Comparing K-means distance measures Robustness to noise

Dhruv Patel
Department of Computer Science
Pace University
New York, USA
dp99977n@pace.edu

Sung-Hyuk Cha
Department of Computer Science
Pace University
New York, USA
scha@pace.edu

***Abstract*— This research paper investigates the robustness of Kmeans distance measures to noisy data. A common clustering approach called Kmeans divides a dataset into K groups based on how similar the data points are to one another. The existence of noise in the data, however, can have a major impact on Kmeans' performance. we evaluate the robustness of various distance measures used in Kmeans, including Euclidean, Manhattan, and Cosine distances, to different levels of noise in the data. Our experimental findings demonstrate that the choice of distance measure has a significant impact on the performance of Kmeans, and evaluating the best distance measure.**

I. Introduction

Clustering and classification are fundamental tasks in supervised as well as unsupervised machine learning models. The application spectrum for Kmeans is vast including use cases as image segregation, customer segregation, Anomaly detection, Recommendation system, DNA micro array analysis and so on. Clustering has been the ML engineer's choice when it comes to finding entities that fall similar to a group of other entities. Knn and Kmeans had been a popular choice as these algorithms cluster data based on their nearest K neighbours, and due to the nature of the algorithm, they can also be used to classify the data, unlike DBSCAN which we will discuss in short while.

The common approach for all Machine learning modelling projects starts with data pre-processing, which include a crucial task of data cleaning. Data cleaning tasks include, null value handling, type checking, outliers and so on. One part where enough emphasis is required is outlier detection, for many datasets because of their nature, it gets confusing to decide whether the data is actually a noise or an extremely motivated characteristic of an entity. This requires lot more research on data which is costly for a project manager. There are approaches such as using DBSCAN that can deal with noisy data but due to the functionality of DBSCAN, classifying the data to their respective class quite impossible in many cases.

In this paper, we researched on weather it a good approach for a organisation to not invest time in researching the outlier or they should accept the outlier and get on modelling as Knn and Kmeans use multiple K neighbours so the outlier will be under shadowed by other true points in the K collection of any datapoint.

II. Literature Review

A. *K – means and Knn Algorithm*

Two popular machine learning methods for data analysis are K-means clustering and K-nearest neighbours (Knn). While they both use distance measurements to group data points, they serve different purposes and employ different strategies. The comparison of the K-means and KNN algorithms is the main subject of this literature study, with a focus on their robustness to noisy data.

The KNN classification method labels each data point based on the labels of its K closest neighbours in the training data. The method chooses the K closest points by calculating the distances between the data point and each training data point. The majority label of the data point's K nearest neighbours is then used to determine the label for the data point. KNN is robust to training data noise, but it is susceptible to test data noise, which can result in misclassification. Several strategies, such as the use of weighted distances or the adjustment of local density estimates, have been suggested to increase the robustness of KNN to noisy data.

K-means, on the other hand, divides a set of data points into K clusters according to how similar they are. The algorithm updates the cluster canter based on the mean of the data points in each cluster after iteratively assigning each data point to the closest cluster canter. While K-means is good at finding clusters in data, it is also sensitive to data noise, which can cause spurious clusters to form or separate clusters to merge. The *Canberra* distance and other alternative distance measures, as well as the use of robust statistics in place of the mean, have all been suggested as ways to increase the resilience of K-means to noisy data.

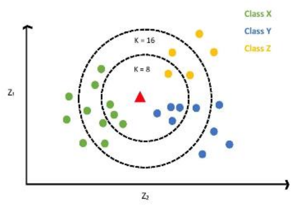

Image 1. Kmeans

In general, K-means and KNN use distance measures similarly, although their goals and approaches are different. KNN is utilized for classification, but K-means is frequently used for clustering but is also used for classification. Both algorithms are sensitive to data noise, but methods have been put forth to make them more robust. In general, the particulars of the work and the features of the data will determine the method and strategy to be used.

B. *DBSCAN*

Density-Based Spatial Clustering of Applications with Noise is a popular clustering algorithm in machine learning that is known for its ability to handle datasets with arbitrary shapes and sizes. In this examination of the literature, we will discuss DBSCAN's salient characteristics, as well as its advantages and disadvantages in relation to the K-means method.

DBSCAN clusters together points that have similar densities. A core point is one that the algorithm considers having a minimal number of neighbours within an established range (epsilon). The same cluster is thought to contain all points that are within the epsilon distance of a core point. The technique is also capable of locating noise points that do not form a cluster.

