Datasets

There were four different datasets used in the project for different machine learning algorithms. For the unsupervised machine learning algorithm, K-means clustering, student clustering dataset [1] consisting of 200 data points and 2 columns namely: CGPA and IQ was used. For Naïve Bayes Classification, mushroom [2] and iris [3] datasets were used. Mushroom dataset consisted of 22 categorical variables and a binary target class with 8124 data points while iris dataset consisted of four numerical variables and three target classes with 150 data points. For simple linear regression, salary dataset [4] was used which consisted of two features namely: years of experience and salary and it consisted of 30 data points.

Privacy and Security Technique

The privacy and security technique employed during the project was differential privacy which mathematical framework for ensuring the privacy of individuals in datasets. It can provide a strong guarantee of privacy by allowing data to be analyzed without revealing sensitive information about any individual in the dataset. Considering two neighboring databases D_1 and D_2 the mechanism is said to be ε-differentially private if an attacker can't tell whether D_1 or D_2 has an individual present or absent, i.e., the output of the query from the data is not an indicator of the individual's presence or absence in the database leading to a strong privacy for the individuals in the database. Thus, a mechanism is differentially private when $\Pr[A(D_1) = O] <= e^{\varepsilon} \Pr[A(D_2) = O]$. Smaller the value of ε, stronger is the privacy of the mechanism. But to achieve that sort of privacy a lot of noise has to be injected leading to poor utility. Depending on the nature of query, different types of noises are injected to achieve differential privacy. Sensitivity plays a major role in the noise being injected. For any query q over the input datasets, the sensitivity of q is $\Delta q = \max \|q(D_1) - q(D_2)\|$ for any two neighboring datasets D_1 and D_2. For queries with numeric output, Laplace noise is added while for non-numeric outputs, exponential mechanism is utilized. Different concepts related to differential privacy like composition and privacy budget are need to be considered while training ε-differentially private machine learning algorithms. This included about to combine different queries while we still need to guarantee ε-differential privacy. Different types of composition like sequential or parallel help in determining the allocation of the privacy budget among different queries. In sequential composition, the combined privacy guarantee of all queries in the sequence can be bounded by the sum of their individual epsilon values i.e. $\varepsilon = \varepsilon_1+\varepsilon_2+\ldots+\varepsilon_k$. While for certain types of queries and noise-adding mechanisms, the privacy guarantee of the combined analysis can be bounded by the maximum epsilon value of the individual queries i.e. $\varepsilon = \max\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k\}$. These concepts combined help in designing ε-differentially private mechanisms.

K-Means Clustering

K-Means Clustering is an unsupervised machine learning algorithm so there is no labeled data for this clustering, which is different in case of supervised learning. It forms clusters of points that share similarities and are dissimilar to the points present in the other clusters. The variable k determines the number of clusters to be formed. K-Means clustering is used in many fields like customer segmentation, fraud detection, anomaly detection, targeting incentives to customers, etc. The function to be minimized in K-means is the inertia or within-cluster sum of squares given as follows:

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

where:

- k: Number of clusters
- x_j: Data point
- S_i: Cluster containing data points
- μ_i: Mean (Centroid) of cluster S_i
- $\|\cdot\|$ is the usual ℓ^2 norm

The K-Means algorithm starts with randomly initializing k centroids for the k clusters and assign the point in the dataset to its nearest centroid. After the assignment, the centroids are recalculated and again the nearest points to the centroids are assigned to that cluster until the algorithm has converged when the assignments don't change.

For implementing, differentially private K-Means, we fix the number of iterations to a particular value. Suppose, the fixed number of iterations is T (T = 10 for the project) then each iteration uses ε/T privacy budget where the total privacy loss is ε. Therefore, in each iteration of getting a new centroid value, noise is injected while computing the size of each cluster and the sums of points in each cluster. As this is to be done, on the entire data, by sequential composition the ε/T changes to ε/2T. The sensitivity for the computing the size of each cluster is 1 cause between two neighboring datasets a point could be either in the cluster or not. The sensitivity for computing the sums of points in each cluster is equal to the difference between the two extremes in the data. Thus, the noises added while computing size of cluster and sums of points in each cluster are Laplace (2T/ε) and Laplace (2T|domain|/ε). The mechanism satisfies ε-differential privacy since the total privacy loss of ε is maintained while allocating it to the two queries as (ε/2T + ε/2T)*T = ε. Hence, the mechanism satisfies ε-differential privacy. Figure 1 and 2 show the injection of noise for computing the size of each cluster and sums of points in each cluster.

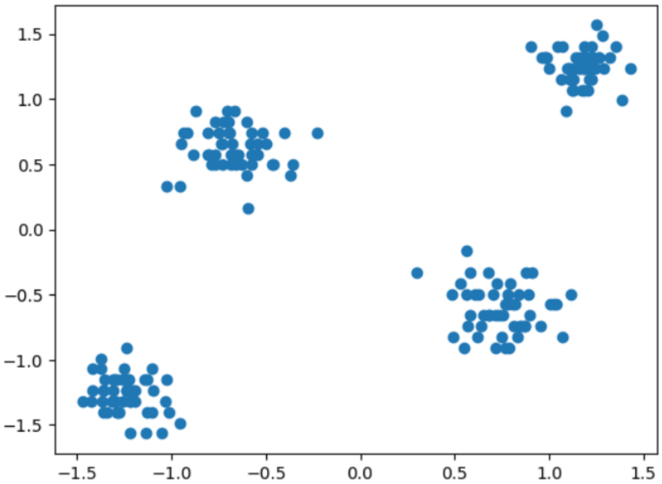


Fig. 1. Scatter Plot of Scaled Student Clustering Dataset

```
#Noisily compute the size of each cluster
size_noise = np.random.laplace(loc = 0, scale = ((2*max_iter)/epsilon))
size = len(cluster_points) + size_noise
```

Fig. 2. Adding noise to the size of each cluster

```
#Compute noisy sums of points in each cluster
x = [sub[0] for sub in cluster_points]
y = [sub[1] for sub in cluster_points]
if len(x) == 0:
  max_x = 0
  min_x = 0
else:
  max_x = max(x)
  min_x = min(x)
if len(y) == 0:
  max_y = 0
  min_y = 0
else:
  max_y = max(y)
  min_y = min(y)
domain_x = abs(max_x - min_x)
domain_y = abs(max_y - min_y)
sum_x_noise = np.random.laplace(loc = 0, scale = ((2*max_iter*domain_x)/epsilon))
sum_y_noise = np.random.laplace(loc = 0, scale = ((2*max_iter*domain_y)/epsilon))
sum_x = sum(x) + sum_x_noise
sum_y = sum(y) + sum_y_noise
x_c = sum_x/size
y_c = sum_y/size
```

Fig. 3. Adding noise to sums of points in each cluster

Naïve Bayes Classifier (Categorical Attributes)

Naïve Bayes classifier is a simple yet effective supervised machine learning algorithm to perform classification tasks. It is based on the Bayes' theorem. It is called Naïve because of its assumption that the presence of a particular feature in a class is unrelated to the presence of any other feature. Bayes theorem provides a way of calculating the posterior probability, $P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$.

Likelihood　　　Class Prior Probability

$$P(c \mid x) = \frac{P(x \mid c)\,P(c)}{P(x)}$$

Posterior Probability　　　Predictor Prior Probability

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- $P(c|x)$ is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Jaideep Vaidya et al. [5] in there research work for differentially private Naïve Bayes Classification, made an assumption for the categorical attributes that all possible values for a give attribute are already known. Based on this they performed differential privacy on Naïve Bayes classifier models trained on categorical attributes. As for a nominal attribute X with r possible attributes values $x_1,...,x_r$, the probability $P(X = x_k|c_j) = n_{kj}/n$ where n is the total number of training examples for which $C = c_j$, and n_{kj} is the number of those training examples that also have $X = x_k$. Here, sensitivity calculations can be performed on the counts and its value would be 1, since the presence or absence of a record in the dataset can increment or decrement the count by maximum of 1. Thus, the count is injected with Laplace noise of Laplace(1/ε) and a ε-differentially private Naïve Bayes classifier is trained and tested for different epsilon values on the mushroom dataset.

