Assignment 6

CS-UH-2218: Algorithmic Foundations of Data Science

Assignments are to be submitted in groups of two or three*. Upload the solutions on NYU classes as one PDF file for theoretical assignments and separate source code files for each programming assignment. Submit only one copy per group. Clearly mention the participant names on each file you submit.*

Problem 1 (10 points)**.**
Download the file `kosarak.dat` from `http://fimi.ua.ac.be/data/` which contains the click-stream data from a hungarian online news portal. Each line represents a user and each number in the line represents a page visited by the user. Our goal is to identify the most popular pages. It is known that the average number of pages visited by a user is less than 10.

1. Identify, using the Misra-Gries algorithm, all pages that are viewed by more than 10% of the users. None of the pages viewed by less than 6% of the users should be identified.

2. Repeat the above using the Count Min Sketch algorithm. The error probability δ should be less than 2%. Try to minimize the number of counters used. Use *conservative updates* which is the kind of updates suggested in Assignment 5, Problem 2. You can use the 2-universal hash functions described in Assignment 5, Problem 3.

Problem 2 (10 points)**.**
Prove that given a set of n data points in one dimension (i.e., n real numbers) and a number k, the optimal k-means centers can be computed in time $O(kn^2)$.

Hint: Use dynamic programming. Recall that the goal is to minimize the sum of squared distances of the points to their nearest center. Create a dynamic programming table of size $n \times k$. Store in the $(i, j)^{th}$ entry of the table, the optimal value for the problem on the first i points using j centers. Prove that the $(i, j)^{th}$ entry of the table can be computed in linear time from the lexicographically smaller entries.

Problem 3 (10 points)**.**
In this exercise, we will use k-means clustering to compress an image. When an image is stored in the PNG format, for each pixel, the red, blue and green components are stored as 8 bit numbers each. Thus, we need 24 bits per pixel [1].

The idea for compressing is the following. We think of each pixel with components (r, g, b) as a point in three dimensions. Thus there are as many points as there are pixels in the image. Then, using k-means clustering, we compute k cluster centers. Each cluster center is a point in three dimensions and thus represents a color. If a point p belongs to a cluster with center

[1] We are assuming RGB format here. In the RGBA format, there is a fourth component called "alpha" which denotes transparency. The RGBA format requires 32 bits per pixel.

c, we will change the color of the pixel corresponding to p to the color represented by c. We expect the error caused by this change to be small.

Let us take $k = 64$ so that each cluster number is encoded by 6 bits. Thus, we will only use 64 distinct colors corresponding to the 64 cluster centers. Each pixel can now store 6 bits of information indicating which of the 64 colors it uses. Thus, apart from the overhead of storing the 64 colors using 64×3 bytes, we only use 6 bits per pixel, leading to a reduction in space by a factor of about 4.

1. Use the above idea to convert the image `Neuschwanstein_small.png` into an image that uses only 64 distinct colors.

2. If you try to use the same idea for a larger image like `Neuschwanstein_large.png`, you will find that k-Means takes a long time since the image has over a million pixels. Use `MiniBatchKMeans` instead of `KMeans` from Scikit-learn. Skim this short paper to learn how MiniBatchKMeans works.

You can start with the code below which shows how to convert an image into a numpy array and how to display such an array. You should also look at the sample code for k-Means available on NYU classes.

```
import matplotlib.image as mpimg

img=mpimg.imread('Neuschwanstein_small.png')
img  = img[:,:,:3] # remove the fourth coordinate alpha
plt.imshow(img)
print(img.shape)
```

To use `MiniBatchKMeans` instead of `KMeans`, you just need to import it and provide an additional parameter called `batch_size`. For instance, you can use code like this:

```
from sklearn.cluster import MiniBatchKMeans

mbkmeans = MiniBatchKMeans(n_clusters = 64, batch_size=10000)
mbclusters = mbkmeans.fit_predict(data) # some data
mbcenters = mbkmeans.cluster_centers_
```