Chapter 8
Data and Model Privacy

Introduction

In 2006, Netflix released 100 million anonymized movie ratings data (containing unique subscriber ID, movie title, release year and date of rating) for an online competition to build an algorithm that would predict movie rating by a subscriber. Within 2 weeks of the data being released, two researchers Arvind Narayanan and Vitaly Shmatikov were able to reverse-engineer the so-called anonymized data and identify the subscriber by matching it with other similar data sources like IMDB. So much so, they not only identified the viewing history of the users, but also their apparent political preferences and other potentially sensitive and personal information.

Before we look at different ways to protect private or personal data, let's spend a moment to understand what private data is. GDPR defines personal data (personally identifiable information or PII) as any information relating to an identified or identifiable natural person ("data subject"); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.

This includes information shared in the social media (text, images, videos, audios), locations using the user's phone, user's device usage patterns, shopping patterns, transaction behaviours, financial information and other personal information like health records, relationship information or even near future plans.

Few years ago, in 2000, computer scientist Latanya Sweeney combined the anonymized hospital data released by the Massachusetts Group Insurance Commission with the 1990 US census records and other information in public domain to identify the health records of the Governor of Massachusetts. Sweeney estimated that 87% of the US population can be identified uniquely using just three

S. Agarwal, S. Mishra, *Responsible AI*,
https://doi.org/10.1007/978-3-030-76860-7_8

parameters: five-digit zip code, gender and date of birth. This is known as linkage attack where anonymized data is combined with other public datasets using common attributes between the datasets to identify individual information. Once a common approach and considered good enough, removing the basic customer identifiers or obfuscating them with hash keys is now known to be a weak approach that does not really protect the private data on its own. The approaches we will see to protect data in this chapter will aim to make data safe from individual identification while keeping it useful for any statistical analysis.

The risks posed by the data that does not truly protect private data are not limited to potential hacks where the data is accessed for nefarious purposes. As we build machine learning models using this data, it can cause biases to come in or for the model to learn relationships dependent on sensitive features. Finally, there is also the risk of private information being available to your teams working on the data and resulting liabilities.

To mitigate these risks, the approach should be to enhance the privacy of the data as soon as you can in your data pipeline. If the data is made private before it gets into your datastore, any downstream consumption will not be able to access the private information. As we will see soon, adding privacy to the data also reduces bias from the model by marking information that is used during the modelling. This in turn impacts the fairness treatment and the explainability and monitoring too. For example, if a model is trained using geospatial data, health records or demographic data, there is a high probability of bias being present in the data. Adding privacy, at different levels, can help us prevent ourselves from one or more of these challenges.

In addition to the above risks, lack of privacy in the data or the model also leaves your model susceptible to adversarial attacks. The adversarial attacks are classified as white box (where the attacker has the model parameters), black box attacks (where the attacker does not know the model parameters or the architecture) or grey box (somewhere in between the two). In both cases, the objective of the attack can be to:

1. Identify and leak data about the users or model parameters
2. Cause misclassification of the target observation
3. Reduce the model accuracy

We will see later in this chapter that adding privacy to your model gives a reliable protection against such attacks by adding noise to the information.

Note: Adversarial Attacks on ML Models

Linear models are very sensitive to the training data. A change in the training data can skew the model parameters, and this vulnerability can be exploited by an attacker to determine if a record was part of the training data. Similarly, KNN models carry a lot of training information and can be exploited by the attacker if the data does not have privacy built into it.

Typically, in a white box attack, an attacker would aim to dilute the model by disturbing the model parameters and thus forcing the model to give wrong outcomes. While in case of black box attack, attacker sends data (assumed to be from the same distribution) to the target model and collects the output which is then used to create a shadow model which mimics the target model. This kind of attack is feasible when attacker has access to the target model mostly possible during transfer learning or models in the cloud.

In a black box attack, training data is used to find the model parameters, while a white box attack can reveal the training data. For example, using some information about the patient, a model trained on medical data can allow an attacker to access confidential information (model inversion) about patient health conditions. Similarly, membership of an individual in training data can be easily found by sending the data to the model and checking model's prediction confidence. The model's prediction (membership inference) would be more confident if the person was part of the training sample, as compared to when they were not. This is very true in case of model that is used for predicting say credit score by a bank, readmission probability by a hospital or survival probability of cancer patient of a locality.

Basic Techniques

Let's begin with the basic approaches to masking the data. As we saw in the examples earlier, they don't offer protection on their own but can serve as a good data hygiene when you have data with personal information. This can be useful if you need to share your data, especially when used in addition to the techniques discussed later in the chapter.

Hashing

A one-way hashing function is a common way to convert sensitive fields into a fixed length unique hash to protect the data from being linked to a specific user. To avoid hashing clash, a unique salt string can be added to the value to create a unique hash, which is then stored. While this creates one-way unique values, there are several challenges with using hashing for masking sensitive fields.

Hashing is useful when you want to convert a value into a unique value, often implemented as a one-way mechanism to securely store passwords. Most of the problems where you will employ machine learning to solve it will involve high-dimensional data where it is already very unique. This limits the value that hashing is going to provide. The other challenge with hashing is that even though it will mask the fields that are being hashed, it cannot mask the relationships that the unmasked data can reveal. As we go along in the book, we will see how those relationships can reveal biases, and handling them correctly plays an important part in creating a responsible AI model.