Algorithm 1 Computing differentially private parameters for Naïve Bayes

Require: ϵ, the privacy parameter for differential privacy
Require: Laplace(a, b) samples the Laplace distribution with mean a and scale b

for each attribute X_j **do**
  if X_j is categorical **then**
    sensitivity, $s \leftarrow 1$
    scale factor, $sf \leftarrow s/\epsilon$
    $\forall$ counts n_{kj}, $n'_{kj} = n_{kj} +$ Laplace$(0, sf)$
    Use n'_{kj} to compute $P(x_i|c_j)$
  else if X_j is numeric **then**
    compute sensitivity, s for mean μ_j as per equation 5
    scale factor, $sf \leftarrow s/\epsilon$
    $\mu'_j \leftarrow \mu_j +$ Laplace$(0, sf)$
    compute sensitivity, s for standard deviation σ_j as per equation 7
    scale factor, $sf \leftarrow s/\epsilon$
    $\sigma'_j \leftarrow \sigma_j$ + Laplace$(0, sf)$
    Use μ'_j and σ'_j to compute $P(x_i|c_j)$
  end if
end for
for each class c_j **do**
  count $nc'_j \leftarrow nc_j +$ Laplace$(0, 1)$
  Use nc'_j to compute the prior $P(c_j)$
end for

Fig. 4. Algorithm for DP parameters for Naïve Bayes classifier by Jaideep Vaidya et al. [5]

```
cnt = np.sum(X_class[feature] == feature_value)
#Adding Laplace noise to count to ensure ε-Differential Privacy
count_sensitivity = 1
count_epsilon = epsilon
count_noise = np.random.laplace(loc = 0, scale = count_sensitivity/count_epsilon)
count = cnt + count_noise
class_n = y_train.value_counts()[class_label]
cond_prob = count/class_n
```

Fig. 5. Adding noise to count n_{kj} for whom $X = x_k$ and $C = c_j$

Naïve Bayes Classifier (Numerical Attributes)

For numerical attributes, the probability $P(X = x \mid c_j)$ depends on the mean μ_j and standard deviation σ_j, where the mean μj and variance $\sigma^2{}_j$ are calculated for class j based on the values of attribute X from the training set. The probability is calculated using the Gaussian probability density function. Jaideep Vaidya et al. [5] assumes that if attribute X_j lies in the range $[l_j, u_j]$, then in the worst case, the difference in means between two neighbouring datasets is bounded by $(u_j - l_j)/(n + 1)$. Thus, the sensitivity of mean is equal to $(u_j - l_j)/(n + 1)$. Similarly, the sensitivity of standard deviation is found out to be equal to $\mathrm{sqrt}(n) * (u_j - l_j)/(n + 1)$. Further, while calculating the prior probabilities $P(c_j)$, the sensitivity of the prior count is 1. Therefore, three Laplace noises are injected while training the Naïve Bayes Classifier model on numerical data. For iris dataset containing four variables, the privacy loss of ε is in sequential composition for the three noises and hence, we can allocate a privacy budget of ε/3 to each of them. Further, injecting noises while computing mean and standard deviation for the different numerical attributes follow sequential composition leading to the budget being ε/12 for both the queries. As for the count query, it depends on a particular target class in the dataset, it satisfies parallel composition and thus, its privacy budget remains ε/3. Thus, we satisfy ε-differential privacy as $((\varepsilon/12+\varepsilon/12)*4 + \varepsilon/3) = \varepsilon$. Therefore, we can ensure that the model is ε-differentially private Naïve Bayes classifier on the iris dataset.

```
avg = sum(class_column)/len(class_column)
uj = max(class_column)
lj = min(class_column)
n = len(class_column)
mean_sensitivity = (uj-lj)/(n+1)
mean_epsilon = epsilon/(3*4)
mean_noise = np.random.laplace(loc = 0, scale = (mean_sensitivity/mean_epsilon))
mean = avg + mean_noise
```

Fig. 6. Adding noise to mean of a feature column within same target class

```
uj = max(class_column)
lj = min(class_column)
n = len(class_column)
avg = np.mean(class_column)
sd = np.sqrt(sum([(x-avg)**2 for x in class_column]) / float(len(class_column)-1))
sd_sensitivity = (np.sqrt(n))*((uj-lj)/(n+1))
sd_epsilon = epsilon/(3*4)
sd_noise = np.random.laplace(loc = 0, scale = (sd_sensitivity/sd_epsilon))
std_dev = sd + sd_noise
```

Fig. 7. Adding noise to standard deviation of a feature column within same target class

Simple Linear Regression (SLR)

Simple linear regression (SLR) is a supervised machine learning algorithm used to perform regression task when the output or the target variable is continuous. It is trained on two features where one is the predictor and the other is the target variable. It aims to find the linear relationships between the two variables and predict values. SLR has the form of $y = \alpha + \beta x$, where α and β is the intercept and slope of the regression line, respectively. The values of α and β are calculated using the following formulas:

$$\widehat{\alpha} = \bar{y} - (\widehat{\beta}\,\bar{x}),$$

$$\widehat{\beta} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Daniel Alabi et al. [6] in there research work on differentially private simple linear regression considered a differential privacy mechanism NoisyStats in simple linear regression. It involves injecting Laplace noise into the ncov(x,y) and nvar(x) statistics from the ordinary least square calculations.

Algorithm 1: `NoisyStats`: $(\varepsilon, 0)$-DP Algorithm

Data: $\{(x_i, y_i)\}_{i=1}^{n} \in ([0,1] \times [0,1])^n$
Privacy params: ε
Hyperparams: n/a
Define $\Delta_1 = \Delta_2 = (1 - 1/n)$
Sample $L_1 \sim \mathrm{Lap}(0, 3\Delta_1/\varepsilon)$
Sample $L_2 \sim \mathrm{Lap}(0, 3\Delta_2/\varepsilon)$
if $nvar(\mathbf{x}) + L_2 > 0$ **then**
  $\tilde{\alpha} = \frac{\mathrm{ncov}(\mathbf{x},\mathbf{y})+L_1}{\mathrm{nvar}(\mathbf{x})+L_2}$
  $\Delta_3 = 1/n \cdot (1 + |\tilde{\alpha}|)$
  Sample $L_3 \sim \mathrm{Lap}(0, 3\Delta_3/\varepsilon)$
  $\tilde{\beta} = (\bar{y} - \tilde{\alpha}\bar{x}) + L_3$
  $\tilde{p}_{25} = 0.25 \cdot \tilde{\alpha} + \tilde{\beta}$
  $\tilde{p}_{75} = 0.75 \cdot \tilde{\alpha} + \tilde{\beta}$
  return $\tilde{p}_{25}, \tilde{p}_{75}$
else
  return $\perp$

Fig. 8. Algorithm for computing DP parameters for SLR by Daniel Alabi et al. [6]

As seen in the above algorithm, the intercept of the regression line α which is injected with two Laplace noises with sensitivity = 1 – 1/n where n in the data size and the slope of the regression line β is injected with one Laplace noises with sensitivity = 1/n*(1+| α |).

```
#Compute noisy covariance and variance for alpha
alpha_sensitivity = 1-(1/len(X))
alpha_epsilon = epsilon/3
alpha_noise = np.random.laplace(loc = 0, scale = alpha_sensitivity/alpha_epsilon)
```

Fig. 9. Adding noise to ncov(x,y) and nvar(x) statistics

```
#Compute beta (Intercept of regression line)
beta_sensitivity = (1/len(X))*(1 + abs(alpha))
beta_epsilon = epsilon/3
beta_noise = np.random.laplace(loc = 0, scale = beta_sensitivity/beta_epsilon)
beta = (np.mean(y) - (alpha*np.mean(X))) + beta_noise
```

Fig. 10. Adding noise to compute beta (slope)

Results

<u>K-Means Clustering:</u>

From Fig. 1, it is pretty certain that the number of clusters in the dataset are four i.e. k = 4. But, it can be verified by Elbow Method and the results of it can be seen in Fig. 11.

