**Machine Learning from Data – IDC**

**HW7 – K-means & Principal Component Analysis (PCA)**

**\*\*\* This assignment can be submitted in pairs.**

In this assignment you will implement the K-means algorithm and use it to transform an image to a quantized version thereof. In the second part you will use a pre-implemented PCA class in order to compress data, and determine the difference between the original data and the decompressed data (the decompressed data is the compressed data transformed back to the original space).

**K-means:**

Color image quantization is the process of transforming an image so that it can be represented with a small subset of colors; a subset smaller than the image's original palette size. Image quantization is a form of lossy data compression since the number of bits used for the palette is reduced. During this compression some data is lost and therefore we must use a careful algorithm that will minimize the perceived difference between the original image and its quantized counterpart. Other than compression, image quantization plays an important role in image recognition algorithms.

When representing images in true color, each pixel is represented by 3 RGB components and each one of the 3 RGB components is represented by 8 bits and can take a value from 0 to 255. This means that in true color representation one can use Million different colors. In this exercise we will perform image quantization from images given in true color into 256, 16 or even less colors.

There are many algorithms for quantization. You are going to use K-means in color spaces to do this. K-means assumes that we know the number of clusters to be found (denoted by k), and the algorithm progresses until the means of the clusters are found (or approximated).

In k-means the color assigned to each pixel is the color of the mean (centroid) of the cluster. This is a point in the RGB space.

The downside of k-means is that starting with different cluster center locations can lead to different local optima!

To summarize, every pixel in the image is a point in the three dimensional RGB space. We want to cluster these points into k clusters. We then replace each pixel by the centroid of the cluster to which it was assigned.

**PCA:**

You will also use PCA to do data compression. PCA is a method to reduce the dimensionality of your data by finding the rotation of the data that best de-correlates them. The new axes are called Principal Components (PCs). You will reduce the data dimensionality by creating a projection matrix and using only *j* of the PCs so we are left with *j* dimensions. We denote A as the PCA matrix (as shown in class) and as the matrix using only the j PCs corresponding to the highest eigenvalues. Then the new ‘compacted’ (lower) j-dimensional vector data is where is the average vector of all the features. These vectors will have 0 in all dimensions from j+1 to n. However, to get back the instance in the original axes you need to multiply back by the inverse of the original matrix . A ‘transformed’ instance is an instance that was projected with PCA to a lower dimensional subspace and then embedded back in the original space with the inverse process. You will try this process using different numbers of principal components. For each such dimension you will measure how far, by Euclidean distance, the transformed instances are from the original instances. This will give you a measure of how much information is lost when using only j PCs.

In order to do the above you need to first install WEKA:

1. See instructions in HW1.

Prepare your Eclipse project:

1. Create a project in Eclipse called HomeWork7.
2. Create a package called HomeWork7.
3. Move the KMeans.java, PrincipalComponents.java and MainHW7.java that you downloaded from the Moodle into this package.
4. Add WEKA to the project (see instructions in HW1).
5. In the same way add the mtj.jar file (from the zip folder that you downloaded) to the external JAR of the project – you will need this jar for the PCA part.

**K-means section:**

1. Copy the messi.jpg image file to your Project directory (so that your java program can access these files without an absolute path).
2. In the main method use the instructions below to open messi.jpg and covert it to an instances object.
3. For a fixed value of k, run the K-Means algorithm on this instances object (from 2), quantize it (see below), convert the quantized instances object back to an image (using methods provided) and save the resulting image.
4. Do this for multiple values (number of clusters). Namely all
5. In KMeans.java you should provide your own implementation of the K-Means algorithm. As part of the K-Means algorithm you should implement the methods listed below.
6. For k=5 also provide a graphical representation of the total error as a function of the iteration number.

For this section, the following methods are mandatory methods, but you can add additional methods:

1. void buildClusterModel:
   1. Input: Instances object

This method is building the KMeans object. It should initialize centroids (by calling initializeCentroids) and run the K-Means algorithm (which means to call findKMeansCentroids methods).

1. void initializeCentroids:
   1. Input: Instances

Initialize the centroids by selecting k random instances from the training set and setting the centroids to be those instances.

1. void findKMeansCentroids:
   1. Input: Instances

Should find and store the centroids according to the KMeans algorithm.

