Final Exam - MAT 180

Aditya Mittal

Due date: June 12, 2025

Instructions: Please submit the final exam (including your code) as a PDF file via Canvas. Use of ChatGPT, Gemini, Claude, or similar tools is **not** allowed. Clearly state all your work.

Problem 1 [40 points]

This problem is about differentially private clustering. The goal is to develop a differentially private version of the classical k-means (or Lloyd's algorithm), an iterative clustering algorithm.

Download the test data `kmeansexample.asc` from the course webpage. This file contains 400 points $\{x_k\}_{k=1}^{400} \in [0,1] \times [0,1]$, belonging to 4 clusters (points $x_1, \ldots, x_{100}$ form cluster 1, ..., points $x_{301}, \ldots, x_{400}$ form cluster 4). We want to study how well a differentially private k-means algorithm can predict these clusters as we change the privacy budget ε. We will use a standard (naive) k-means algorithm with random initialization as a baseline. (Wikipedia has a clear description of kmeans if you are not familiar with it.) Since we are using random initialization and the kmeans objective function is non-convex you may get different results in different runs even for the non-private version (think of how to handle this).

1. Derive an ε-differentially private version of k-means. Let us call it DP-kmeans. Give a pseudocode description of DP-kmeans. Explain how you calculate the sensitivity and your resulting choice of ε for the algorithm. At which step(s) in the iteration do you need to protect privacy?

2. Test your DP-kmeans algorithm numerically with the dataset `kmeansexample.asc`. The result should be a graph which shows how the clustering accuracy changes as you increase ε. (Note that even non-private k-means may not get 100% clustering accuracy).

Solution to (1):

Problem setup: we want to find k clusters in our dataset using an iterative clustering algorithm called Lloyd's algorithm.

Input: A dataset $\mathbf{x} = (x_1, x_2, \ldots, x_n) \subset \mathbb{R}^2$ as defined in the `kmeansexample.asc` dataset above.

Goal: Find k clusters in $\mathbb{R}^2$ that minimize the k-means cost. The k-means cost is defined using the squared ℓ_2 norm as:

$$\text{Cost} = \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - c_j\|_2^2,$$

where C_j denotes the set of points in the j-th cluster and c_j is the center of that cluster. Note that this cost function is not convex, and the iterative algorithm runs for T iterations. In each iteration, the new cluster center is updated as the average of all points assigned to that cluster.[1]

I will first show the non-private pseudocode of Lloyd's algorithm to solve the k-means clustering problem. This will introduce relevant notation.

Algorithm: Non-Private K-Means (Lloyd's Algorithm)

Require: Dataset X, number of clusters k, number of iterations T

Ensure: Final cluster centers $c_1, c_2, ..., c_k$

1: Initialize k cluster centers $\{c_1, c_2, \ldots, c_k\}$ by randomly sampling data points from X

2: **for** $t = 1$ to T iterations: **do**

3: Assign each point to the nearest cluster center:

$$k_i = \arg\min_{j \in \{1,\ldots,k\}} \|x_i - c_j\|_2^2, \quad \forall i = 1, \ldots, n.$$

4: **for** $j = 1$ to k clusters: **do**

5: Create jth cluster set of all points assigned $k_i = j$:

$$K_j = \{i : k_i = j\}$$

6: Compute cluster size:

$$n_j = |K_j|$$

7: Compute cluster sum:

$$a_j = \sum_{i \in K_j} x_i$$

8: Update cluster center:

$$c_j = \begin{cases} \frac{a_j}{n_j} & \text{if cluster size } n_j \geq 1 \\ \text{RandomSample}(X) & \text{otherwise} \end{cases}$$

9: **end for**

10: **end for**

11: **return** $\{c_1, c_2, \ldots, c_k\}$

[1] The algorithm can terminate in fewer than T iterations if the change in within-cluster error is smaller than a pre-defined tolerance level, such as $\text{tol} = 10^{-5}$.

The non-private Lloyd's algorithm above minimizes the total within-cluster sum of squares for all cluster. To make this ε-differentially private, we add independent Laplace($\frac{\Delta}{\varepsilon}$) noise to both the cluster counts and the cluster sums, where Δ is the global sensitivity. Analysis is shown after the algorithm.

Algorithm: ε-DP K-Means

Require: Dataset X, number of clusters k, number of iterations T, privacy parameter $\varepsilon > 0$
Ensure: Final cluster centers $c_1, c_2, ..., c_k$

1: Set $\varepsilon' = \frac{\varepsilon}{2T}$
2: Initialize k cluster centers $\{c_1, c_2, \ldots, c_k\}$ by randomly sampling data points from X
3: **for** $t = 1$ to T iterations: **do**
4: Assign each point to the nearest cluster center:

$$k_i = \underset{j \in \{1,\ldots,k\}}{\arg\min} \; \|x_i - c_j\|_2^2, \quad \forall i = 1, \ldots, n.$$

5: **for** $j = 1$ to k clusters: **do**
6: Create jth cluster set of all points assigned $k_i = j$:

$$K_j = \{i : k_i = j\}$$

7: Compute cluster size:

$$n_j = |K_j| \quad \text{(global sensitivity } \Delta = 1)$$

8: Compute cluster sum:

$$a_j = \sum_{i \in K_j} x_i \quad \text{(global sensitivity } \Delta = 1)$$

9: Add noise to cluster size:

$$\hat{n}_j = n_j + \text{Lap}\left(\frac{\Delta}{\varepsilon'}\right)$$

10: Add noise to cluster sum:

$$\hat{a}_j = a_j + (Y_1', ..., Y_d'), \sim \text{Lap}\left(\frac{\Delta}{\varepsilon'}\right) \text{ i.i.d.}$$

11: Update cluster center:

$$c_j = \begin{cases} \frac{\hat{a}_j}{\hat{n}_j} & \text{if } n_j \geq 1 \\ \text{RandomSample}(X) & \text{otherwise} \end{cases}$$

12: **end for**
13: **end for**
14: **return** $\{c_1, c_2, \ldots, c_k\}$

The algorithm described above is ε-differentially private. We add noise to both the cluster sizes and the cluster sums using the Laplace mechanism. The global sensitivity of this problem is 1.