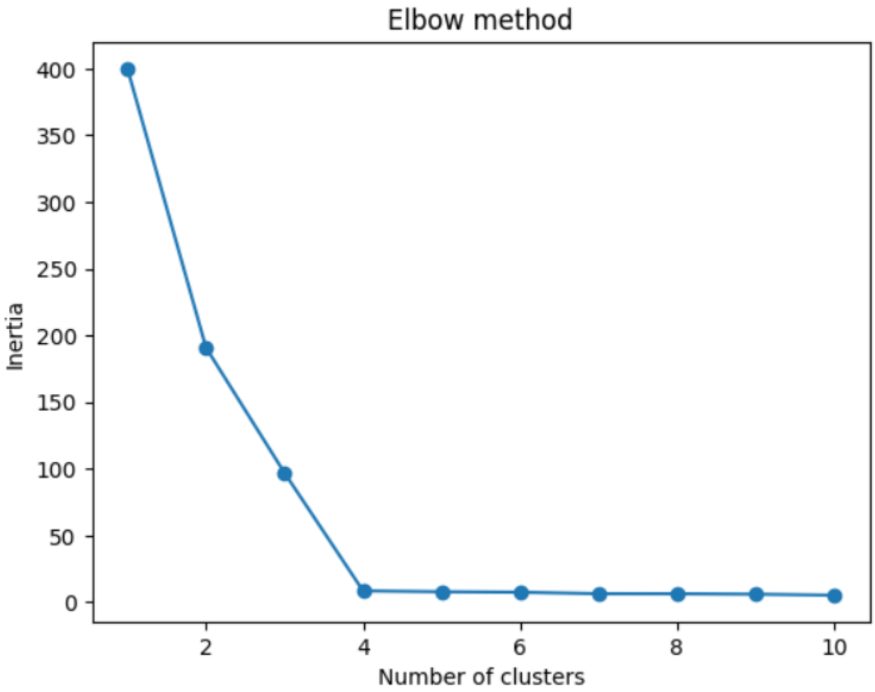
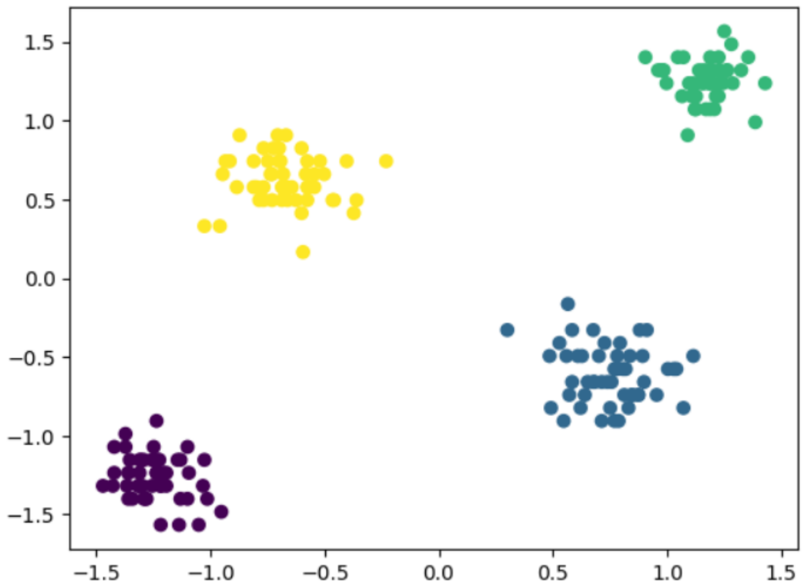


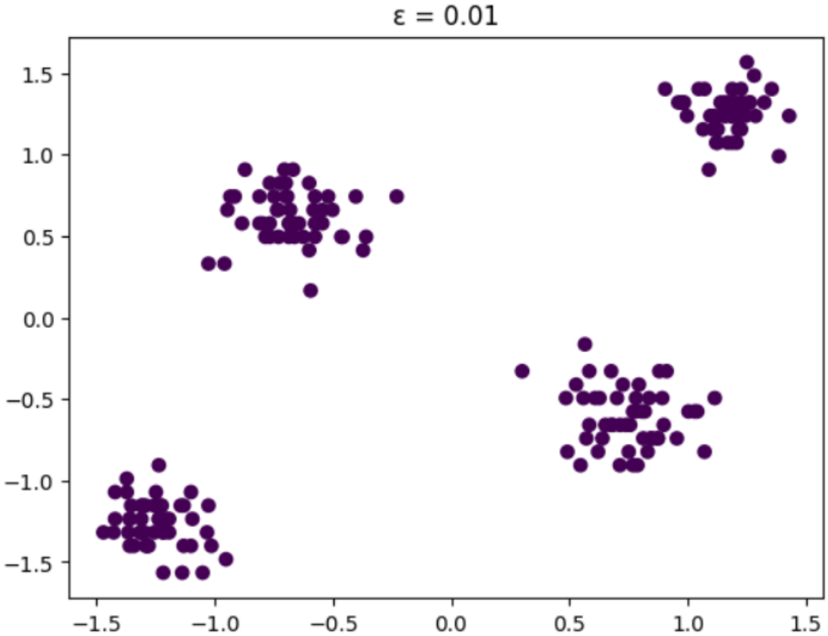
Fig. 11. Elbow Method to find optimal value of k




```
Inertia or within-cluster sum ofsquares: 8.477237626556295
Number of iterations: 4
Centroids:
 [[-1.24429338 -1.26080305]
 [ 0.75084754 -0.60958735]
 [ 1.16540625  1.25008684]
 [-0.6719604   0.62030356]]
```

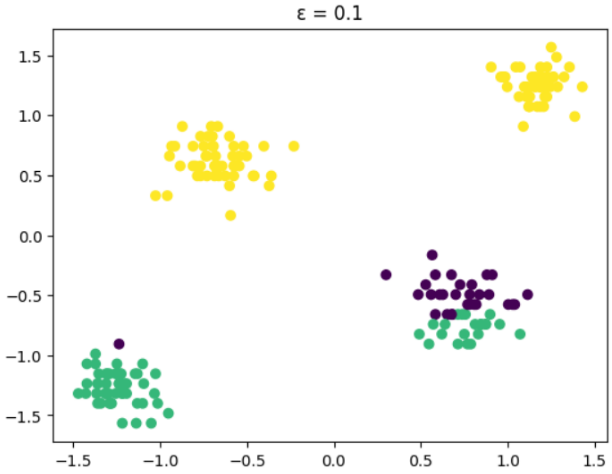
Fig. 12. K-Means clustering using sklearn with k = 4

Fig. 12. shows the output of the normal Kmeans clustering algorithm with k = 4 and it can be observed that the data is pretty good for clustering and the number of iterations taken to find convergence is 4.