Your stopping condition for when to stop iterating can be either when the centroids have not moved much from their previous location, the cost function did not change much, or you have reached a preset number of iterations. In this assignment we will only use the preset number option. A good preset number of iterations is 40. Use one or any combination of these methods to determine when to stop iterating.

1. double calcSquaredDistanceFromCentroid:
   1. Input: 2 Instance objects – one is an instance from the dataset and one is a centroid (if you're using different data structure for the centroid, feel free to change the input).
   2. Output: should calculate the squared distance between the input instance and the input centroid.
2. int findClosestCentroid:
   1. Input: Instance
   2. Output: the index of the closest centroid to the input instance
3. Instances quantize
   1. Input: Instances
   2. Output: should replace every instance in Instances by the centroid to which it is assigned (closest centroid) and return the new Instances object.
4. double calcAvgWSSSE
   1. Input: Instances
   2. Output: should be the average within set sum of squared errors. That is it should calculate the average squared distance of every instance from the centroid to which it is assigned. This is Tr(Sc) from class, divided by the number of instances. Return the double value of the WSSSE.

Additional points:

Use the methods provided in MainHW7 to read \ write images and to convert an image into a WEKA Instances object and vice versa.

In order to open an image file in Java and create a BufferedImage object use:

BufferedImage image = ImageIO.read(new File("myImage.jpg"));

In order to save an BufferedImage object as a jpg in Java use:

File outputfile = new File("output.jpg");

ImageIO.write(out, "jpg", outputfile);

\* Where out is a BufferedImage object.

In order to convert a BufferedImage (which is the object that store the read image) to Instances use convertImgToInstance.

In order to convert a WEKA instances object back into a Java BufferedImage object use convertInstancesToImg.

\* When you use the method to convert an image into an Instances object what it does is take every pixel (x,y coordinate) of the image and turn it into an instance where the features are the R,G,B values of that pixel. The number of instances will be equal to the number of pixels, image.getHeight() \* image.getWidth().

**PCA section:**

1. In your main method, load the libras.txt data set.
2. For each number of principal components do the following:
   1. Run PCA on the instances and transform them back to the original space using principal components.
   2. Measure the average Euclidean distance of the new data set from the original data set and print this average distance to the console.
   3. Loop through (notice that the libras data set has 91 features initially, so using 91 principal components should be the exact recovery of the data)

\* An explanation of how to use the PCA algorithm is provided below (make sure to use the PCA code provided in the zip and not Weka’s PCA).

1. Copy all of the average distance outputs from the console and paste them into an excel spread sheet and make a scatter plot out of them. That is you should make a scatter plot where the x-axis is the number of principal components used and the y-axis is the average distance of the transformed data from the original data.
2. Call the excel file 'hw7' and save it.

For this section, the following methods are mandatory methods and you can't change their signatures, but you can add additional methods:

1. static double calcAvgDistance: Calculates the average Euclidean distance between the original data set and the transformed data set.
   1. Input: 2 Instances objects (the original and the transformed).
   2. Output: The average distance between the instances.

Implement this method inside your main class MainHW7. This method should iterate over all instances in the transformed and original set and for each corresponding pair of instances, it should measure the Euclidean distance between them and then average over the number of instances.

In order to run PCA on a set of Instances you need to create a PrincipalComponents object using the java class that is provided in the zip file and set the number of principal components. You also need to tell the PCA object whether you do or do not want to transform the data back to the original space (to run the inverse projection). In summary:

PrincipalComponents pca = new PrincipalComponents();

pca.setNumPrinComponents(i);

pca.setTransformBackToOriginal(true);

pca.buildEvaluator(data);

Instances transformedData = pca.transformedData(data);

double dist = calcAvgDistance(data, transformedData);

In this example we created a new PrincipalComponents object, we set the number of principal components to an int called i, we told the PrincipalComponents object that we do want to transform the data back to the original space, we ran the build method, and then we transformed the data. After that we calculated the average Euclidean distance between the two data sets.

You should hand in a KMeans.java, MainHW7.java and the hw7 excel file filled with the PCA results. All of these files should be placed in a hw\_7\_##id1##\_##id2##.zip folder with the id of both members of the team.

\*\*\* Submitting in groups on Moodle does not work. Please only submit one zip folder per pair