DBSCAN does, however, have some restrictions. Its inability to categorize data into predetermined groups or classes is one of its key drawbacks. DBSCAN is typically used for clustering, while K-means can be used for classification. The provision of two important parameters, epsilon and the minimum number of points necessary to create a cluster, is also needed by DBSCAN. The selection of these parameters may involve some trial and error and may have a substantial impact on how well the algorithm performs. Due to its more intricate calculations, DBSCAN generally performs slower than K-means.

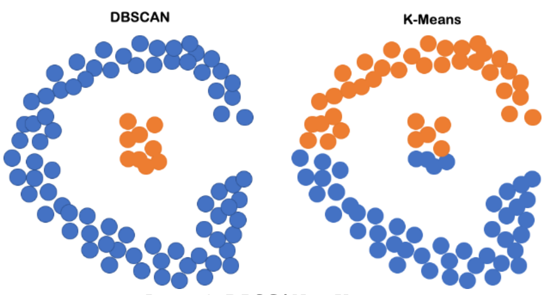

Image 2. DBSCAN vs Kmeans

In conclusion, even though DBSCAN offers a number of advantages, such as the capacity to handle datasets of any shape or size and its resistance to noisy data, it cannot always be used in place of K-means. For categorization, K-means can be employed and is typically quicker than DBSCAN. The particular task at hand and the properties of the data will determine which algorithm is used.

C. *Elbow method*

The elbow approach is a well-liked way for figuring out how many clusters in a dataset are ideal for algorithms like K-means clustering. We will examine the elbow method's main characteristics, its advantages and disadvantages, and how it stacks up against other methods for figuring out the ideal number of clusters in this literature survey.

The elbow method works by plotting the within-cluster sum of squares (WCSS) against the number of clusters. The total of the squared distances between each data point and the designated cluster center is what the WCSS calculates. The point of inflection on the elbow-shaped plot of the WCSS against the number of clusters represents the ideal number of clusters. This is due to the fact that decreasing the number of clusters below this point causes a marked increase in the WCSS while increasing the number of clusters above it causes a diminishing reduction in the WCSS.

The simplicity and usability of the elbow approach are two of its main benefits. Implementing the procedure is simple and doesn't call for a lot of computing power. The elbow method is also popular and has been utilized with numerous datasets and clustering techniques.

The silhouette method and the gap statistic are two other approaches that have been suggested to find the ideal number of clusters. The gap statistic compares the WCSS of the observed data to a null reference distribution produced by a random process, whereas the silhouette approach assesses how similar a data point is to its own cluster in comparison to other clusters.

$$\mathrm{WCSS} = \sum_{C_k}^{C_n} \left(\sum_{d_i \, in \, C_i}^{d_m} distance(d_i, C_k)^2 \right)$$

Image 3. WCSS

The elbow approach is a popular and straightforward method for figuring out the ideal number of clusters in a dataset. Despite few drawbacks, it is still a common alternative because of how simple it is to use and how widely it is used. For some datasets and clustering algorithms, alternative techniques, including the silhouette approach and gap statistic, can also be used to estimate the ideal number of clusters.

D. *Euclidean Distance*

A common distance metric used in machine learning to assess how similar two data points are is euclidean distance. The straight-line distance in Euclidean space between two places is known as the Euclidean distance. The total of the squared differences between each coordinate of the two points is used to calculate it. In clustering algorithms like K-means, hierarchical clustering, and DBSCAN, Euclidean distance is frequently used.

$$\sqrt{\sum_{\mathbf{i=1}}^{\mathbf{i=n}} (\mathbf{x_i} - \mathbf{y_i})^2}$$

Image 4. Euclidian Distance

Euclidean distance's simplicity and use are two of its main benefits. The computation is simple, widely accepted by the machine learning community, and easily understood. Euclidean distance is also simple to understand and intuitive. However, there are some restrictions on Euclidean distance. Its presumption that all aspects or features of the data are equally significant is one of its limitations. Euclidean distance may not always produce the best results when some dimensions are more crucial than others. Furthermore, Euclidean distance may not function well in datasets with high-dimensional spaces and may be sensitive to outliers.

E. *Squared Euclidian Distance*

A variation of Euclidean distance, frequently employed in machine learning techniques like K-means clustering. The total of the squared differences between each coordinate of two data points is used to determine the squared Euclidean distance between two points. The squared Euclidean distance is faster to calculate than the Euclidean distance because it is not rooted. When speed is a factor, Squared Euclidean distance is frequently used as a rough approximation of Euclidean distance.

