K Means Clustering on Cifar10 Dataset

Nandini Chinta
SUNY Buffalo, NY
nandinic@buffalo.edu

Abstract

This documentation describes how K Means Clustering is implemented on Cifar10 dataset. The dataset is loaded from Keras, and the model is trained on a pre-processed test data of 10,000 instances of images from Cifar10 which are Gray scaled first and then reshaped to 2D array to make the computations easier. The K Means Clustering is implemented on Cifar10 and validated using Silhouette Score from sklearn library and Dunn's Index from validclust library. This project scored Silhouette Score of 0.057 and Dunn's Index of 0.100

1 Loading and splitting the Dataset

Cifar10 is the dataset on which K Means Clustering is implemented on in this project. It contains 60,000 instances of images on total which splits into 50,000 and 10,000 instances of images for training and test data respectively. The dataset is loaded using Keras library which directly gives the data after splitting it into training and testing.

2 Data Pre-processing

Data Pre-processing clears out anomalies and outliers that can reduce our accuracy and make the predictions as error-free as possible. The most critical phase of any ML model contains steps that cleanse the data, normalize the data, and make it easier for further computations in the model training and validating.

2.1 Gray Scaling

Gray Scale does the major part in data pre-processing for K Means Clustering. Here, we convert the colour images into black and white so that computations become easier. This is done using cv2 library which provides a function to grayscale any coloured image. In Gray scaling we change the contrast of the image keeping black to the minimum and white to the maximum.

2.2 Data Normalization

Data Normalization makes all values in the dataset range same and minimum for easy math. Here, Cifar10 dataset contains RGB values ranging from 0-255. So, we normalize it to 0-1 by dividing each value with 255 so that every value range from 0 to 1.

2.3 Data Reshaping

Since each instance in the dataset Cifar10 contains the images of size 32*32*3 which is way more complex to compute, we reshape it to 2D array. In test data, we have the data size 10000*(32*32*3) which we convert into 10000*1024.

3 Model Building

K Means Clustering, by the name itself we can say we use mean of the data to do the clustering. This is an unsupervised learning model that groups the data into a cluster based on given features. The steps go as follows:

1. Select random data points as centroids.
2. For each data point in the dataset, calculate distance (here, we use Euclidean Distance) between the data point and all centroids.
3. Append the datapoint into the cluster with minimal distance.
4. Compute centroids for each cluster.
5. Repeat steps 2,3,4 until certain iterations or when the centroids don't change often.

3.1 Initial Centroids

Initially, centroids are selected at random. Here in this project, we used random function to select from the range of given number of clusters count. And then we selected the datapoint with index of randomly selected integer from random function.

3.2 Data Clustering

This includes data being assigned to a cluster. Distance between data point and the centroid is calculated using Euclidean distance using SciPy library. The data is assigned to the nearest cluster from the data point. Therefore, we also call this as k- nearest neighbours (KNN). The mean of each cluster after assigning all datapoints to the clusters is calculated. These become new centroids for the clusters. Again, Euclidean distance between datapoints and all centroids is calculated, and the data points are reassigned to their nearest cluster. The process is followed for given certain number of iterations.

$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b-a)^2}$$

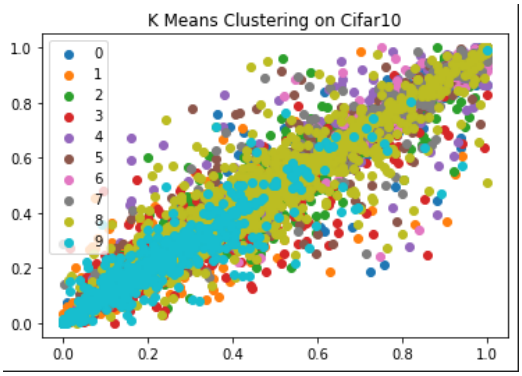


Figure 3.2.1: Clustering Plot

4 Validation

Validation in K Means Clustering checks for the quality of the cluster. Here we use two methods to validate the model. Silhouette Score and Dunn's Index of the clusters.

4.1 Silhouette Score

Silhouette Score of the clusters gives how likely the data point was assigned to the right cluster. Score ranges from -1 t 1, 1 being the best and -1 being the worst clustering. Validation is also done on the same data we used to train the model. Moreover, Silhouette Score can be computed only for the data that contains labels count between 2 and data size. It is computed using below formula.

Score = (b - a) / max (a, b)

Where, a = mean intra-cluster distance
b = mean nearest-cluster distance

For this project, the model obtained Silhouette Score of 0.057

4.2 Dunn's Index

Dunn's Index is a metric to validate how well the clustering is done. It indicates how well separated the clusters are. It uses inter cluster distance and the diameter of the cluster to compute the Dunn's Index. It indicates that the means of various clusters are distant enough. Higher the Dunn's Index higher the quality of clustering. The main drawback of Dunn's Index is, since it must compute distance between each datapoints, the computational cost increases with increase in data size. This project obtained Dunn's Index of 0.100.

$$DI_m = \frac{\min\limits_{1 \leqslant i < j \leqslant m} \delta(C_i, C_j)}{\max\limits_{1 \leqslant k \leqslant m} \Delta_k}$$

