Assignment 2 – Final

Sai Mahesh Pullagura
11/14/2021

Overview

The task of this project is to perform unsupervised learning using the dataset "Cifar 10". For part 1, the raw dataset is to be grouped into an optimal number of clusters using k-means algorithm. While for the second task, the k-means clustering has been performed on a representation generated by the auto-encoder method. The entire code has been implemented with the help of Python.

Dataset

The Cifar-10 dataset contains images of size 32*32 for a total of 1024 pixels. This pixel-value is an integer between 0 and 255. Overall, the given dataset has a training set of 50,000 examples, and a test set of 10,000 examples.

Data processing

- I have used the function "**keras.datasets.cifar10.load_data()**" to obtain the required data consisting of both Training and Testing data. The initial shape of the Test input data set is **(10000, 32, 32, 3)** and that of Train data set is (50000,32,32,3).
- Next, we convert the image (data point) in the given data set to **gray scaling** using the in-built module **"cvtColor(image, COLOR_BGR2GRAY)".** The updated shape of the Test input data set is **(10000, 32, 32)** and that of Train data set is **(50000,32,32).**
- Finally, the data set is normalized to arrange the input range between 0 and 1. Since the maximum possible pixel value for the given data set is 255, we normalize the input data using the operation **"X_Test/255.0".** This makes the final shape of the Test and Train input data as **(10000, 1024)** and **(50000, 1024)** respectively.

Part I -- K-Means Clustering

- The main idea behind the K-means algorithm is that given a dataset in D-dimension Euclidean space, k-means partitions the data into multiple **clusters** such that the data points that are close to a centroid say k1 will fall under the cluster k1.
- We initially begin by **randomly selecting k clusters labels**.
- Then based on the **Euclidian distance** between each single data point **p** with each of the **k** clusters, if the distance is minimum between the two, then **p** is assigned to **k1**.
- Furthermore, once all the data points gets assigned to any one of the random cluster that have been initially selected, we then find the mean of each cluster among its own data points and make the estimated mean value as the new centroid for that group.
- Note that we iterate through the above 2 steps repeatedly such that all the points are clustered optimally with minimal possible distance between the data point and its corresponding centroid.

- Once we undergo a sufficient number of iterations, we then determine the accuracy of our model by calculating the values of “**silhouette_score**” and “**Dunn_score**”.

- **Programmatically**, we first begin with initializing the **number of clusters to 10**.
- Then we randomly select 10 data points from the Test input data and make them the **cluster labels (centroids).** They are initialized using below steps:
  ```
  List_of_Random_Indices = np.random.choice(X_Test.shape[0], No_of_Clusters,
  replace = False)
  Cluster_Label = X_Test[List_of_Random_Indices,:]
  ```

- Then for each data point, we start calculating the Euclidian distance with each of the 10 centroid using below operation:
  ```
  a=scipy.spatial.distance.euclidean(X_Test[i],cluster_Label[j])
  ```

- We estimate the centroid to which a datapoint is the closest.
  ```
  Cluster_to_Dataset_Label[i] = np.argmin(a).
  ```

- We then determine the mean of all data points present in each current centroid and then make the mean calculated as the new centroid to that group.
  ```
  Updated_Label = np.mean(points_in_a_given_cluster, axis = 0)
  cluster_Label[c] = Updated_Label
  ```

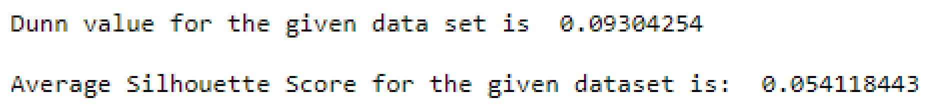
- The above three steps are performed iteratively (20 times in this case) so that all clusters become independent and contains data points that are in its given proximity

- Finally, to estimate the accuracy of the model, we calculate the values of **Dunn** and **Average silhouette score** as below.
  ```
  dist = pairwise_distances(X_Test)
  Dunn_Score=dunn(dist, Cluster_to_Dataset_Label)

  Average_Silhouette_Score=silhouette_score(X_Test, Cluster_to_Dataset_Label).
  ```

Results

The accuracy of K-means algorithm has been determined with below characteristics.



```
Dunn value for the given data set is  0.09304254

Average Silhouette Score for the given dataset is:  0.054118443
```