The squared Euclidean distance's computational efficiency is one of its key benefits. The computation of the distance metric is quicker than Euclidean distance since it does not involve the square root. Additionally, because squared differences magnify the impact of significant deviations, Squared Euclidean distance is less susceptible to outliers than Euclidean distance.

$$\|x - y\| = \sum_{i=1}^{d} (x_i - y_i)^2$$

Image 5. Squared Euclidian Distance

Squared Euclidean distance is quicker to compute than Euclidean distance, but it does not maintain the actual distances between data points as well. In general, Euclidean distance is more precise and better suited for low-dimensional datasets when computational performance is unimportant.

F. *Manhattan Distance*

Manhattan distance, also referred to as taxicab distance, is a distance measure that is frequently employed in machine learning techniques like clustering and K-nearest neighbours. The total of the absolute differences between each coordinate of two data points is used to determine the Manhattan distance. It is less susceptible to outliers than Euclidean distance because it does not calculate with squares or square roots. Because it does not distort distances in high-dimensional space, the Manhattan distance is particularly helpful when working with high-dimensional data.

Manhattan distance, also referred to as taxicab distance, is a distance measure that is frequently employed in machine learning techniques like clustering and K-nearest neighbours. The total of the absolute differences between each coordinate of two data points is used to determine the Manhattan distance. It is less susceptible to outliers than Euclidean distance because it does not calculate with squares or square roots. Because it does not distort distances in high-dimensional space, the Manhattan distance is particularly helpful when working with high-dimensional data.

$$\sum_{i=1}^{n} |x_i - y_i|$$

Image 6. Manhattan Distance

In conclusion, Manhattan distance is a straightforward and uncomplicated distance metric that is appropriate for high-dimensional data and less sensitive to outliers. It does, however, have limitations in that it does not take feature correlations into account and treats each characteristic equally. For some datasets and machine learning tasks, other distance measures might be better suitable.

G. *Chebyshev Distance*

The Chebyshev distance metric, usually referred to as the maximum distance or L∞ distance, is frequently employed in machine learning methods like clustering and K-nearest neighbours. The biggest absolute difference between the coordinates of two data points is used to determine the Chebyshev distance. It is less susceptible to outliers than Manhattan and Euclidean distances since it only takes into account the biggest difference between the coordinates, making it ideal for data with extreme values. It is especially helpful in applications like control theory where the maximum deviation is important.

$$D_{\text{Chebyshev}}(x, y) := \max_i(|x_i - y_i|).$$

Image 7. Chebyshev Distance Max

Chebyshev distance's resistance to outliers and capacity for handling data with extreme values are two of its primary features. It is helpful in applications like control theory where the biggest deviation is important. It is also a straightforward and effective distance measure in terms of calculation. Chebyshev distance does, however, have some restrictions. Its inability to account for the magnitudes of the coordinate disparities is one of its drawbacks. Additionally, it equalizes all of the data's features or dimensions, which may not be appropriate for all datasets.

Chebyshev distance is more appropriate for data with extreme values and is less sensitive to outliers than Euclidean distance. It may not be appropriate for all datasets and does not retain the actual distances between data points as well.

$$\lim_{p\to\infty}\left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{1/p}$$

Image 8. Chebyshev Distance infinite

In conclusion, for data with extreme values and applications where the highest deviation is crucial, Chebyshev distance is a reliable distance metric. Alternative distance metrics might be more appropriate for some machine learning tasks, and it might not be appropriate for all datasets.

H. *Canberra Distance*

Canberra distance is a typical distance metric in machine learning techniques like clustering and K-nearest neighbours. The distance between two points in Canberra is computed by dividing the total absolute difference in the coordinates of the two points by the total of their absolute values. It is better suitable for datasets with widely changing magnitudes because, unlike Euclidean and Manhattan distances, it considers the magnitudes of the discrepancies between coordinates. Applications for feature selection and data compression benefit the most from it.

The capacity to handle datasets of widely different magnitudes is one of Canberra distance's key features, which makes it particularly helpful in feature selection and data compression applications. It is a computationally effective distance metric as well. Canberra distance does, however, have some drawbacks. Its vulnerability to extremely small values, which might result in division by zero or numerical instability, is one of its limitations. Additionally, it ignores feature correlations, which in some circumstances could lead to less-than-optimal clustering solutions.

$$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n}\frac{|p_i - q_i|}{|p_i| + |q_i|}$$

Image 9. Canberra Distance

In conclusion, the Canberra distance is a metric for measuring distance that takes the magnitudes of coordinate discrepancies into consideration. It is particularly helpful in applications for feature selection and data compression. However, it is susceptible to very small values and ignores feature correlations. For some machine learning tasks, different distance measurements might be better suitable.

I. *Chi-Square Distance*

Chi-square distance is a popular distance metric in machine learning techniques like clustering and K-nearest neighbours. The sum of the squared differences between the coordinates of two data points divided by their sum is used to determine the chi-square distance. It is especially helpful in applications that compare probability distributions or histograms.