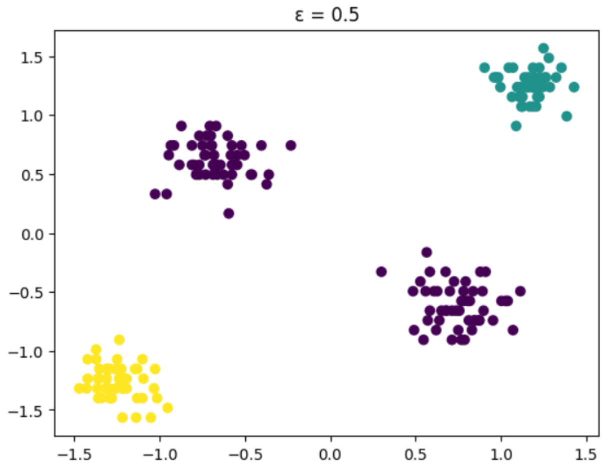

```
Inertia or within-cluster sum ofsquares: 6345.065413535866
Number of iterations: 10
Centroids:
 [[0.0, 0.0], [2.868143274187415, 4.636710172785531], [0.0, 0.0], [-0.0, -0.0]]
```

Fig 13. DP-KMeans with ε = 0.01



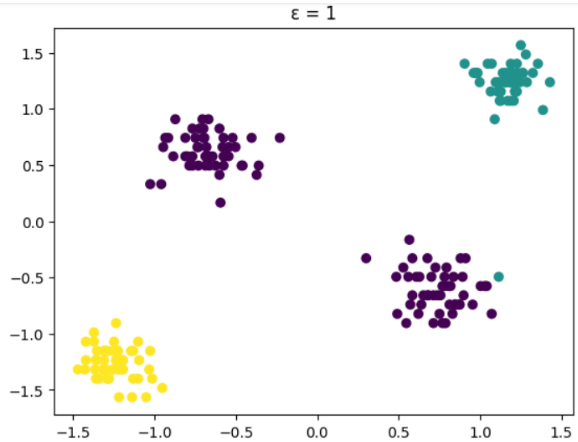

```
Inertia or within-cluster sum ofsquares: 80873.37374361423
Number of iterations: 10
Centroids:
 [[0.6919915340545056, 1.1764156005456592], [0.0, 0.0], [-34.24327340375698, -4.800157787442781], [1.8188359431192982, -1.031744351422992]]
```

Fig 14. DP-KMeans with ε = 0.1



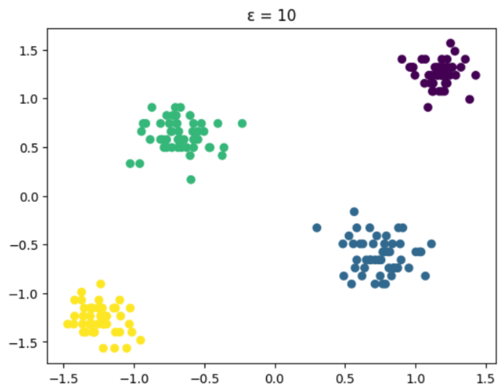

```
Inertia or within-cluster sum ofsquares: 450.7307446355621
Number of iterations: 10
Centroids:
 [[1.3664446602518003, 0.11675916740571718], [1.9406235762078818, 1.884278979108949], [-1.9544538226969805, -2.682910459206153], [-0.0, -0.0]]
```

Fig 15. DP-KMeans with ε = 0.5




```
Inertia or within-cluster sum ofsquares: 202.61423201232532
Number of iterations: 10
Centroids:
 [[0.08288790407217143, -0.019416093749500644], [1.4253782965988337, -0.13288717741324702], [-1.0805120954550382, -0.9093944811344143], [-0.0, -0.0]]
```

Fig 16. DP-KMeans with ε = 1




```
Inertia or within-cluster sum ofsquares: 9.034051157330145
Number of iterations: 10
Centroids:
 [[1.1500462744250572, 1.2378065047011884], [0.7761036471285595, -0.6233618298360162], [-0.6713405775161183, 0.7083252434990456], [-1.247375399918525, -1.3073243178142286]]
```

Fig 17. DP-KMeans with ε = 10

Figures 13, 14, 15, 16 and 17 show the clusters formed after applying DP-KMeans with ε = 0.01, 0.1, 0.5, 1 and 10 respectively. It can be observed that as ε increases the inertia/ within-clusters sum of squares decreases as the noise injected reduces and privacy decreases.

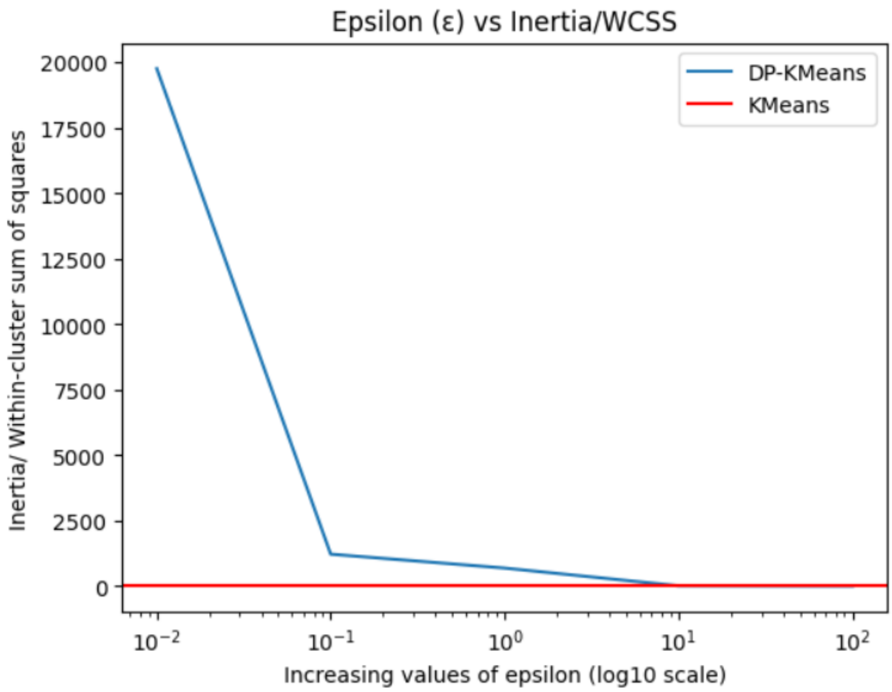


Fig 18. Epsilon (ε) vs Inertia

From Fig 18, it can be seen that the inertia decreases as epsilon increases and as observed before the clusters become better concentrated.

Naïve Bayes (Categorical Attributes):

```
Performance evaluation of CategoricalNB:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      1257
           1       1.00      0.91      0.95      1181

    accuracy                           0.95      2438
   macro avg       0.96      0.95      0.95      2438
weighted avg       0.96      0.95      0.95      2438

[[1252    5]
 [ 111 1070]]

Accuracy Score:  0.9524200164068909
Precision Score:  0.9557585747378193
Recall Score:  0.9524200164068909
F1 Score:  0.9522647200904559
```

Fig 19. Performance evaluation of CategoricalNB()

Fig 19 shows the performance of CategoricalNB() from sklearn and it can be seen that the normal model is performing very good on the mushroom dataset.

```
Performance evaluation of cat_naivebayes:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1257
           1       1.00      1.00      1.00      1181

    accuracy                           1.00      2438
   macro avg       1.00      1.00      1.00      2438
weighted avg       1.00      1.00      1.00      2438

[[1252    5]
 [   1 1180]]

Accuracy Score:  0.9975389663658737
Precision Score:  0.9975445796959569
Recall Score:  0.9975389663658737
F1 Score:  0.9975390857156406
```

Fig 20. Performance evaluation of cat_naivebayes

Fig 20 shows the performance of cat_naivebayes, a categorical naïve bayes classifier coded from scratch and it can be seen that the model is performing better on the mushroom dataset than CategoricalNB.

```
Performance evaluation of 0.1-Differentially Private dp_cat_naivebayes:
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      1257
           1       0.99      0.94      0.96      1181

    accuracy                           0.97      2438
   macro avg       0.97      0.96      0.97      2438
weighted avg       0.97      0.97      0.97      2438

[[1243   14]
 [  70 1111]]

Accuracy Score:  0.9655455291222313
Precision Score:  0.9664842659693494
Recall Score:  0.9655455291222313
F1 Score:  0.9655025541982926
```