Part II – Auto-Encoder

- The main idea behind Auto encoder is to map an input **x** to an output **r** through an internal representation code **h**. Thus, the network consists of 2 parts: an encoder that transforms an input to a representational data (in hidden layer) and a decoder which aims to convert the obtained representational data back to original data.
- We first begin by compressing the given data, by assigning the optimal active number of neurons to a specific value which will be less than the actual data sets.
- Then, the input data is converted to an encoded format with respect to the set optimal active neurons as determined earlier to create a hidden layer. The nodes in the hidden layer now serves as an input to the next layer and this operation continues to perform till we obtain the desired code layer, which in-turn acts as an input layer to decode layer. The decode function aims to learn the limited features obtained from the code layer (due to sparse representation) and finally outputs the actual result.
- These hidden layers created are then passed as parameter to obtain the actual model of the auto-encoder along with defining the encoder and decoder models respectively. The auto-encoder model is used in training and predicting the given datasets.
- Furthermore, as part of the assignment, we use the predicted output from the encoder model and pass it to the k-means algorithm to cluster the data accordingly. The accuracy of the k-means model is then determined using the ASC value.
- **Programmatically**, we first set the value of **active_neurons** to be **128**. Secondly, we build the auto-encoder model by defining hidden layers between the input and output layer.
- The input encoder layer is first initialized with a shape of (1024,). This layer is now **sparsed**/ encoded from a **size of 1024 to 128** in order to reach the **code layer**. The output of the code layer is then fed as input to the **decoder layer** to get the final result of size (**1024,)**. Finally, the **auto-encoder** is now modelled using the input of encoder layer and output of decoder layer.

```
input_encoder = Input(shape=(1024,))
output_encoder=Dense(active_neurons, activation='linear')(input_encoder)
output_decoder = Dense(1024, activation="linear")(output_encoder)
autoencoder_final=Model(input_encoder,output_decoder,
name='autoencoder_final')
```

- Next, we define the encoder and decoder models as follows:

Encoder Model -- ***encoder_final = Model(input_encoder, output_encoder)***

Decoder Model --
```
encoded_input = Input(shape=(active_neurons,))
decoder_layer = autoencoder_final.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))
```

- Post creating both of the models, we begin **compilation** of the autoencoder model using **complie()** and **train** same model using **fit().**

- Once the model is trained, the result are **predicted** based on the encoder and decoder models defined earlier, using the in-built function **predict()**.

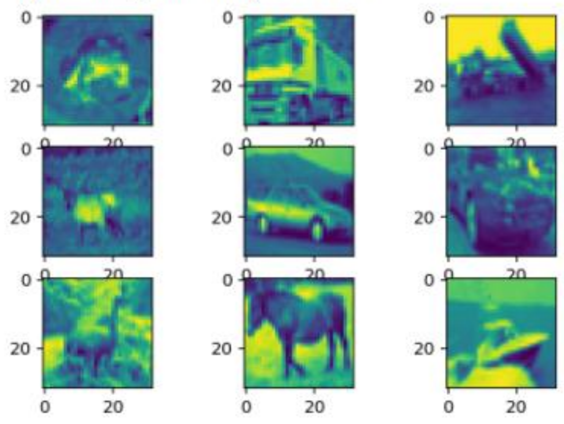
- The accuracy of the **predicted decoder model** can be compared with the help of **image plotting** between the original image data set and decoded data set. Note that the comparison has been depicted in the results section below.

- Finally, we perform the **k-means clustering** on the **predicted encoded data set** and evaluate the accuracy of the algorithm by determining **Average_Silhouette_Score.**

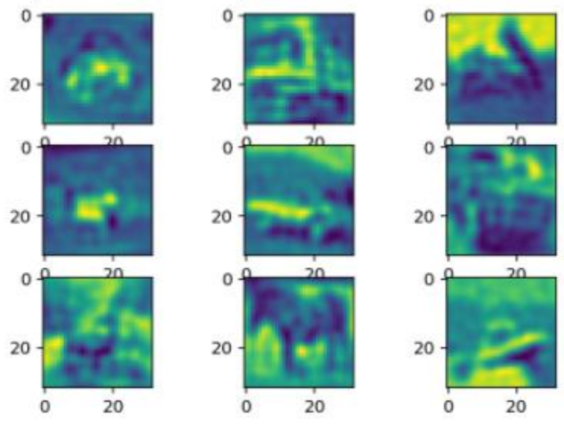
Results

Comparison between original image data set and predicted decoded data set.

```
Sample Original Image data set:
```



```
Sample Decoded Image data set:
```



Accuracy of K-Means Algorithm:

The efficiency of the K-means algorithm on the Decoded output is as follows.

```
Average Silhouette Score for the given dataset is:  0.024419282
```

References

- Lecture slides -- Chap9.1-Kmeans.pdf
- Chaitra cheluvaraju -- chaitracheluvaraju_assignment_sample_report.pdf
- Url -- http://localhost:8888/notebooks/CSE574%20ML/Professor%20Jupyter %20Notebook/K_Means_Clustering.ipynb
- Url -- https://piazza.com/class/ksuh71kldm036e?cid=157
- Url -- https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/.
- Url -- https://blog.keras.io/building-autoencoders-in-keras.html
- Lecture Slides -- 14.1 Autoencoders.pdf