ε-Differential Privacy Analysis We can show the proof of ε-differentially privacy by applying the Composition and Post-Processing theorem.
For each iteration T, we assign a privacy budget of $\varepsilon' = \frac{\varepsilon}{T}$ and split it evenly between:

- cluster sizes noise ($\frac{\varepsilon}{2T}$)
- cluster sum noise ($\frac{\varepsilon}{2T}$)

By composition, the total privacy cost per iteration is:

$$\frac{\varepsilon}{2T} + \frac{\varepsilon}{2T} = \frac{\varepsilon}{T}$$

over T iterations, the total private cost is:

$$T * \frac{\varepsilon}{2T} = \varepsilon$$

The final cluster centers $c_j = \frac{\hat{a}_j}{\hat{n}_j}$ are calculated using the noisy sums divided by the noisy counts. By the post-processing property, applying any function to the output of a private mechanism does not change its privacy loss. Thus, this step does not use any additional privacy budget. Note that we do not add noise directly to the cluster centers directly because it is hard to determine a global sensitivity for the ratio $\frac{\hat{a}_j}{\hat{n}_j}$ based on the effect of one person being removed/added.

Sensitivity Analysis Let $\mathbf{x}$ and $\mathbf{x}'$ be two neighboring datasets that differ by the removing one record, with no replacement.

- **Cluster Counts:** The vector of cluster sizes $(n_1, \ldots, n_k)$ create a histogram of cluster labels. Removing one record decreases exactly one cluster count by 1, so the sensitivity of the counts vector is 1. Add Laplace noise with scale:

$$Y_j \sim \text{Lap}\left(\frac{1}{\varepsilon'}\right) = \text{Lap}\left(\frac{2T}{\varepsilon}\right)$$

- **Cluster Sums:** Each cluster sum is defined as

$$a_j = \sum_{i \in K_j} x_i.$$

  Removing one data point affects only one cluster sum by at most 1 in each coordinate, so the sensitivity of each coordinate of a_j is 1. Add independent Laplace noise to each coordinate with scale:

$$Z_j \sim \text{Lap}\left(\frac{1}{\varepsilon'}\right) = \text{Lap}\left(\frac{2T}{\varepsilon}\right)$$

Thus, the entire algorithm is ε-differentially private.

Solution to (2):

Let us consider an example using the k-means dataset. We begin by looking at the raw data. All code is attached below after the written results.

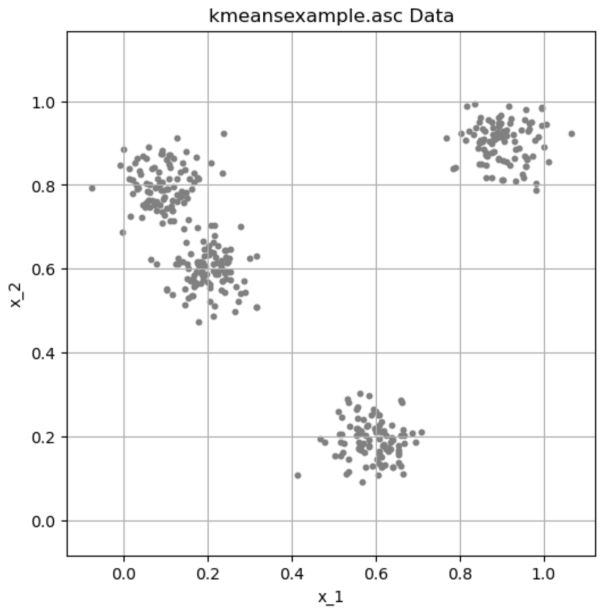

Figure 1: Raw Data

We know from the given info and plot above that there are 4 clusters. I first show results of the non-private algorithm. I measure accuracy using two metrics: within-cluster sum of squares (WCSS) to measure cluster tightness, and the percent of correctly assigned labels. I fix the number of iterations to $T = 5$. Below is an example run where the initial cluster centers are randomly selected from the data points to show how the centroids converge over multiple iterations.

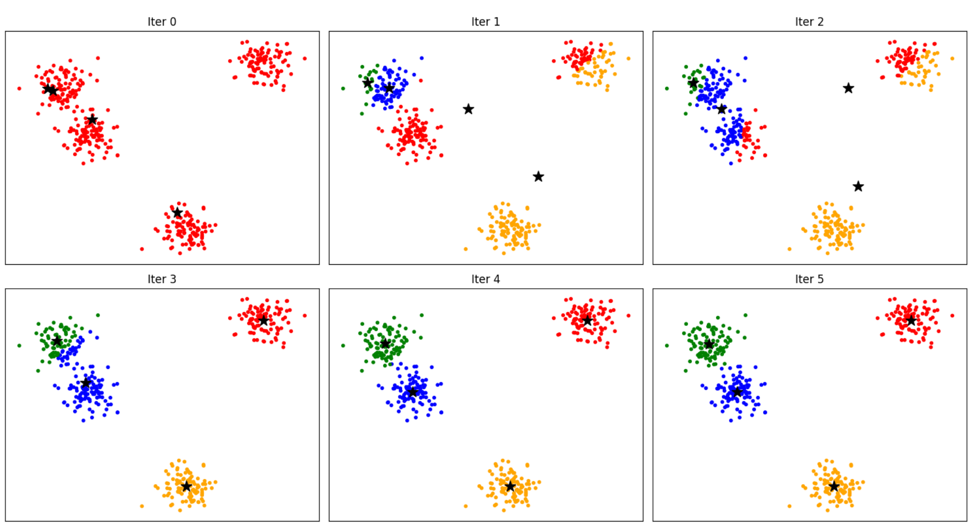

Figure 2: Cluster results with non-private K-Means

The within-cluster sum of squares for this run is 2.03, with a corresponding accuracy of 99.5%. However, due to randomness in the initial cluster assignments, these results can vary. To address this issue, I compute the average performance over 100 runs: the mean WCSS is 4.53, and the mean accuracy is 86%.