Where, C_i = i^{th} Cluster

C_j = j^{th} Cluster

$\delta(C_i, C_j)$ = Inter Cluster distance

Δ_k = Diameter of Cluster – k

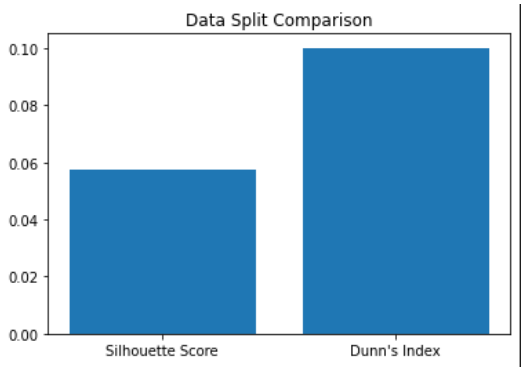


Figure 4.2.1: Silhouette Score & Dunn's Index

5 Auto-Encoders

Autoencoding is a feedforward algorithm in an unsupervised neural network that is built to learn compressing and encoding the data and then backpropagates to reconstruct the compressed encoded data back to its original form as close as possible. The data is first given to the encoder which compresses and encodes the data into a code which is then given to the decoder which decodes the encoded data using backpropagation to match closely to its original image. To build an encoder we need 3 functions: one to encode the data, one to decode the data and the other to measure the loss function.

Autoencoders are especially used for dimensionality reduction. These are data specific as they learn features specifically for given data, they can't be used for another dataset. For instance, autoencoders trained on MNIST dataset can't be used for Cifar-10 dataset. Also they are lossy and unsupervised due to its compression loss and trained with raw inputs.

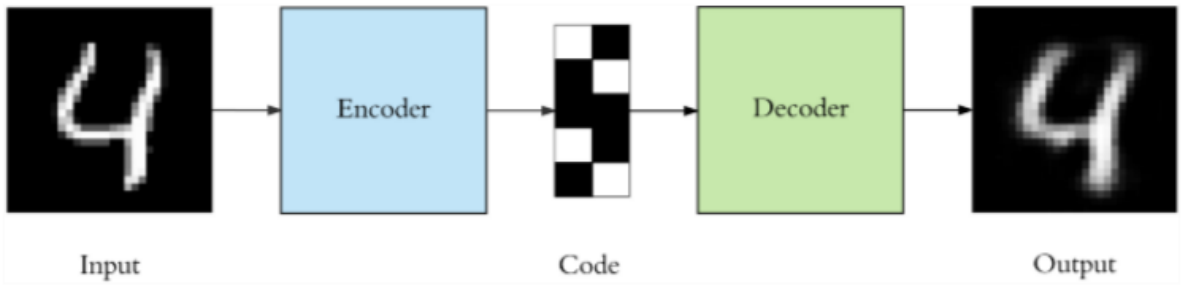


Fig 5.1: Structure of auto encoders

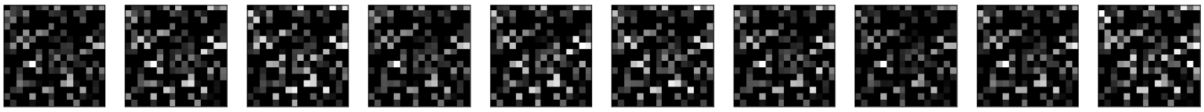
5.1 Implementation

The above grayscaled data of Cifar-10 dataset is used for implementing autoencoders. Here we have encoder layer and decoder layer. The very first layer, encoder layer, is fed with the input data. Here in autoencoders, data is considered as vectors. For instance, the grayscaled input data each image of size 32*32 is considered as a vector of 1024 size. This encoder layer compresses the data into given 256 dimensions which is the encoded data. This encoded data is then used in the next layer, decoder layer. The output of this layer is stored. All layers are dense layers. Activation functions like relu and sigmoid activation functions are used. A model that takes input images and provides our auto encoded outputs, is created using keras library. The model is fed on the training data. Using compile and fit functions from keras, autoencoder model is trained with 'adam' optimizer and 'binary crossentropy' loss functions. This loss function measures how much data is lost during the process of compression. The model is predicted using predict function in keras.

Below are the images of input, encoded and decoded data after implementing autoencoders on Cifar-10 dataset. Although, we try to backpropagate the encoded images while decoding, there will be a data loss due to which the below decoded images are not as clear as input images.

Input data:

Encoded data:



Decoded data:

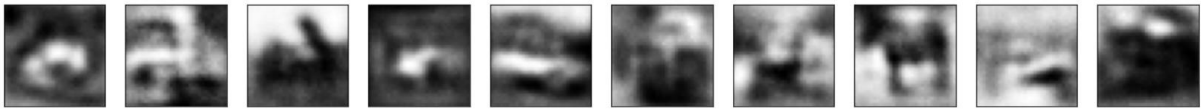


Fig 5.1.1: Original, encoded and decoded images

5.2 Model Testing Scores

Autoencoders are tested using Silhouette Scores. Here, with encoded_dimensions size 256, 20 epochs and 512 batch size, the autoencoder scored Silhouette Score of 0.104

5 Conclusion

In brief, this unsupervised learning algorithm classifies given data into various clusters based on similarity between the data and the cluster. However, K Means Clustering gives less accuracy when compared to other clustering techniques due to its assumption that clusters are spherical and evenly sized. In this project, K Means Clustering held Silhouette Score of 0.057 and Dunn's Index of 0.100 when centroids with these indices are taken randomly - [4673, 6274, 1690, 2511, 7456, 2225, 4004, 2919, 4008, 8296]. Applying autoencoders, Silhouette Score raised to 0.104

References

[1] https://www.analyticssteps.com/blogs/what-k-means-clustering-machine-learning
[2] https://en.wikipedia.org/wiki/Grayscale
[3] https://en.wikipedia.org/wiki/Dunn_index
[4] https://validclust.readthedocs.io/en/latest/validclust.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[6] https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
[7] https://www.jeremyjordan.me/autoencoders/
[8] https://pages.mtu.edu/~thavens/papers/ICPR_2008_Havens.pdf