**CPSC6114: Fundamentals of Machine Learning**

**Assignment 4: Non-parametric Algorithm for Image Clustering**

Name: Jemin Kachhadiya

Model

The goal of this assignment is to use an **unsupervised machine learning non-parametric algorithm** to **cluster a set of images.**

In contrast to **supervised machine learning**, where ML models use labeled data to train and test before making predictions, **unsupervised data is working with unlabeled data**. Its goal is to find certain similarities in data points to cluster them together thereby allowing to assign certain grouping (and in some cases eventually a label). Supervised learning can be costly as human beings have to review large sets of data and assign labels. Unsupervised learning models is an effective method to reduced this cost while eventually achieving the same results (**Hybrid models** – that use a mix of labeled and unlabeled data – is fairly popular as well and can produce better results than purely unsupervised models)

There are various non parametric algorithms exist today to cluster image data, including **agglomerative** clustering, density-based spatial clustering (**DBSCAN**), **HDBSCAN** (extension of DBSCAN), and **K-Means**.

### We will use K-Means algorithm for the purposes of this assignment. K-Means clustering is a method to divide n observations into k predefined non-overlapping clusters / sub-groups where each data point belongs to only one group. We are trying to divide our complete data into similar k-clusters. ‘Similar’ can have different meanings with different use cases. Similar can mean the same size, the same color in the image, same features, or anything you can think of. K-Means is a centroid-based algorithm where we assign a centroid to a cluster and the whole algorithm tries to minimize the sum of distances between the centroid of that cluster and the data points inside that cluster.

### The steps of the algorithm are:

### Select a value for the number of clusters k

### Select k random points from the data as a center

### Associate each data point with the nearest center calculating the Euclidean Distance.

### Calculate the centroid and mean of all data points in the cluster.

### Repeat 2, 3, 4 until stopping criteria.

### Stopping Criteria:

### the Maximum number of iterations is reached.

### Centroid of the newly formed cluster does not change.

### Data points remain in the same cluster.

### This algorithm aims at minimizing an Objective Function known as squared error function given by:

Text

Description automatically generated

### (Where k = number of cluster, n = number of observations/cases, xij = case i belong to cluster j, cj = centroid for cluster j)

(https://www.analyticsvidhya.com/blog/2021/06/k-means-clustering-and-transfer-learning-for-image-classification/)

Dataset Analysis

The dataset consists of **15,000 examples** (jpeg files) of 15 handwritten Chinese characters. Visually, we can tell that the sequence of characters is the same and repeats itself with every 16th character. Since all characters are handwritten, same characters may contain certain variation.

Model Building

For the purposes of this assignment we will be using **clustimage library** in Python. Clustimage is specifically designed for image clustering and evaluation. It takes care of image pre-processing, feature extraction and optimal cluster evaluation.

We start by importing the relevant library:

*from clustimage import Clustimage*

The data then is loaded from a file or from a directory (this step will be described below). Before we can begin the clustering exercise, the image data must be pre-processed and the data features must be extracted.

**Image pre-processing** in this case consists of three steps:

1. **Color scale**: conversion of the image into grayscale (2D) or color (3D)
2. **Scale:** normalize all pixel values between the min and mx range of 0 to 255
3. **Dim:** resize each image to make sure the number of features is the same

Once images are loaded and pre-processed, their **feature extraction** process can begin. **PCA (or Principal Component Analysis)** and **HOG (Histogram of Oriented Gradients)** are two popular methods.

**PCA** uses dimension reduction approach to minimize the number of features while retaining as mush (95%+) information about the object as possible. It extracts principal components where most of the variance is seen (e.g. six features such as “Name”, “Logged In To Site”, “Purchased on Site”, “Student”, “Faculty”, “Other” - > can be reduced to four features such as “Name”, “Active User”, “Academia”, “Other”)

**HOG** extracts features related to the direction and orientation of edges from image data. It simplifies the representation of the image to contain only the most important information such as the number of occurrences of gradient orientation in localized portion of an image.

1. The HOG descriptor focuses on the structure or the shape of an object. HOG features contain both edge and direction information.
2. The complete image is broken down into smaller regions (localized portions) and for each region, the gradient orientation is calculated.
3. Finally, the HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradient orientations of the pixel values, hence the name Histogram of Oriented Gradients.

(https://towardsdatascience.com/a-step-by-step-guide-for-clustering-images-4b45f9906128)

Not all applications are useful when using HOG features as it “only” provides the outline of the image. This can be a limitation with some of the images.

Note! Both **PCA-HOG** technique can be used whereby dimensions reduction is done first and HOG processing done on top of it.