Chi-square distance is advantageous in applications like image processing and computer vision due to its ability to compare histograms or probability distributions. It also satisfies the triangle inequality and is a symmetric distance metric.

However, Chi-square distance also has some drawbacks. One is that it may not be suitable for datasets with negative values, as it requires non-negative data. Additionally, it may be sensitive to differences in the number of dimensions or bins in the histograms being compared.

$$X^2 = \frac{1}{2}\sum_{i=1}^{n}\frac{(x_i - y_i)^2}{(x_i + y_i)}$$

Image 10. Chi-Square Distance

In comparison to Euclidean distance, Chi-square distance is more appropriate for comparing histograms or probability distributions. However, it may not be suitable for all datasets, and alternative distance metrics may be more appropriate for certain machine learning tasks.

J. *Purity score*

Purity score is a measure of cluster quality used in k-means clustering. The degree to which a cluster exclusively includes one class label is indicated by the purity score. It is determined as the total number of data points divided by the sum of the greatest class frequency for each cluster. Purity score is easy to grasp and interpret and is extensively utilized in practice.

The ease of use and interpretability of purity score is one of its key benefits. It offers an easy-to-understand gauge of cluster quality that can be shared with stakeholders. It is also a quick and effective measure in terms of calculation.

However, there are certain restrictions on the purity score, though. The fact that it presumes that the ground truth labels are known which is not always the case in practice is one of its limitations. Furthermore, because of potential bias toward the dominant class, it might not be suitable for datasets with class imbalance.

In contrast to other cluster quality metrics, purity score is a straightforward and understandable indicator of cluster quality. Alternative measures, such as the adjusted Rand index or F1 score, may be more suitable for some machine learning tasks and may not be appropriate for all datasets.

$$Purity = \frac{1}{N}\sum_{i=1}^{k} max_j|c_i \cap t_j|$$

Image 11. Purity Score

N = number of objects (data points), k = number of clusters, ci is a cluster in C, and tj is the classification which has the max count for cluster ci.

In conclusion, the purity score is a straightforward and interpretable measure of cluster quality utilized in k-means clustering. For machine learning projects with special needs, other approaches should be taken into account because it has some limits.

III. IMPLEMENTATION

A. Dataset

Iris dataset is a popular choice to benchmark clustering and classification machine learning models. The reason for the same is that the data is simple, the data is denser in their range and most of all, it's perfectly balanced. There are in total 150 instances contain facts about 3 different flowers {Setosa, Versicolor, and Virginica} including the sepal length, sepal width, petal length, and petal width with all the three flowers having 50 entries of each.

Image 12. Iris

The dataset was first used in a 1936 study titled "The use of multiple measurements in taxonomic problems" by British statistician and biologist Ronald Fisher. Fisher demonstrated a linear discriminant analysis method to categorize the three iris species based on their morphological traits using the Iris dataset.

With one row for each iris flower and four columns for the four physical measures, the iris dataset is organized as a table. The species name for the iris flower in each row is also included. The dataset is frequently used as a dummy example for machine learning clustering, classification, and visualization tasks.

III.A.1 Data challenge

Iris dataset is overall a simple dataset but there is one thing to notice before getting started with model. The main challenge is cluster conflict, cluster conflict is when two clusters collide with each other. In our dataset, versicolor and virginica are colliding for all the features where as Setosa is spread apart from both the other class for all features.

This can result in conflict for class determination as K-means work by taking in consideration the majority class datapoints in the K closest set and assign the datapoint in question to the majority class. This nature can result in misclassification of some data points resulting in affecting the score of the model. This is also a good thing for this research as the data challenge will thoroughly test the model for all the noise we set and the observed score will be equally affected.

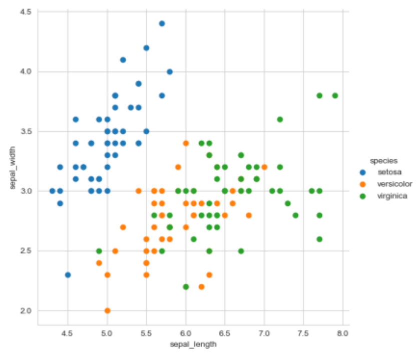

Image 13a.

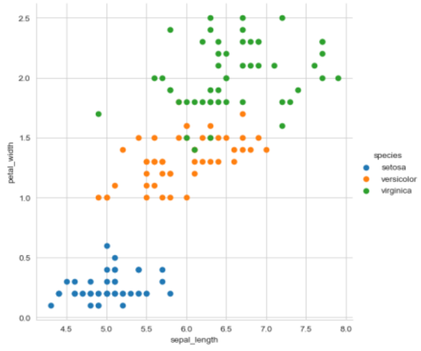

Image 13b.

