Means

We use the K-Means implementation from sklearn package in Python using the parameters given below:

- **n_cluster**: 6 as there 6 true class in the dataset
- **init:** Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- **n_init**: 10, Number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- **max_iter**: 200, Maximum number of iterations of the k-means algorithm for a single run.

Evaluation

K-Means (k = 6)

- Cluster Separation: **909.11**
- Cluster Cohesion: **86.52**

Off the shelf (k = 6)

- Cluster Separation : **491.275252**
- Cluster Cohesion : **479.915766**

4.2.2. Agglomerative Clustering

Next, we use Agglomerative Clustering algorithm. This algorithm recursively merges the pair of clusters that minimally increases a given linkage distance. It starts with the points as individual clusters and at each step and merge the closest pair of clusters until only k clusters remain.[1]

Parameters:

- **n_cluster**: 6 as there 6 true classes in the dataset.
- **affinity**: We choose euclidean distance as the proximity measure between two clusters. It performed better than manhattan and cosine distance.
- **compute_full_tree**: Parameter used to decrease computational time.
- **linkage**: Ward as it minimizes the variance of the clusters being merged.

Off the shelf Agglomerative Clustering (k = 6)

- Cluster Separation : **475.22**
- Cluster Cohesion : **87.884**

4.2.3. Birch Clustering

We tried using the birch clustering algorithm that is an online-learning algorithm provided as an alternative to MiniBatchKMeans. It constructs a tree data structure with the cluster centroids being read off the leaf. The tree data structure consists of nodes with each node consisting of a number of subclusters. The maximum number of subclusters in a node is determined by the branching factor. Each subcluster maintains a linear sum, squared sum and the number of samples in that subcluster. In addition, each subcluster can also have a node as its child, if the subcluster is not a member of a leaf node.[2]

However, the model failed to recognize more than 2 clusters. Therefore, we chose to implement the MiniBatchKMeans Clustering as our final clustering method.

4.2.4. MiniBatchKMeans Clustering

The MiniBatchKMeans is a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.[3]

Parameters:
- **Batch_size =** 100
- **n_cluster**: 6 as there 6 true class in the dataset
- **Init:** Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- **N_init**: 10 : Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- **Max_iter**: 200: Maximum number of iterations of the k-means algorithm for a single run.

Off the shelf MiniBatchKMeans Clustering (k = 6)
- Cluster Separation : **492.244**
- Cluster Cohesion : **600.75**

5. References

[1]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering
[2]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html#sklearn.cluster.Birch
[3]http://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html#sklearn.cluster.MiniBatchKMeans