For the purpose of this assignment we will use HOG technique:

*cl = Clustimage(method='hog')*

Once the feature extraction is set, we can use **fit\_transform** function of the clustimage library to **load, normalize, and cluster** our data:

*results = cl.fit\_transform("C://Users//denis//Downloads//archive//data//data\_Test4", cluster='kmeans', evaluate='silhouette', metric='euclidean', min\_clust=1, max\_clust=20, cluster\_space='low', black\_list=None)*

To evaluate the degree of success in clustering images, we will use a **Silhouette score.**  The silhouette value is a measure of how similar a sample is to its cluster (cohesion) compared to other clusters (separation). To produce the score:

1. We first measure (using Euclidian formula, for example) the distance between point A in cluster and all other points in the cluster. We then average them to produce **value a**. (This measures cohesion)
2. We also measure the distance between potin A and then points in other cluster(s), and average them. We then select the minimum value of these averages (in case of multiple clusters) produce **value b** (This measures separation)
3. We then use the formula **(b-a)/max(a,b)** to produce the Silhouette score

The score is bounded between **-1 and 1**. A score of 1 indicates the best result in a sense that a data point A is very compact within the cluster to which it was assigned, and far away from other clusters. The worst values is -1. Intuitively, as the distance between points within the cluster grows, and the distance between clusters shrinks, the score will become smaller (and eventually negative close to -1). On the other hand when all points in the clustered are grouped tightly together and are far away from other clusters, it would mean good clustering and the score will approach 1.

**An Elbow graph** depicts the relationship between the number of clusters chosen and the silhouette score. It will be one of our main tools in calibrating the model

Clustimage library has a helpful **clusteval** function that can produce both Elbow graph and depict Silhoute scored by cluster.

*cl.clusteval.plot()*

*cl.clusteval.scatter(cl.results['xycoord'])*

We will use additional plots to help us understand the performance of the model:

*# Unique images per cluster and scatterplot*

*cl.plot\_unique(img\_mean=False, show\_hog=True)*

*cl.scatter(dotsize=50, zoom=0.5, img\_mean=False)*

*# Unique images in each cluster*

*cl.plot(labels=3, show\_hog=True)*

Model Execution and Analysis

**First Run: Test**

Before using the model on the set of Chinese characters, we want to understand its basic effectiveness on a simpler set of images. We created four images of standard shapes and populated 84 jpeg files.



Running the model with the parameters outlined above, the model correctly recommended five clusters, with a silhouette score close to 1. We can now apply this model onto the set of Chinese characters

Chart, line chart

Description automatically generatedChart

Description automatically generatedChart, box and whisker chart

Description automatically generated

**Second Run: Two Chinese Symbols**

We now start using Chinese symbols from the library and chose **two symbols from 42 files** to see how the model behaves. Upon completion the model properly assigned two characters into two clusters. However, the silhouette scores are now lower (but still high) – around 0.7 - than during our test run.

Chart, line chart

Description automatically generated

Chart

Description automatically generated

Scatter chart

Description automatically generated with medium confidence

**Third Run: Full Fifteen Chinese Symbols**

We know that the set consists of samples of 15 characters. We now visually select **all 15 of them and use 66 files.**  The model recommends 19 clusters. The average silhouette score is around **0.45.** Clearly the model is not performing as well as before, allocating same characters to different clusters. Still. The silhouette score of 0.6 is high enough and the model is correct most of the times.

Chart, line chart

Description automatically generated

Chart

Description automatically generated

A picture containing square

Description automatically generated

**Third Run: Full Fifteen Chinese Symbols**

In the last run, we use **1,700 files** from the entire sample. The model’s performance continues to decrease, with 43 clusters recommended. Silhouette score is now a little over 0.4. The model put various different characters into the same cluster, as evident below with the images from Cluster 3. **At this point the reliability of this model (or at least its settings) is questionable**

**Chart, line chart

Description automatically generated**

**Chart, bar chart

Description automatically generated**

**Graphical user interface

Description automatically generated**

**A picture containing calendar

Description automatically generated**

Conclusion

It appears that the model functions well given either a small number of observations, or a more structured images without deviations (like we see present in handwritten images). The model produces high silhouette score in those cases. However, as the number of images increases the Silhouette score tends to **decrease fairly significantly from close to 1 to around 0.4**. We can see that clusters start to contain different characters that do not belong together. At this point, the reliability of the model becomes a subject of review. We can try to change model’s settings: use a different clustering algorithm (e.g. agglomerative), or try using PCS or PCA-HOG feature extraction. The current model still produces **reasonably good results** and depending on the problem it is trying to solve it may be “close enough”. This is a judgement call on the part of the data scientists and project owners.

**Full Python Code:**

*# Import library*

*from clustimage import Clustimage*

*# Initialize feature extration method*

*cl = Clustimage(method='hog')*

*# Load the data, specificy clustering algotirhm, evaluation technique, distance measure, min/max number of clusters*

*results = cl.fit\_transform("C://Users//denis//Downloads//archive//data//data\_Test4", cluster='kmeans', evaluate='silhouette', metric='euclidean', min\_clust=1, max\_clust=20, cluster\_space='low', black\_list=None)*

*# Silhouette plots*

*cl.clusteval.plot()*

*cl.clusteval.scatter(cl.results['xycoord'])*

*# Unique images in clusters and scatterlot of datapoints in clusters*

*cl.plot\_unique(img\_mean=False, show\_hog=True)*

*cl.scatter(dotsize=50, zoom=0.5, img\_mean=False)*

*# Plot unique image on top of each cluster*

*cl.plot(labels=3, show\_hog=True)*