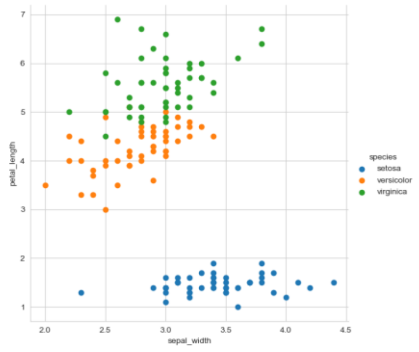

Image 13c.

In the above scatter plot, it can observe that the Setosa sits separate from versicolor and virginica by some margin across all the cross-feature scatter plots. Also, thing to notice is that versicolor and virginica collide among many features. However, the denser distribution is still separate, there is a thin line where the minimum of one flower feature is similar to maximum flower feature of another. The task here will be to calculate the distance and provide adequate importance to that distance so that the correct class datapoints are in majority. This will be a good test for all the distance measures.

B. Base lines and utilities

Before starting with the modelling, we need to establish some base line parameters so that the evaluation can be done with the noisy data. To set the baseline, we first used the elbow method to determine the perfect value for K which will be used throughout the experiment. Then using that K value, we will observe the purity score for default Kmeans algorithm provided by the sklearn library in python. Then plotting the clusters with centroids to see the class separation, mainly how well virginica and versicolor are separated.

III.B.1 Elbow method

Elbow method, as discussed, is used to find the optimum value of K for the Kmeans algorithm. we used the KElbowVisualizer() procedure provided in sklearn library as its much simpler to implement and the results are adaptable. In order to get the procedure to execute, one needs to first define the model and train the model on the train data using .fit() procedure. After executing the, the results are presented in the below figure.

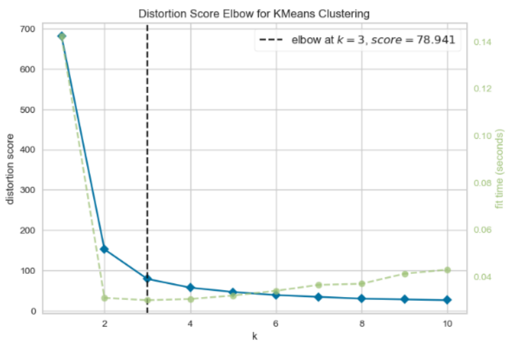

Image 14. Elbow method

The figure plotted by the procedure depicts that the optimum value of K is 3 and the received score is 78% for training data. The reason for the value to be 3 is that the distance slop at 3 starts to get constant, before 3 the slop is dropping at some high rate.

III.B.2 Baseline model score

The criteria for the baseline model is that the data has to be clean and original, no standardisation or any data pre-processing, no noise added, using the default distance measure provided by the sklearn and using the k value identified optimum by the elbow method. The idea here is so that we can get some baseline of how the default model works with the default distance measure so that we have come parameter to compare the experiment results with something other than itself. We executed the Kmeans using fit_predict() as it wont just train the data but also predict the clusters centroids for each class. After execution we calculated the purity score of the prediction which came out to 89.33 %. This score will be the baseline for our experiment.

III.B.3 Cluster centres

Sklearn has the procedure as model.cluster_centers_ which returns an array containing the value for all the clusters that will be used to plot the graph. After getting the clusters, we used the matplotlib library to plot the clusters using the .scatter() procedure. The resulting graph obtained is below.

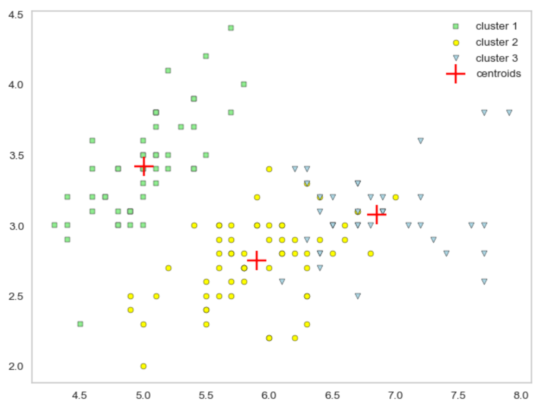

Image 15. clustering

In the graph, all the cluster centers are plotted as “+”, the different coloured points belong to the respective class. As expected, setose did not possessed any challenge to the model, the cluster center feels nice in the mean middle of the data. Versicolor and virginica, as expected, did conflict with each other but the model was able to predict the classes nicely, it can also be seen that few of the virginica are predicted as versicolor as there are more versicolor in the plot, but the data had them in equal ratio.