Let's next look at the private version. First, I show the result of a single run with privacy parameter $\varepsilon = 5$ and $T = 5$ iterations:

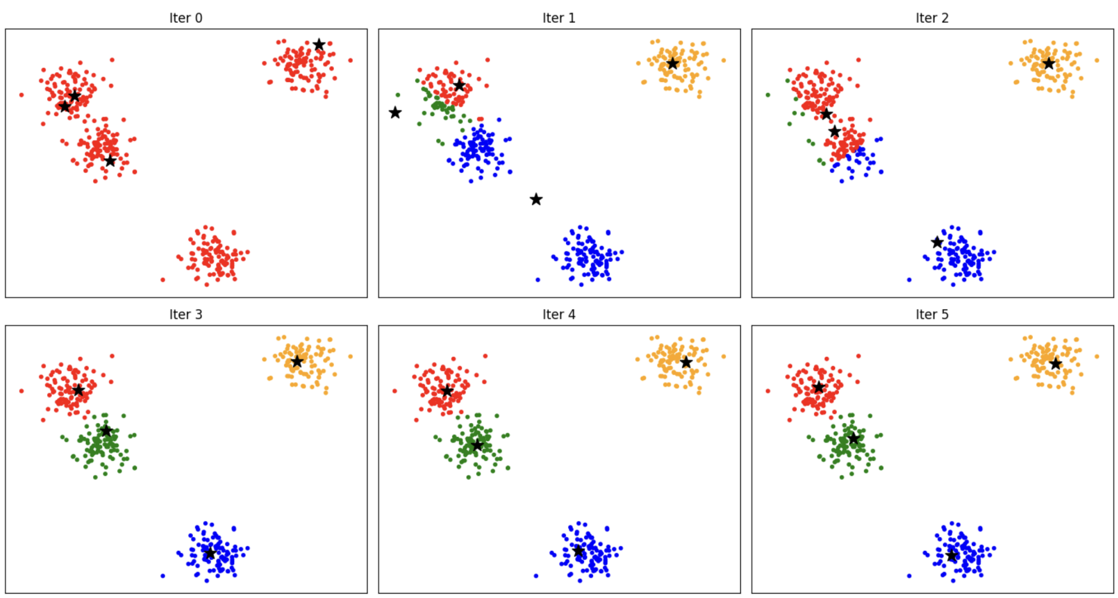

Figure 3: Cluster results with private ε-DP K-Means. $\varepsilon = 5$, $T = 5$

In this example, the final cluster centers are slightly perturbed due to the added noise. The WCSS is 3.231, and the clustering accuracy is 97.75%. Both the choice of ε and T affect the amount of noise added during the algorithm.

To understand the impact of the privacy budget, I will compute WCSS and accuracy across 100 runs with a fixed number of iterations $T = 5$ and varying values of ε.

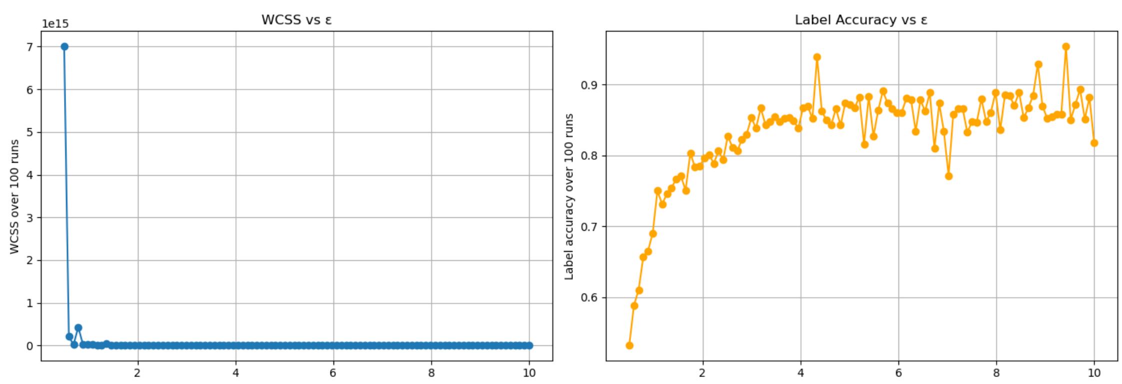

Figure 4: ε-DP accuracy results with varying ε ($T = 5$ over 100 runs)

We see that label accuracy improves significantly as ε increases. For small values such as

$\varepsilon = 0.5$, the percent of correctly labeled points is around 30%. Accuracy increases quickly with larger ε and appears to stabilize between 80% and 90% for higher values and matches the performance of the non-private algorithm. This shows that the amount of noise added has a substantial impact on accuracy. The results of WCSS vs ε show the same result. We see that WCSS is very large for small ε, meaning that even if some points are labeled correctly, the actual centroids can be far from the clusters due to the added noise.