Fig 21. Performance evaluation of DP-Naïve Bayes with ε = 0.1

```
Performance evaluation of 0.5-Differentially Private dp_cat_naivebayes:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1257
           1       0.99      0.95      0.97      1181

    accuracy                           0.97      2438
   macro avg       0.97      0.97      0.97      2438
weighted avg       0.97      0.97      0.97      2438

[[1250    7]
 [  58 1123]]

Accuracy Score:  0.9733388022969647
Precision Score:  0.9741368083209058
Recall Score:  0.9733388022969647
F1 Score:  0.9733096705695583
```

Fig 22. Performance evaluation of DP-Naïve Bayes with ε = 0.5

```
Performance evaluation of 1-Differentially Private dp_cat_naivebayes:
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1257
           1       0.99      0.96      0.97      1181

    accuracy                           0.98      2438
   macro avg       0.98      0.98      0.98      2438
weighted avg       0.98      0.98      0.98      2438

[[1249    8]
 [  51 1130]]

Accuracy Score:  0.9757998359310911
Precision Score:  0.9763677761964338
Recall Score:  0.9757998359310911
F1 Score:  0.9757789524809065
```

Fig 23. Performance evaluation of DP-Naïve Bayes with ε = 1

```
Performance evaluation of 10-Differentially Private dp_cat_naivebayes:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1257
           1       1.00      0.99      0.99      1181

    accuracy                           0.99      2438
   macro avg       0.99      0.99      0.99      2438
weighted avg       0.99      0.99      0.99      2438

[[1252    5]
 [  12 1169]]

Accuracy Score:  0.9930270713699754
Precision Score:  0.9930421013493556
Recall Score:  0.9930270713699754
F1 Score:  0.9930263889879237
```

Fig 24. Performance evaluation of DP-Naïve Bayes with ε = 10

From Fig 21, 22, 23 and 24 we can observe the performance of DP-Naïve Bayes on mushroom dataset i.e. categorical attributes and it is again observed that as ε increases from 0.1 to 10, the performance of the model increases as the laplace noise injected has a narrower spread.