C. Adding noise

In the experiment, we’ll iteratively add noise to the data at the rate of 1% and will go till 25%, which means that we will be adding noise quiet a lot of time, to keep the whole program modular, we created a module or procedure that takes in the number of error we want to add and will return the train data and test data with defined numbers of outliers in the data. This makes the program clean and highly modular.

III.C.1 Noise criteria

Before creating the algorithm, we discussed what sort of noise to be added, and as described in the Introduction, the best sort of noise that can be added is outliers. We also don’t want the noise to be very extreme, so the criteria decided for the noise was that the noise value has to be <min feature value and >max feature value. This ensures that the noise will not get mixed with true data points and not too extremely separate that it becomes easily noticeable and ruins the model just from the start.

III.C.2 Input and output of the get_noisy_dataframe module.

The input to the module will be the rate of error or number of errors that will be added to each feature, which will be calculated by get_percent_values() which takes in the %age value and return the absolute number of error.

The output of the module will be the X data or the train data containing the 4 feature values for all the flowers and Y data or the test data which contains the label values of all the flowers.

III.C.3 Algorithm

The detailed pseudo code is defined below:

```
get_noisy_dataframe(error_rate)
  1. get data as iris
  2 for each feature, perform:
  3. get feature values in feature_column
  4. get max value in feature_column as MAX
  5. get min value in feature_column as MIN
  6. declare counter as xx
  7. while xx < error_rate, perform, else: go to 14.
  8. generate random value between 0,10
  9. if random_value<MIN or random_value>MAX, then continue,
       else: go to 8.
  10. get random row index as ran_row_ind
  11. assign random_value at feature_column[ran_row_ind]
  12. increment counter xx
  13. return to 7.
  14. replace column in iris dataset with feature_column
  15. split iris dataset in to X and Y
      16. return (X,y)
```

D. Evaluating noise to centroid

After generating the noisy data, we need to verify as if the noise is affecting the model at all. So, we again ran the baseline model with iteratively adding the noise and observing the result. After each run, we plotted the clusters with centroids to observe the impact.

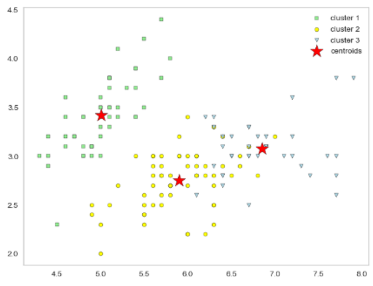

Image 16. 0% error

At 0 error per column, the results are as expected, similar to what we observed during the baseline.

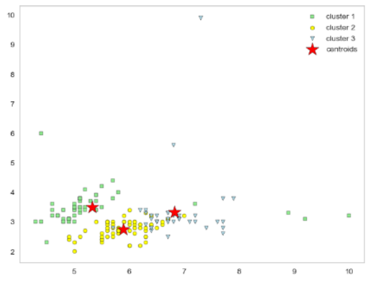

Image 17. 3 error

At 3 error per column, totalling at 12 error in total, the purity score dropped from 89% to 86%, the deduction isn't huge as the noise points added are far from the clusters, so the mean centroids where not affected much. The cluster centroids are still close to the center of the clusters.

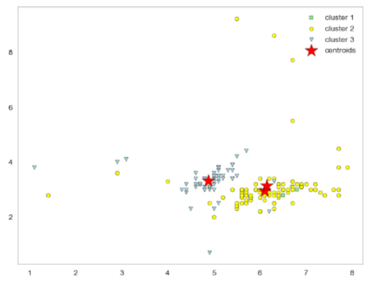

Image 18. 6 error

At 6 error per column, totalling 24 error in total, the purity score dropped from 86% to 65%, significant deduction. The reason here is that the noise points added moved the cluster center towards other cluster, resulting in misclassification of data points. And as expected, virginica and versicolor did conflicted with each other and most of the datapoints in one class are classified as others.

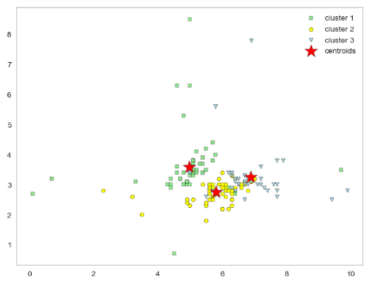

Image 19. 9 error

At 9 error per column, totalling 36 error in total, the purity score increased from 65% to 84%, significant increase. The reason for this unexpected change is that the noise is generated on random and in this run, the noise is not too far from the cluster, so the centroids where not shifted that far and resulting in conflicting with other clusters.