The number of iterations T also affects accuracy. While more iterations can improve the estimation of cluster centers, they also increase the amount of noise added at each step. Since the noise scale is $\frac{2T}{\varepsilon}$, larger T adds more noise and can hurt performance. This trade-off is shown in Figure 5 below:

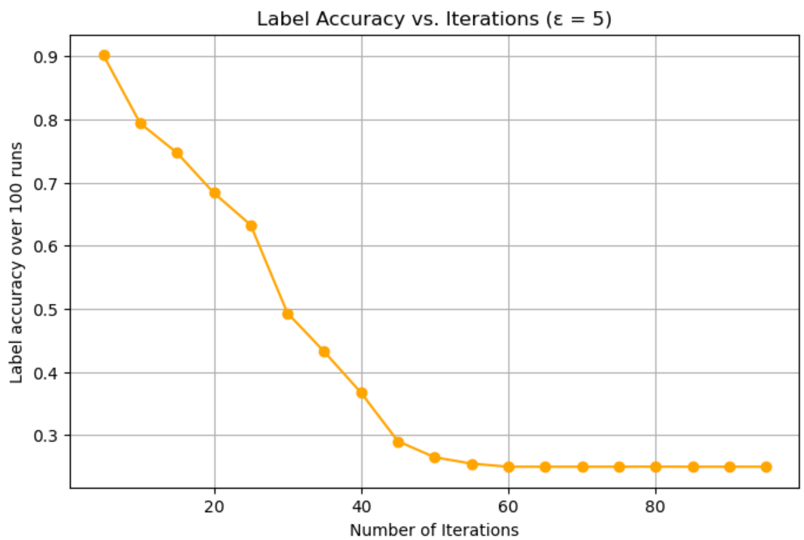

Figure 5: Impact of $\varepsilon = 5$ and different T on label accuracy

We see that label accuracy drops as T increases for a fixed ε. This shows the importance of balancing both the number of iterations and the privacy budget when building iterative differentially private algorithms.

Problem 2 [20 points]

Repeat the steps in Problem 1, but with (ε, δ)-DP instead of ε-DP. (How does this change your choice of the privacy budget? How do you choose δ?...)

Here is (ε, δ)-DP K-Means algorithm by adding noise using the Gaussian mechanism:

Algorithm: (ε, δ)-DP K-Means

Require: Dataset X, number of clusters k, number of iterations T, privacy parameter $\varepsilon > 0$, failure prob $\delta > 0$

Ensure: Final cluster centers $\{c_1, c_2, \ldots, c_k\}$

Set privacy budget and failure prob: $\varepsilon' = \frac{\varepsilon}{2T}, \quad \delta' = \frac{\delta}{2T}$

Initialize cluster centers $\{c_1, \ldots, c_k\}$ by randomly sampling k data points from X

for $t = 1$ to T **do**

  Assign each point x_i to the nearest cluster center:

$$k_i = \arg \min_{j \in \{1,\ldots,k\}} \|x_i - c_j\|_2^2$$

  for $j = 1$ to k **do**

    Create jth cluster set of all points assigned $k_i = j$:

$$K_j = \{i \mid k_i = j\}$$

    Compute cluster size:

$$n_j = |K_j|$$

    Compute cluster sum:

$$a_j = \sum_{i \in K_j} x_i$$

    Add noise to cluster size ($\Delta = 1$):

$$\hat{n}_j = n_j + \mathcal{N}(0, \sigma^2), \quad \sigma^2 = \frac{2\ln(1.25/\delta')\Delta^2}{\varepsilon'}$$

    Add noise to cluster sum ($\Delta = 1$:

$$\hat{a}_j = a_j + (Y_1', \ldots, Y_d') \sim \mathcal{N}(0, \sigma^2 I_d)$$

    Update cluster center:

$$c_j = \begin{cases} \frac{\hat{a}_j}{\hat{n}_j} & \text{if } \hat{n}_j > 0 \\ \text{RandomSample}(X) & \text{otherwise} \end{cases}$$

  end for

end for

return $\{c_1, c_2, \ldots, c_k\}$

(ε, δ)-DP K-Means Analysis The algorithm satisfies (ε, δ)-differential privacy since I used the Gaussian mechanism (which adds (ε, δ)-DP noise). The analysis of the privacy budget and failure probability can be done using the composition theorem. The privacy analysis for ε follows the same idea from Question 1 and can be extended to include δ. Note that δ is very small, typically $\leq 1/n$. We can get a tighter privacy guarantee with advanced composition, but I didn't do it as it was too complex :)
The noise is added using independent mean-zero Gaussian distribution with variance

$$\sigma^2 = \frac{2\log(1.25/\delta')\Delta^2}{\varepsilon'^2}$$

to both the cluster sizes and cluster sums. Sensitivity analysis is also the same as before using histogram bins. Both the cluster size and sum sensitivity is 1 as in Question 1.

Thus, adding Gaussian noise calibrated to this sensitivity ensure the algorithm satisfies (ε, δ)-differential privacy for the chosen privacy budget.

Now for numerical results. First, I show results of single run with $\varepsilon = 5$, $\delta = 1/400$ over $T = 5$ iterations.