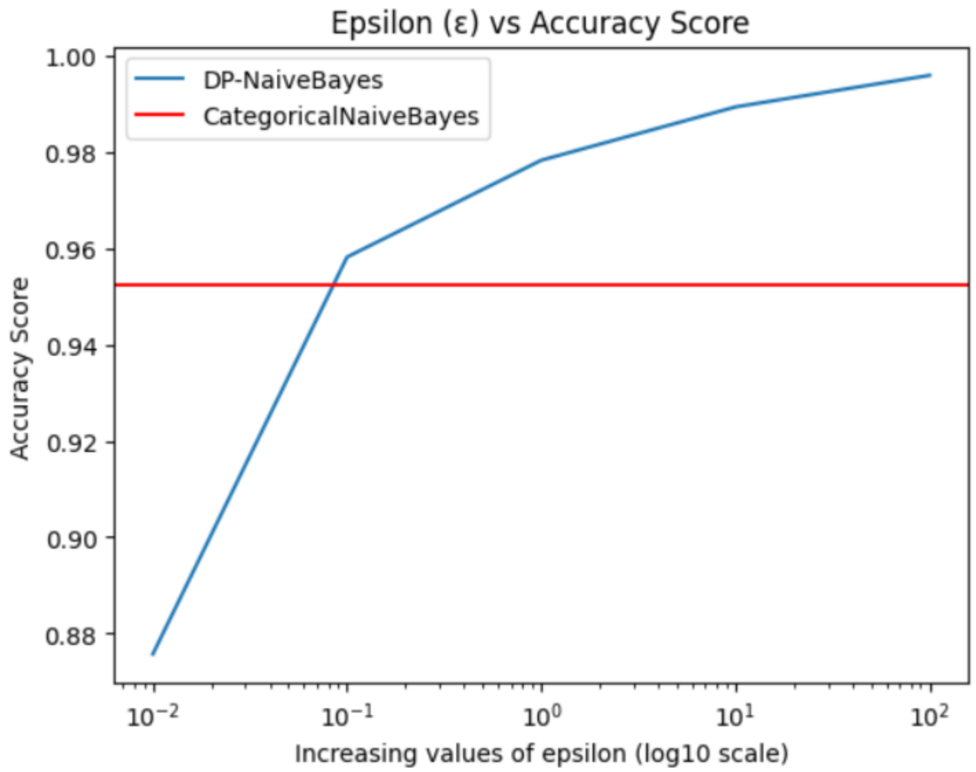


Fig 25. Epsilon (ε) vs Accuracy Score

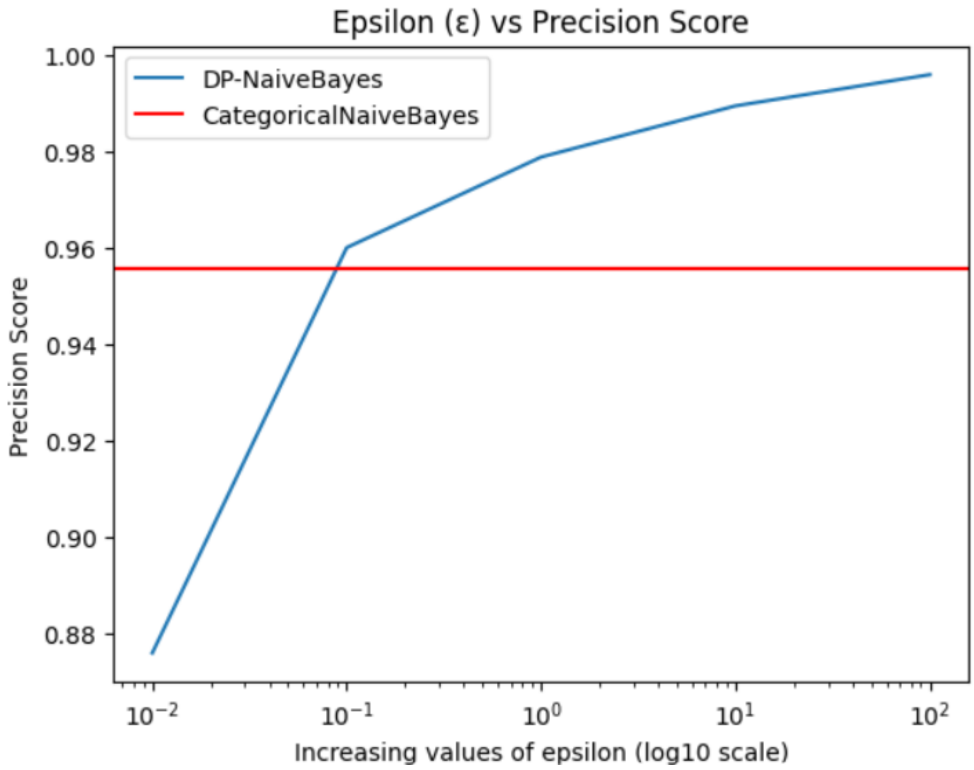


Fig 26. Epsilon (ε) vs Precision Score

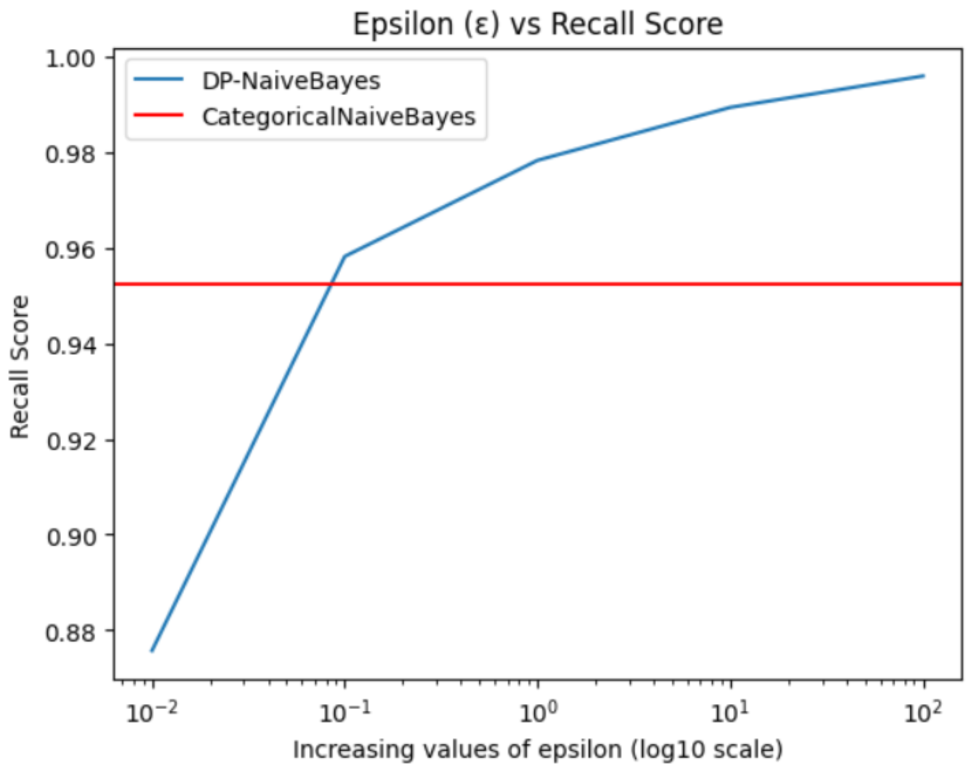


Fig 27. Epsilon (ε) vs Recall Score

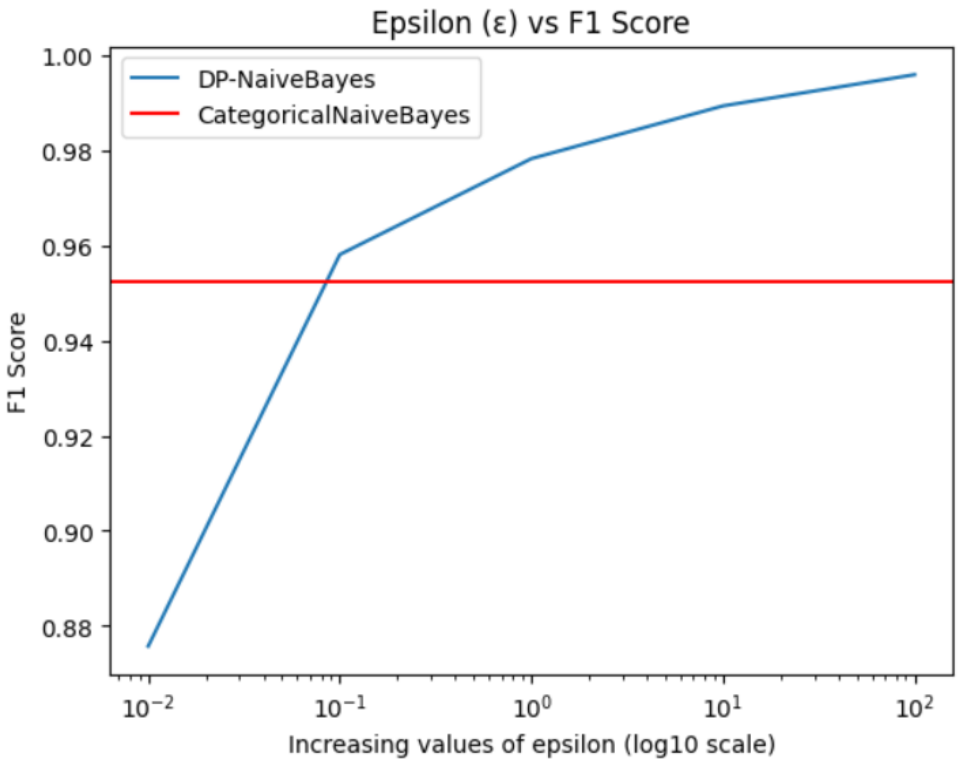


Fig 28. Epsilon (ε) vs F1 Score

From figure 25, 26, 27 and 28 we see the relationship between epsilon and accuracy, precision, recall and F1 scores, respectively. The scores tend to increase as epsilon increases and the DP-Naïve Bayes model performs better than the Categorical Naïve Bayes model's performance even at ε = 0.1.

Naïve Bayes (Numeric Attributes):

```
Performance evaluation of GaussianNB:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45

[[19  0  0]
 [ 0 12  1]
 [ 0  0 13]]

Accuracy Score:  0.9777777777777777
Precision Score:  0.9793650793650793
Recall Score:  0.9777777777777777
F1 Score:  0.9777448559670783
```

Fig 29. Performance evaluation of GaussianNB() from sklearn

Fig 29 shows the performance of GaussianNB() model from sklearn library on the iris dataset with three target classes.

```
Performance evaluation of 0.1-Differentially Private Gaussian Naive Bayes model:
              precision    recall  f1-score   support

           0       0.25      0.11      0.15        19
           1       0.24      0.31      0.27        13
           2       0.10      0.15      0.12        13

    accuracy                           0.18        45
   macro avg       0.20      0.19      0.18        45
weighted avg       0.20      0.18      0.17        45

[[ 2  6 11]
 [ 2  4  7]
 [ 4  7  2]]

Accuracy Score:  0.17777777777777778
Precision Score:  0.20241830065359478
Recall Score:  0.17777777777777778
F1 Score:  0.17460531238309018
```

Fig 30. Performance evaluation of DP-Naïve Bayes with ε = 0.1

```
Performance evaluation of 0.5-Differentially Private Gaussian Naive Bayes model:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.29      1.00      0.45        13
           2       0.00      0.00      0.00        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45

[[ 0 19  0]
 [ 0 13  0]
 [ 0 13  0]]

Accuracy Score:  0.28888888888888886
Precision Score:  0.08345679012345678
Recall Score:  0.28888888888888886
F1 Score:  0.12950191570881225
```

Fig 31. Performance evaluation of DP-Naïve Bayes with ε = 0.5

```
Performance evaluation of 1-Differentially Private Gaussian Naive Bayes model:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.00      0.00      0.00        13
           2       0.29      1.00      0.45        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45

[[ 0  0 19]
 [ 0  0 13]
 [ 0  0 13]]

Accuracy Score:  0.28888888888888886
Precision Score:  0.08345679012345678
Recall Score:  0.28888888888888886
F1 Score:  0.12950191570881225
```

Fig 32. Performance evaluation of DP-Naïve Bayes with ε = 1

```
Performance evaluation of 10-Differentially Private Gaussian Naive Bayes model:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.50      1.00      0.67        13
           2       0.00      0.00      0.00        13

    accuracy                           0.71        45
   macro avg       0.50      0.67      0.56        45
weighted avg       0.57      0.71      0.61        45

[[19  0  0]
 [ 0 13  0]
 [ 0 13  0]]

Accuracy Score:  0.7111111111111111
Precision Score:  0.5666666666666667
Recall Score:  0.7111111111111111
F1 Score:  0.6148148148148148
```

Fig 33. Performance evaluation of DP-Naïve Bayes with ε = 10

From Fig 30 to 33, we can observe the performance changes of DP-Naïve Bayes with increasing ε and as expected as ε increases the performance of the DP-Naïve Bayes model improves.