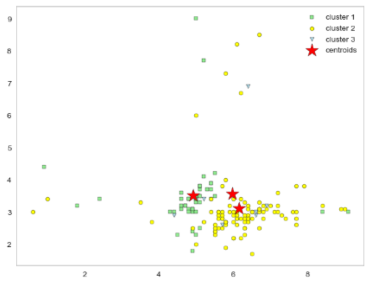

Image 20. 12 error

At 12 error per column, totalling 48 error in total, the purity score decreased from 84% to 63%, significant decrease. The reason here is similar to 6 error per column. The reason its not in line with 9 error per column is also same, the error generated is random, so this case was not as good as 9 error per column for the model.

III.D.1 Error conclusion

As we see the above results, the conclusion can be made that the model is very sensitive to the noise when Euclidian distance is used. In order to deal with the issue of random worst-case scenario, we need to make multiple runs for the same error range and then take the average of all those runs, that way the rare worst case or best case scenario will be dimmed down against all the average case scenarios.

E. Algorithm

The main module that is responsible to perform all the experiment, named *run()*, is on top of 3 other modules, first is the *get_percent_value()* that takes in the %age value and return the absolute error that need to be added, then we have *get_noisy_dataframe()* that takes in the error rate and return a dataframe with that many errors in it, and at last we have *pyRunKm()* that takes in the X data, the y data and the distance measure and returns purity score that the Kmeans model generated on the given data and the given distance measure. The run() module takes in two values, the min percentage error and the maximum percentage error, and it iteratively keeps adding noise at a constant rate till it reaches maximum percentage error, it returns a 3D array, one dimension for the number of run which will be 0 to 100, one dimension for each separate errors score, and the last dimension for the distance measure.

III.E.1 Pseudo code for *run()*

```
run(min_error_percent,max_error_percent)
  1. declare a list as error_list = []
  2 for each run_error_percentage in range (min_error_percent,
     max_error_percent), perform: 3-9
  3.get error_rate from get_percent_values(run_error_percent)
  4. get X,y from get_noisy_dataframe(error_rate)
  5. declare a temperary list as temp_list = []
  6. for each distance_measure, perform 7-8
  7. get purity from pyRunKm(X,y,value)
  8. append purity to temp_list
  9. append temp_list to error_list
  10. declare list for each distance
  11. append respective list from error_list to distance_list
  12. return error_list
```

IV. Results

The modules performed respectively slower than expected as total of 25 (total percent value) * 6 (total distance measures) * 100 (total runs performed) = 15,000 executions where done. After doing 100 runs for error rate ranging from 0% to 25% in each column, large set of purity scores where obtained. First, we put all the runs to a plot to check the variation in purity score based on random best-case and worst-case scenarios appeared due to randomly generated noise.

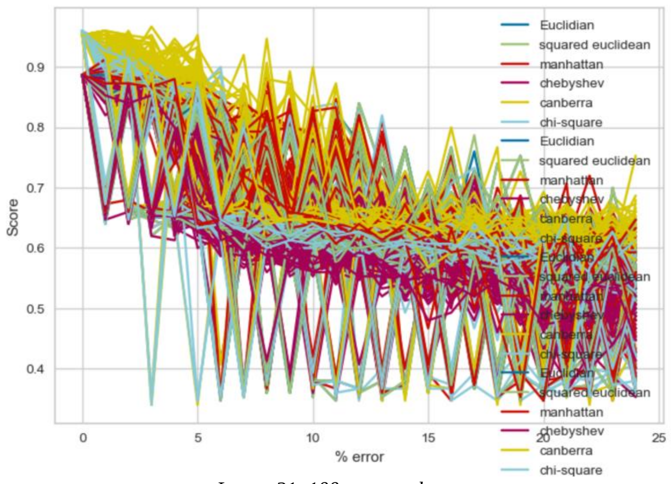

Image 21. 100 run overlap

As expected, a lot of worst-case and best-case scenarios had happened during the 2,500 (%values * total runs) runs. There are sudden down spikes that are returning back to up and there are lot of up spikes that are returning down. One thing to notice is that the hue of yellow is denser over all the other colour and is performing better then other distance measures, that distance measure is, Canberra distance measure.

This graph is not sufficient to draw strong conclusions as there are many best- and worst-case scenarios taking place. So, to get a meaningful graph, we took the average of all the 100 runs for all the distance measures overt the range of percentage error. The graph we obtained is followed.