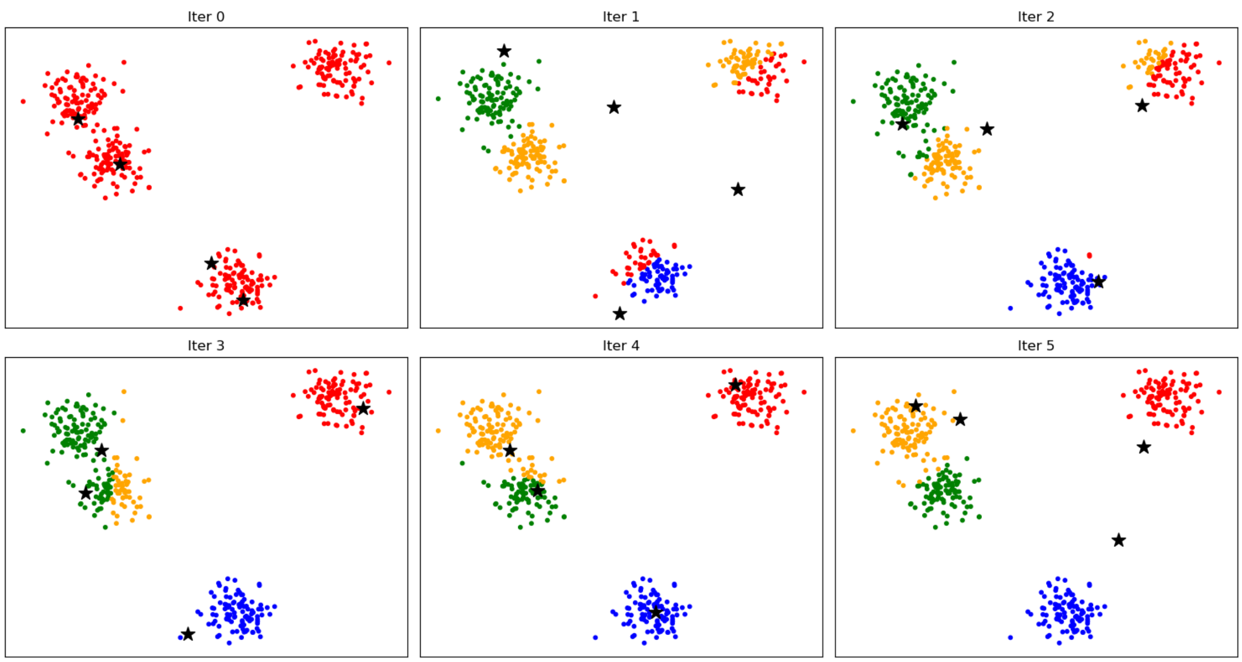

Figure 6: Cluster results with private (ε, δ)-DP K-Means.

The final cluster centroids deviate considerably from their expected positions. This change is due to the amount of noise added to the cluster counts and sums during each iteration. The WCSS is 20.50, which is substantially higher than in non-private runs. Label accuracy is 85%. Next, I show results on the effect of the privacy budget on WCSS and label accuracy (over 100 runs).

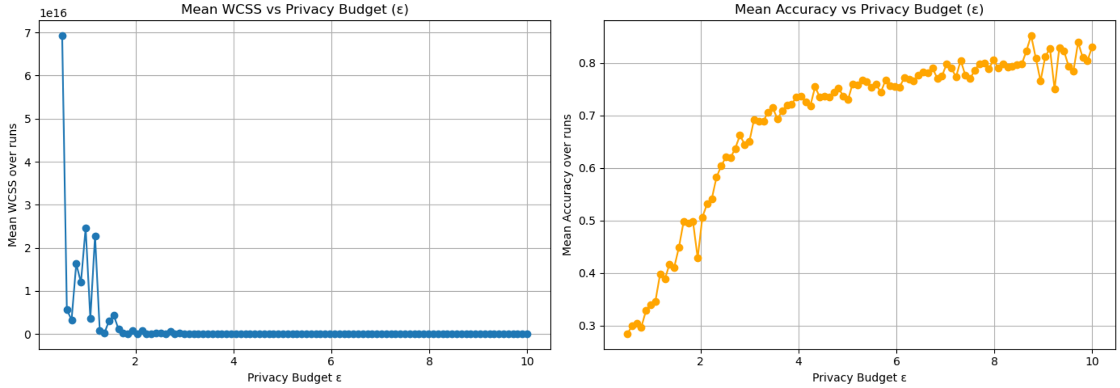

Figure 7: (ε, δ)-DP accuracy results with varying ε ($T = 5$ over 100 runs, $\Delta = 1/400$)

The ε vs label accuracy plot shows that as ε increases, the percent of correctly labeled points increases up to 80%. The accuracy is initially very low for small ε values, and this is also reflected in the WCSS vs ε plot. In homework 2, I noticed a trend of ε-DP algorithm with Laplace mechanism to be generally more accurate, this seems to hold here as well. Overall, similar results can be seen as question 1.

Problem 3 [40 points]

This problem is about evaluating the fairness of COMPAS, which is a tool used in many jurisdictions in the United States to predict recidivism risk, that is, the risk that a criminal defendant will reoffend. COMPAS assigns scores from 1 (lowest risk) to 10 (highest risk) to each defendant. The prediction algorithm behind COMPAS uses 137 "features" (including criminal history, age, gender, and criminal history of the defendant, but not race!) to generate the risk score.

Download the test data `compas-scores.csv` from the course webpage. We want to analyze whether COMPAS is fair or unfair with respect to the two fairness definitions: equalized odds and sufficiency (also known as predictive parity). In this problem, we will focus specifically on African-American and Caucasian defendants, as these two groups were at the center of the dispute between ProPublica and Northpointe.

We will first turn the problem into a binary classification problem. That means we translate COMPAS' predicted risk score contained in the column `decile_score` into a binary label by setting any decile score ≥ 6 equal to 1 and any decile score ≤ 4 equal to 0. (We ignore any decile score equal to 5, since this essentially amounts to random guessing.) The true recidivism value (i.e., if a defendant committed another crime or not in the next two years) is contained in the column `two_year_recid`.

Verify whether this binary classifier satisfies (approximately) or violates (approximately) the fairness criteria of equalized odds and sufficiency.

Here are the relevant mathematical definitions for each metric:

Equalized Odds:

$$P(\hat{Y} = 1 \mid A = 0, Y = y) = P(\hat{Y} = 1 \mid A = 1, Y = y), \quad \forall y \in \{0, 1\}$$

We want both the true positive rate (TPR) and false positive rate (FPR) to be approximately equal across groups.

Sufficiency:

$$P(Y = 1 \mid A = 0, \hat{Y} = \hat{y}) = P(Y = 1 \mid A = 1, \hat{Y} = \hat{y}), \quad \forall \hat{y} \in \hat{\mathcal{Y}}$$

We want the actual label Y to be independent of the sensitive attribute A, conditional on the predicted score. This would need to hold true for all possible predicted scores for the model to fully satisfy sufficiency. For COMPAS data, the set $\hat{\mathcal{Y}}$ represents the predicted risk scores: $\hat{\mathcal{Y}} = \{1, 2, 3, 4, 6, 7, 8, 9\}$. The sensitive attribute A is $\{0 : \text{African American},\ 1 : \text{Caucasian}\}$.