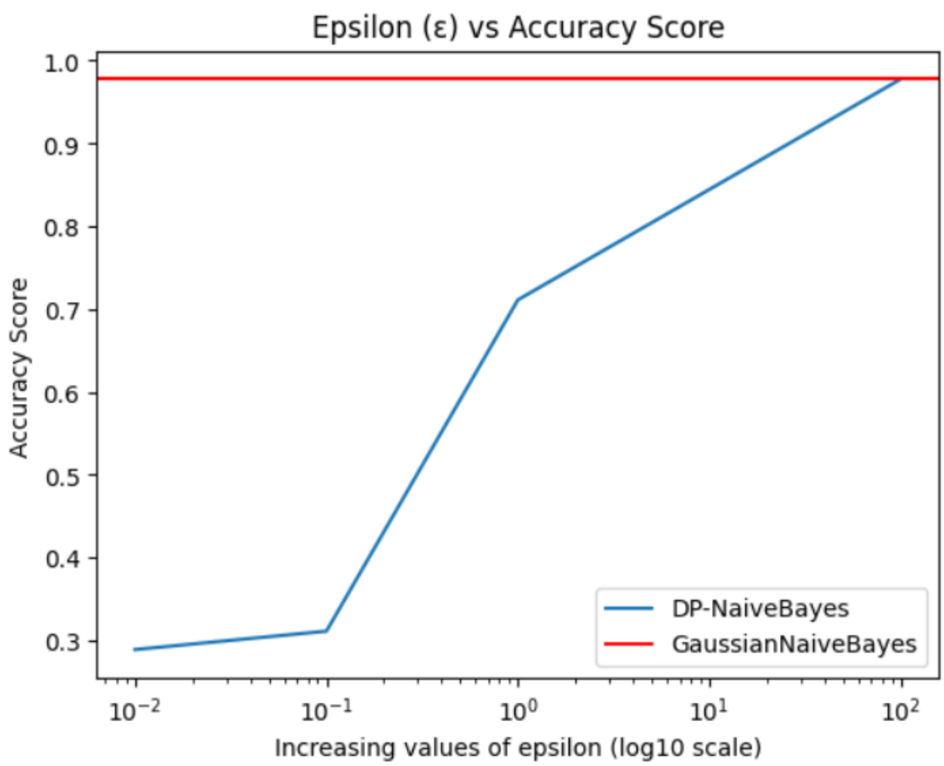


Fig 34. Epsilon (ε) vs Accuracy Score

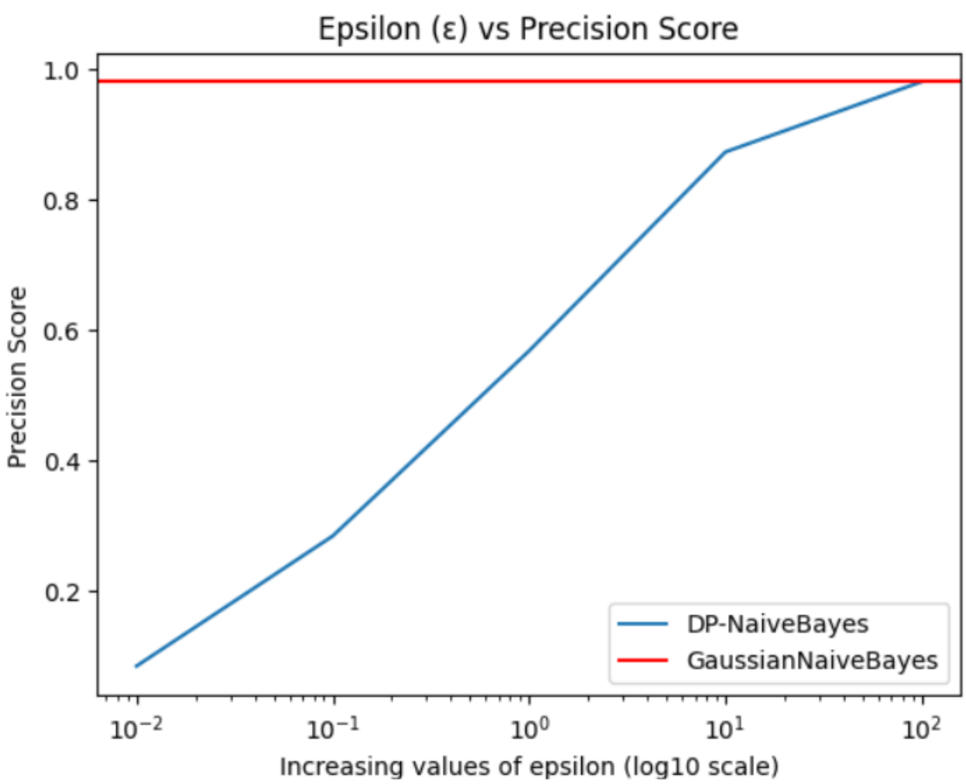


Fig 35. Epsilon (ε) vs Precision Score

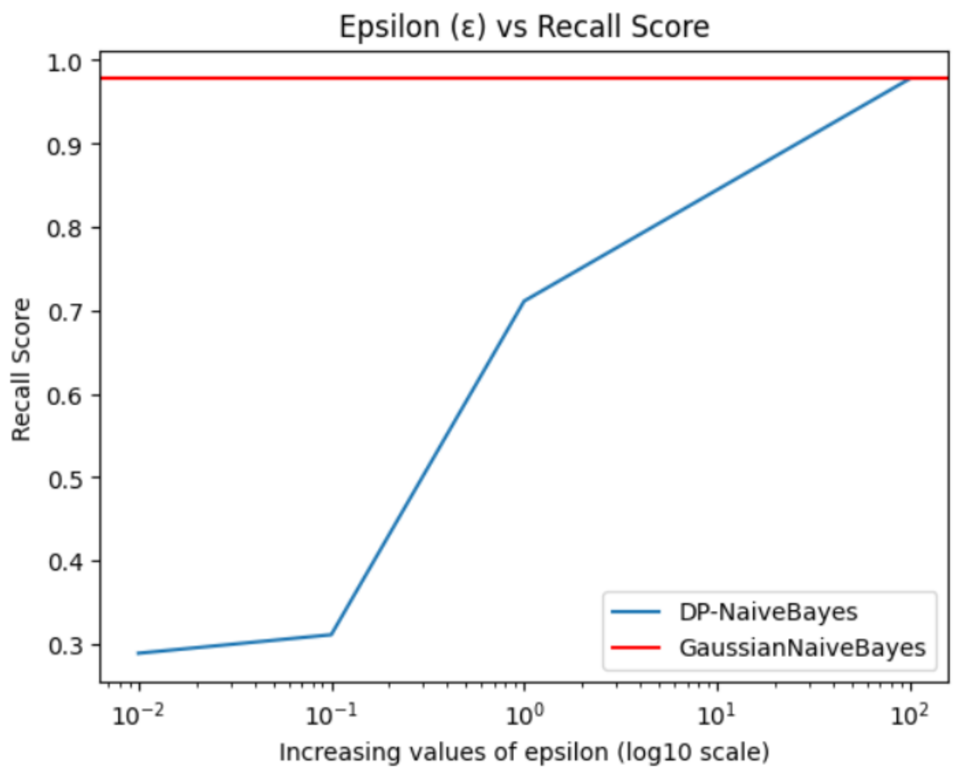


Fig 36. Epsilon (ε) vs Recall Score

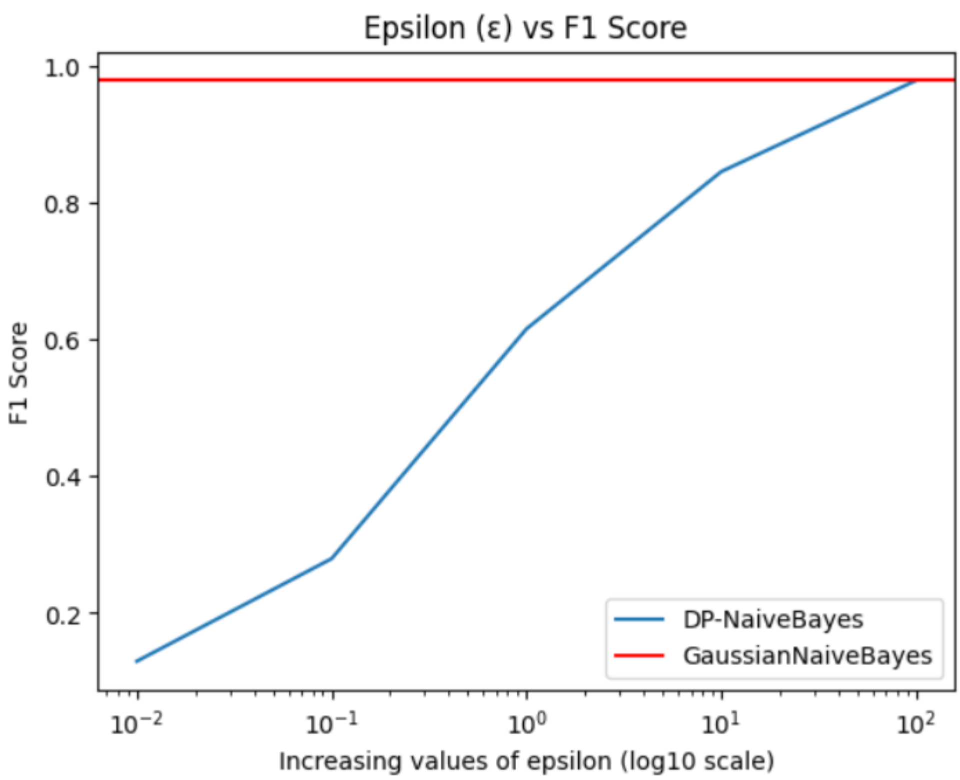


Fig 37. Epsilon (ε) vs F1 Score

From figure 34, 35, 36 and 37 we see the relationship between epsilon and accuracy, precision, recall and F1 scores, respectively. The scores tend to increase as epsilon increases and meet the standard Gaussian Naïve Bayes model's performance. But it is at a very high value of ε = 100 which doesn't make much sense as there isn't any privacy at the value.

<u>**Simple Linear Regression (SLR):**</u>

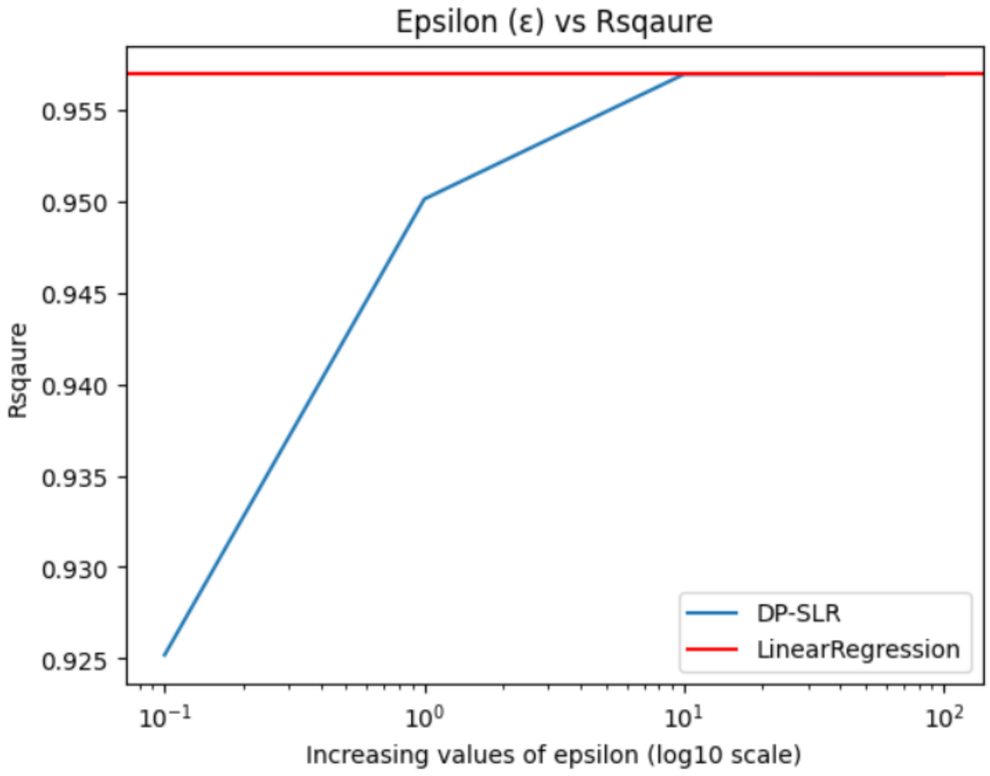


Fig 38. Epsilon (ε) vs R^2

Fig 38. depicts the relationship between the Epsilon (ε) and coefficient of determination (R^2), and it can be observed that R^2 increases as Epsilon (ε) increases and reaches pretty close to the LinearRegression model performance.

Using LinearRegression() from sklearn:
The regression line is y = 25792.200198668696 + 9449.962321455076 * X
Rsquare = 0.9569566641435086

Using 0.1-Differentially Private Simple Linear Regression (SLR):
The regression line is y = 42097.62948457824 + 9147.497802123875 * X
Rsquare = 0.6586036292697537

Using 0.5-Differentially Private Simple Linear Regression (SLR):
The regression line is y = 24362.748104864124 + 9530.011667693794 * X
Rsquare = 0.9555001589459717

Using 1-Differentially Private Simple Linear Regression (SLR):
The regression line is y = 26503.19815370011 + 9491.151798838231 * X
Rsquare = 0.9557483605472767

Using 10-Differentially Private Simple Linear Regression (SLR):
The regression line is y = 25746.957855041284 + 9462.660021527541 * X
Rsquare = 0.9569542564970671

From the above results obtained from the NoisyStats simple linear regression model, we can see that as ε increases from 0.1 to 10, the coefficient of determination increases as the Laplace noise injected becomes narrower leading to less privacy but more utility.

Conclusion