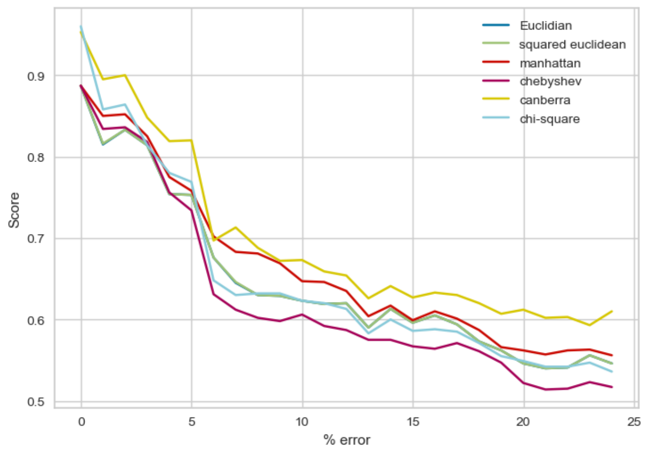

Image 22. 100 run average

This graph is of the average purity score obtained over 100 runs. Its clear that the Canberra distance measure is performing best over any error rate. Following we have the Manhattan distance measure performing better than other measures but still significantly poor than Canberra over the range of 5% to 25%. In the middle there is a mix between chi-square, squared Euclidian, and Euclidian. And the worst performing measure, Chebyshev distance measure. One interesting thing to notice is that the Euclidian and the squared Euclidian perform just the same over all the range. The detailed insights are as followed.

- Starting with the best performing, Canberra distance measure, which kept the score of 61%+ even at 25% error per column. The significant drop was observed at 5% to 6% error where the score dropped from 82% to 69%.
- The second-best performing measure was Manhattan distance measure, holding the score at 56%. The simplicity of this measure can be a base to use it over any other distance.
- Then we have Euclidian and squared Euclidian performing exactly the same over all the range of error. and as they are performing the same, squared Euclidian distance should be used as its less computationally costly.
- Then we have chi square distance measurements, it performed best for 0% error data for all the 100 runs but as there was 1% error, the score went from 97% to 86%, major score loss. Chi squared suffered the biggest loss going from 5% error to 6% error, which makes chi square the least robust distance measure and the best performing distance measure at 0%.
- At last we have Chebyshev distance measure, the least robust distance measure and the worst performing one.

V. Conclusion

We used the Iris dataset that contained 150 instances of 4 features and 3 classes having 50 instances for each class. The noise criteria we selected was to generate a random value that's between 0 to 10 and is not in the range of feature values. We did runs for 0% error per column to 25% error per column for all the 6 distance measures and averaged the purity score of 100 different runs. The results obtained after the experiment state that the Kmeans machine learning algorithm is not robust in handling the noisy data, just by adding 1% noise to each column dropped the score significantly. Over the period of iteratively adding the error, the score kept decreasing spikily until it was 5%, then the drop in score of marginal over each iteration. All the distance measures performed differently.

By observing all the insights, it can be concluded that the most robust distance measure for Kmeans for noisy data is Canberra distance measure and the least robust distance measure is Chebyshev distance measure. All the distance measurements suffered great score loss from 5% error to 6% error. All the distance measure gained some score going from 1% error to 2% error.

Going back to the question, is it a good idea for project managers to skip researching on outliers and reduce the project cost? Until the error is in minor range, i.e., 1% to 3%, it won't make a significant impact on model's performance, investing that time to experimenting with standardised data, min-max normalization, feature generation or any other data pre-processing task can give greater returns for the project.

VI. Future Work

Kmeans and Knn are widely used machine learning algorithms when it comes to clustering and classification. Kmeans perform very well under clean data but as the noise starts to get in the training data, nonlinear loss of score is observed, however, this experiment was done on un-processed data. And the distance measure used where unweighted. There are many experiments that can be conducted on similar cases. Some of the possible future experiments can include:

1. Varying K value: the elbow method provides the optimum K value for model; however it is possible that changing the K value respective to the noise can provide different results as more datapoints are taken into consideration and that can change the classification probability. So, to perform elbow method after every error introduction iteration.

2. Weighted distance measures: all the distance measures used where unweighted, so the importance to a neighbouring point just next to the datapoint in question has equal emphasis compared to the one that the farthest. Weighted distance can improve the chances of correct classification as it makes sense that two alike points will be closes to each other.

3. Considering different dataset: A different dataset with different characteristics can impose different result. By applying the same experiment on different dataset can be used to confirm the conclusions drawn from this experiment.

4. Run selection: we have done 100 runs on the same dataset and for many runs, we were able to see some extreme cases of sudden score gain or sudden score loss. This was because of the random value generated as noise, we can modify the program by checking the variance in score over all the error rate and decline the one that are having very high variance, this can improve the ratio of meaningful run we get at the end.