	Predicted = 0	**Predicted = 1**
Actual = 0	990 (TN)	616 (FP)
Actual = 1	532 (FN)	1193 (TP)

Table 1: Confusion Matrix for African-American Group

	Predicted = 0	**Predicted = 1**
Actual = 0	1139 (TN)	219 (FP)
Actual = 1	461 (FN)	394 (TP)

Table 2: Confusion Matrix for Caucasian Group

y	$P(Y{=}1 \mid A{=}0, \hat{\mathcal{Y}}{=}y)$	$P(Y{=}1 \mid A{=}1, \hat{\mathcal{Y}}{=}y)$	$P(Y{=}0 \mid A{=}0, \hat{\mathcal{Y}}{=}y)$	$P(Y{=}0 \mid A{=}1, \hat{\mathcal{Y}}{=}y)$
1	0.229	0.209	0.771	0.791
2	0.303	0.313	0.697	0.687
3	0.419	0.341	0.581	0.659
4	0.460	0.396	0.540	0.604
6	0.560	0.572	0.440	0.428
7	0.593	0.615	0.407	0.385
8	0.682	0.719	0.318	0.281
9	0.708	0.694	0.292	0.306
10	0.794	0.703	0.206	0.297

Table 3: Conditional probabilities of Y given predicted score $\hat{Y}$ and sensitive attribute A. $A = 0$: African-American, $A = 1$: Caucasian.

To show if the COMPAS algorithm *approximately* satisfies equalized odds and sufficiency, I use a threshold of $\epsilon = 0.10$. If the group differences are larger than this threshold, then the metric is not satisfied.

Equalized Odds

We compute the True Positive Rate (TPR) and False Positive Rate (FPR) for each group:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

For the COMPAS data:

True Positive Rate (TPR):

$$\text{TPR}_{\text{AA}} = \frac{1193}{1193 + 532} = 0.691, \quad \text{TPR}_{\text{Cauc}} = \frac{394}{394 + 461} = 0.461$$

False Positive Rate (FPR):

$$\text{FPR}_{\text{AA}} = \frac{616}{616 + 990} = 0.384, \quad \text{FPR}_{\text{Cauc}} = \frac{219}{219 + 1139} = 0.161$$

Since the differences in TPR ($0.691 - 0.461 = 0.230$) and FPR ($0.384 - 0.161 = 0.223$) between groups are both larger the threshold $\epsilon = 0.10$, the COMPAS algorithm does **not** approximately satisfy equalized odds.

Sufficiency

The absolute differences for each group can be shown in the table below:

y	$\lvert P(Y=1 \mid A=0,\hat{\mathcal{Y}}) - P(Y=1 \mid A=1,\hat{\mathcal{Y}})\rvert$	$\lvert P(Y=0 \mid A=0,\hat{\mathcal{Y}}) - P(Y=0 \mid A=1,\hat{\mathcal{Y}})\rvert$
1	|0.229 - 0.209| = 0.020	|0.771 - 0.791| = 0.020
2	|0.303 - 0.313| = 0.010	|0.697 - 0.687| = 0.010
3	|0.419 - 0.341| = 0.078	|0.581 - 0.659| = 0.078
4	|0.460 - 0.396| = 0.064	|0.540 - 0.604| = 0.064
6	|0.560 - 0.572| = 0.012	|0.440 - 0.428| = 0.012
7	|0.593 - 0.615| = 0.022	|0.407 - 0.385| = 0.022
8	|0.682 - 0.719| = 0.037	|0.318 - 0.281| = 0.037
9	|0.708 - 0.694| = 0.014	|0.292 - 0.306| = 0.014
10	|0.794 - 0.703| = 0.091	|0.206 - 0.297| = 0.091

Table 4: Absolute differences in conditional probabilities by predicted score.

Since the differences $|P(Y=1|A=0,\hat{Y}=y) - P(Y=1|A=1,\hat{Y}=y)|$ and $|P(Y=0|A=0,\hat{Y}=y) - P(Y=0|A=1,\hat{Y}=y)|$ for all $y \in \hat{\mathcal{Y}}$ are less than $\epsilon = 0.10$, the COMPAS algorithm **approximately satisfies sufficiency.** In fact, it satisfies sufficiency for all scores except $y=3$ and $y=10$ using a stricter threshold of $\epsilon = 0.05$. Thus, we can say that the algorithm approximately satisfies sufficiency.