The project helped in gaining extra knowledge about the application of ε-differential privacy mechanism in the field of machine learning. With the ever-increasing data supply, it has become a requirement to extract meaningful insights from this data. But as the supply of data increases exponentially, so does the need to respect the privacy of the data owner and make sure that the security of the data is withheld, with its integrity, utility, and availability ensured. The project included both supervised machine learning algorithms like Naïve Bayes classifiers and simple linear regression and an unsupervised machine learning algorithm called K-Means clustering. In Naïve Bayes, the categorical and numerical attributes are to be handled differently. The DP-Naïve Bayes for the categorical attributes seemed to perform better than the Categorical Naïve Bayes model. The other observed was the performance improvement as the value of ε increases. DP-SLR also provided similar results to the Linear Regression model for a slightly higher value of ε. While applying ε-DP, it is crucial to understand what parameter has to be tempered with and with what quantity. To determine that, one must have knowledge of sensitivity, sequential and parallel compositions, and privacy budget allocation. These optimization problem, if solved correctly can help in maintaining the privacy of the individual in the dataset and thereby ensure that ε-differential privacy is satisfied and at the same time, the analysts and other concerned parties are able to extract valuable insights without compromising privacy and security of the data.

References

[1] College Student Data (https://www.kaggle.com/datasets/kuchhbhi/cpga-iq-placement)

[2] Mushroom. (1987). UCI ML Repository. https://doi.org/10.24432/C5959T.

[3] Fisher,R. (1988). Iris. UCI ML Repository. https://doi.org/10.24432/C56C76.

[4] salary_data of Employees with years of Experience (https://www.kaggle.com/datasets/harsh45/random-salary-data-of-employes-age-wise)

[5] J. Vaidya, B. Shafiq, A. Basu and Y. Hong, "Differentially Private Naive Bayes Classification," 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Atlanta, GA, USA, 2013, pp. 571-576, doi: 10.1109/WI-IAT.2013.80.

[6] Alabi Daniel, Audra McMillan, Jayshree Sarathy, Adam Smith, and Salil Vadhan. “Differentially private simple linear regression.” Proceedings on 23rd Privacy Enhancing Technologies Symposium (PoPETS ‘22), 2022, 2022 (2), 184–204.