Here is the code:

```python
### problem 1 & 2 code
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linear_sum_assignment

file_path = "/Users/adityamittal/Desktop/final/kmeansexample.asc"
data = np.loadtxt(file_path)

# plot raw data
plt.figure(figsize=(6, 6))
plt.scatter(data[:, 0], data[:, 1], s=10, color='grey')
plt.title("kmeansexample.asc Data")
plt.xlabel("x_1")
plt.ylabel("x_2")
plt.grid(True)
plt.axis("equal")
plt.show()

### I initially created a non-private version of this K-Means class for STA 142B
### Assignment 2 by professor Wolgang Polonik.
### This class template allowed me to easily add noise to cluster sums and counts without redoing
### all the code for k-means logic. I added necessary functions to add noise as needed.
### Comments will help tell where privacy is being added and what's happening generally :)
### I show results for all privacy variations.

class KMeansClustering:

    def __init__(self, data: np.ndarray, d: int, k: int, tol: float, max_iter: int,
                 private: bool = False, epsilon: float = 1.0, delta: float = 1/data.shape[0],
                 private_mechanism: 'laplace', seed: int = 2025):
        """
        Clustering class for both private and non-private K-Means.
        private_mechanism: 'laplace' for eps-DP, 'gaussian' for (eps, delta)-DP
        """
        self.data = data
        self.d = d
        self.k = k
        self.tol = tol
        self.max_iter = max_iter
        self.n = data.shape[0]
        self.seed = seed
        self.private = private
        self.epsilon = epsilon
```

```python
        self.delta = delta
        self.private_mechanism = private_mechanism.lower()
        self.partitions = {i: [] for i in range(k)}
        self.centers = np.zeros((k, d))
        self.next_centers = np.zeros((k, d))
        self.labels = np.array([])
        self.counter = 0

        # split privacy budget for iterations
        if private:
            self.eps_prime = self.epsilon / (2 * self.max_iter)
            self.delta_prime = self.delta / (2 * self.max_iter)
            np.random.seed(self.seed)

    def initialize_centers(self, method: int = 1):
        if method == 0:
            self.centers = self.data[:self.k, :]
        # random intialization of k centers as data points
        elif method == 1:
            np.random.seed(self.seed)
            self.centers = self.data[np.random.choice(self.n, self.k, replace=False), :]

    # includes both private and non-private versions
    def search(self):
        self.partitions = {i: [] for i in range(self.k)}
        self.next_centers = np.zeros((self.k, self.d))

        # setup
        for i in range(self.n):
            distances = np.linalg.norm(self.data[i] - self.centers, ord=2, axis=1)
            label = np.argmin(distances)
            self.partitions[label].append(i)

        for j in self.partitions:
            indices = np.array(self.partitions[j], dtype=int)
            if indices.size == 0:
                self.next_centers[j] = self.centers[j]
                continue

            # non-private counts
            Cj = len(indices)
            sum_j = np.sum(self.data[indices, :], axis=0)

            if self.private:
```

```python
                if self.private_mechanism == 'laplace':
                    # eps-DP with laplace noise
                    scale = 1 / self.eps_prime
                    noisy_Cj = Cj + np.random.laplace(loc=0.0, scale=scale)
                    noisy_sum_j = sum_j + np.random.laplace(loc=0.0, scale=scale, size=self.d)
                elif self.private_mechanism == 'gaussian':
                    # (eps, delta)-DP with gaussian noise
                    sigma = np.sqrt(2 * np.log(1.25 / self.delta_prime)) / self.eps_prime
                    noisy_Cj = Cj + np.random.normal(loc=0.0, scale=sigma)
                    noisy_sum_j = sum_j + np.random.normal(loc=0.0, scale=sigma, size=self.d)
                else:
                    # bad argument
                    raise ValueError("Wrong private_mechanism argument.")

                # incase added noise gives negative sums
                noisy_Cj = max(noisy_Cj, 1e-6)
                self.next_centers[j] = noisy_sum_j / noisy_Cj
            else:
                self.next_centers[j] = sum_j / Cj

    # more helper functions to run K-Means
    def is_updated(self):
        return np.sum(np.abs(self.centers - self.next_centers)) >= self.tol

    def get_labels(self):
        self.labels = np.zeros(self.n, dtype=int)
        for cluster_label, indices in self.partitions.items():
            self.labels[indices] = cluster_label
        return self.labels

    def get_centers(self):
        return self.centers

    def get_clusters(self):
        return self.partitions

    def get_cost(self):
        self.cost = 0.0
        for cluster_label, indices in self.partitions.items():
            cluster_points = self.data[indices]
            center = self.next_centers[cluster_label]
            self.cost += np.sum(np.linalg.norm(cluster_points - center, axis=1) ** 2)
        return self.cost
```

```python
    # function to fit K-Means
    def fit_model(self, init_method: int = 1):
        self.initialize_centers(init_method)
        self.counter = 0

        while self.counter < self.max_iter:
            self.search()
            self.counter += 1

            if not self.is_updated():
                print(f"Convergence Reached! Number of Iterations: {self.counter}")
                break

            self.centers = np.copy(self.next_centers)
        else:
            print("Maximum Number of Iterations Reached!")

        self.get_labels()

    # function to fit K-Means AND plot at each iteration
    def fit_and_plot_iterations(self, init_method: int = 1):

        self.initialize_centers(init_method)
        self.counter = 0
        centers_list = [np.copy(self.centers)]
        labels_list = [np.zeros(self.n, dtype=int)]

        # copy from fit_model()
        while self.counter < self.max_iter:
            self.search()
            self.counter += 1
            self.get_labels()

            centers_list.append(np.copy(self.next_centers))
            labels_list.append(np.copy(self.labels))

            if not self.is_updated():
                print("Convergence Reached! Number of Iterations:", self.counter)
                break
            self.centers = np.copy(self.next_centers)
        else:
            print("Maximum Number of Iterations Reached!")

        # plots
```

```python
        num_plots = len(centers_list)

        plots_per_row = 3
        n_rows = (num_plots + plots_per_row - 1) // plots_per_row
        n_cols = min(num_plots, plots_per_row)

        fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 4 * n_rows), squeeze=False)

        colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown', 'pink', 'gray']
        for i in range(num_plots):
            row = i // plots_per_row
            col = i % plots_per_row
            ax = axes[row, col]
            ax.set_title(f"Iter {i}")
            cluster_colors = [colors[label % len(colors)] for label in labels_list[i]]
            ax.scatter(self.data[:, 0], self.data[:, 1], c=cluster_colors, s=10)
            ax.scatter(centers_list[i][:, 0], centers_list[i][:, 1], marker='*',
                    c='black', s=150, edgecolors='k')
            ax.set_xticks([])
            ax.set_yticks([])

        for j in range(num_plots, n_rows * n_cols):
            fig.delaxes(axes[j // plots_per_row, j % plots_per_row])
        plt.tight_layout()
        plt.show()


def compute_accuracy(pred_labels, true_labels, k):
    # compute accuracy of labels
    conf_matrix = np.zeros((k, k), dtype=int)
    for i in range(k):
        for j in range(k):
            conf_matrix[i, j] = np.sum((pred_labels == i) & (true_labels == j))
    row_ind, col_ind = linear_sum_assignment(-conf_matrix)
    new_pred_labels = np.zeros_like(pred_labels)
    for pred_label, true_label in zip(row_ind, col_ind):
        new_pred_labels[pred_labels == pred_label] = true_label
    return new_pred_labels

### experiments
# non-private single run
model = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5, private=False, seed=np.random.randint(0,
model.fit_and_plot_iterations()
print(f"Final cost (WCSS): {model.get_cost():.8f}")
```

```
labels = compute_accuracy(model.get_labels(), np.array([0]*100 + [1]*100 + [2]*100 + [3]*100), k=4)
accuracy = np.mean(labels == np.array([0]*100 + [1]*100 + [2]*100 + [3]*100))
print(f"Accuracy: {accuracy:.4f}")

# private single run: eps-dp
model_private = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5, private=True, private_mechanism=
model_private.fit_and_plot_iterations()
print(f"Final cost (WCSS): {model_private.get_cost():.8f}")
labels = compute_accuracy(model_private.get_labels(), np.array([0]*100 + [1]*100 + [2]*100 + [3]*100),
accuracy = np.mean(labels == np.array([0]*100 + [1]*100 + [2]*100 + [3]*100))
print(f"Accuracy: {accuracy:.4f}")

## e-dp accuracy over 100 runs
eps_values = np.linspace(0.5, 10, 100)
mean_costs = []
mean_accuracies = []
true_labels = np.array([0]*100 + [1]*100 + [2]*100 + [3]*100)

for eps in eps_values:

    # WCSS calculation
    costs = []
    for _ in range(100):
        seed = np.random.randint(0, 10000)
        model = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5,
                                 private=True, epsilon=eps, private_mechanism='laplace',  seed=np.rando
        model.fit_model(init_method=1)
        cost = model.get_cost()
        costs.append(cost)
    mean_costs.append(np.mean(costs))

    # Accuracy calculation
    accuracies = []
    for _ in range(100):
        model = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5,
                                 private=True, epsilon=eps, private_mechanism='laplace', seed=np.random
        model.fit_model(init_method=1)
        pred_labels = model.get_labels()
        corrected_labels = compute_accuracy(pred_labels, true_labels, k=4)
        accuracy = np.mean(corrected_labels == true_labels)
        accuracies.append(accuracy)
    mean_accuracies.append(np.mean(accuracies))

# Plot side-by-side
```

```
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
axs[0].plot(eps_values, mean_costs, marker='o', linestyle='-')
axs[0].set_title("WCSS vs ")
axs[0].set_xlabel("")
axs[0].set_ylabel("WCSS over 100 runs")
axs[0].grid(True)
axs[1].plot(eps_values, mean_accuracies, marker='o', color='orange')
axs[1].set_title("Label Accuracy vs ")
axs[1].set_xlabel("")
axs[1].set_ylabel("Label accuracy over 100 runs")
axs[1].grid(True)
plt.tight_layout()
plt.show()

# private single run: (e,delta)-dp
model_private = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5, private=True, private_mechanism=
model_private.fit_and_plot_iterations()
print(f"Final cost (WCSS): {model_private.get_cost():.8f}")
labels = compute_accuracy(model_private.get_labels(), np.array([0]*100 + [1]*100 + [2]*100 + [3]*100),
accuracy = np.mean(labels == np.array([0]*100 + [1]*100 + [2]*100 + [3]*100))
print(f"Accuracy: {accuracy:.4f}")

## e-dp accuracy over 100 runs
eps_values = np.linspace(0.5, 10, 100)
mean_costs = []
mean_accuracies = []
true_labels = np.array([0]*100 + [1]*100 + [2]*100 + [3]*100)

for eps in eps_values:

    # WCSS calculation
    costs = []
    for _ in range(100):
        seed = np.random.randint(0, 10000)
        model = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5,
                                  private=True, epsilon=eps, private_mechanism='gaussian',  seed=np.rand
        model.fit_model(init_method=1)
        cost = model.get_cost()
        costs.append(cost)
    mean_costs.append(np.mean(costs))

    # Accuracy calculation
    accuracies = []
    for _ in range(100):
```

```
        model = KMeansClustering(data, d=2, k=4, tol=1e-5, max_iter=5,
                                 private=True, epsilon=eps, private_mechanism='gaussian', seed=np.rando
        model.fit_model(init_method=1)
        pred_labels = model.get_labels()
        corrected_labels = compute_accuracy(pred_labels, true_labels, k=4)
        accuracy = np.mean(corrected_labels == true_labels)
        accuracies.append(accuracy)
    mean_accuracies.append(np.mean(accuracies))

# Plot side-by-side
fig, axs = plt.subplots(1, 2, figsize=(14, 5))
axs[0].plot(eps_values, mean_costs, marker='o', linestyle='-')
axs[0].set_title("WCSS vs ")
axs[0].set_xlabel("")
axs[0].set_ylabel("WCSS over 100 runs")
axs[0].grid(True)
axs[1].plot(eps_values, mean_accuracies, marker='o', color='orange')
axs[1].set_title("Label Accuracy vs ")
axs[1].set_xlabel("")
axs[1].set_ylabel("Label accuracy over 100 runs")
axs[1].grid(True)
plt.tight_layout()
plt.show()

### problem 3 code
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix

file_path = '/Users/adityamittal/Desktop/final/compas-scores.csv'
df = pd.read_csv(file_path)

# pre-processing
df = df[df['race'].isin(['African-American', 'Caucasian'])]
df = df[df['decile_score'] != 5]
df['predicted'] = df['decile_score'].apply(lambda x: 1 if x >= 6 else 0)
df['true'] = df['two_year_recid']
df['score'] = df['decile_score']

groups = ['African-American', 'Caucasian']

# equalized odds confusion matrix
for i in groups:
    subset = df[df['race'] == i]
```

```
    tn, fp, fn, tp = confusion_matrix(subset['true'], subset['predicted']).ravel()
    print(f"\nConfusion matrix for {i}:")
    print("TN:", tn)
    print("FP:", fp)
    print("FN:", fn)
    print("TP:", tp)

# probabilities for each score - sufficiency metric
scores = sorted(df['score'].unique())
print("\nProbabilities by score and group:")
for i in scores:
    print(f"\nScore = {i}")
    for group in groups:
        subset = df[(df['race'] == group) & (df['score'] == i)]
        print(f"  {group}: P(Y=1)={subset['true'].mean()}, P(Y=0)={1 - subset['true'].mean()}")

```