Finally, once a field has been hashed, it cannot be used for any exploratory analysis or model training. This restricts the use of the technique to the fields that are unique identifiers of a person and would not have been included in the analysis or the training anyway. An example of such a field is customer ID which is needed as a unique identifier to tell the records separately but using the original value can reveal the individual's identity. Beyond the masking of ID values, we do not recommend using hashing.

K-Anonymity, L-Diversity and T-Closeness

There are plenty of techniques for data anonymization. A few of them are simple masking, data aggregation, data grouping or converting numeric data to categories. However, most of these methods are quite weak and lead to a lot of information loss. These techniques also don't prevent attackers from reverse engineering to find out the sensitive information. Let's quickly go through some techniques that are stronger and don't require us to lose a lot of data.

k-anonymity is a technique to ensure that the information of a single person in the data would be similar to k-1 individual in the dataset. For instance, if the attacker would like to identify an individual using some of their private data like postcode, marital status and age, then they would end up finding K individuals with same combination of protected features. It would further generalize some of features (e.g. convert all date of birth between 1980 and 1989 to 1980) or use suppression for anonymity like reducing a six-digit postcode to first three digits followed by '*" (CV1*, WG1*, etc.). However, if the attacker is able to access multiple datasets with different k-anonymity, they can use that to identify the missing data and ultimately can reveal information that is private.

l-diversity builds upon k-anonymity and creates intra-group diversity for sensitive features. However, if the attacker has the knowledge of the distribution of the original data, l-diversity can be decoded to find out original information. Building on the drawback of l-diversity, t-closeness is computed as the distance between the sensitive attribute of a class and the complete data set which is less than or equals to t. The distance between both the distribution is computed using Wasserstein distance (as seen in the previous chapter).

Differential Privacy

Let's consider an example – you are working on using the data from calls to the emergency services to understand if there are any local trends that may require attention. The data contains the information that the callers provided to the emergency dispatcher (including name, address, age, symptoms) and the diagnosis and the details of the subsequent treatment given. Removing fields from the dataset is not only going to increase privacy – just the street name and the age might identify the caller uniquely – but will also render the data unusable with a lot of missing information.

Differential privacy helps us overcome both these challenges. Put simply, differential privacy works by introducing noise to the data so that it does not really significantly change the distribution, keeping the data usable for any analysis and model training, but distorting it enough to prevent specific individuals to be identified. For our example above, changing some of the fields in the dataset, say the age and the street name, to values in the neighbourhood of the original values, will make it much more difficult, if not impossible, to identify the caller and also allows us to keep all the data, albeit with some noise, available for analysis.

In this section and the subsequent ones in this chapter, we will look at some of the ways to introduce calibrated noise to the data that will allow us to produce very accurate statistics while keeping the data private.

The best thing about differential privacy is its flexibility. Adding noise to the data won't make it easily identifiable, reduce the number of rows or lead to major loss of information. In case of differential privacy, a small amount of noise is added to the required data in order to mask the information and prevent any individual identity to be revealed. This noisy input is then used for any future analysis or ML training.

Let's look at the maths involved in the differential privacy. A randomized algorithm f provides (ε, δ)-differential privacy if, for all neighbouring databases D_1 and D_2, and for any set of outputs S

$$P\left[f\left(D_1\right) \in S\right] \le e^{\varepsilon} \times P\left[f\left(D_2\right) \in S\right] + \delta$$

where, f is any function

e^{ε} is very close to 1,with $\varepsilon > 0$
large e^{ε} gurantees no privacy
ε (epsilon) is a privacy parameter
or the privacy budget (lower ε, stronger privacy)
δ is failure probability or tolerance.
If $\delta = 0$, then we have (ε, 0) differential privacy

The notion of the sensitivity of a function is central to the design of differentially private algorithms. Given a function f on a dataset D_1, the sensitivity is used to adjust the amount of noise required for $f(D_1)$. If f is a function that maps a dataset (in a matrix form) into a fixed-size vector of real numbers, we can define the generic sensitivity of f as

$$\text{Sensitivity} = \delta\left(f\right) = \max_{D_1,\ D_0} \| f\left(D_1\right) - f\left(D_2\right) \|_i$$

Where $\|.\|_i$ denotes to l_i norm with $i \in \{1, 2\}$
D_1, D_2 are two neighbouring dataset

Sensitivity is the maximum change in output with a change in one record between D_1 and D_2. Alternatively, it is the maximum difference one row can have as we apply the differential privacy algorithm. The noise added to the D_1 to make it D_2 can be any noise of choice be it Gaussian or Laplace. In case of the function being a machine learning model, D_1 and D_2 are two datasets that differ by one training sample, and the sensitivity would be the maximum change in the output when one training sample is removed from the training set. The noise added in D_1 can be computed using the sensitivity and the privacy budget on either Laplace or Gaussian distribution.

Using Laplace:

$$\text{Private}\mathbf{R} = f\left(D_1\right) + lap\left(\frac{\delta f}{\varepsilon}\right)$$

Where δf denotes the sensitivity of f and Lap ($\frac{\delta f}{\varepsilon}$) represents the noise drawn from the Laplace distribution with the centre of 0 and the scaling of $\frac{\delta f}{\varepsilon}$

Using Gaussian:

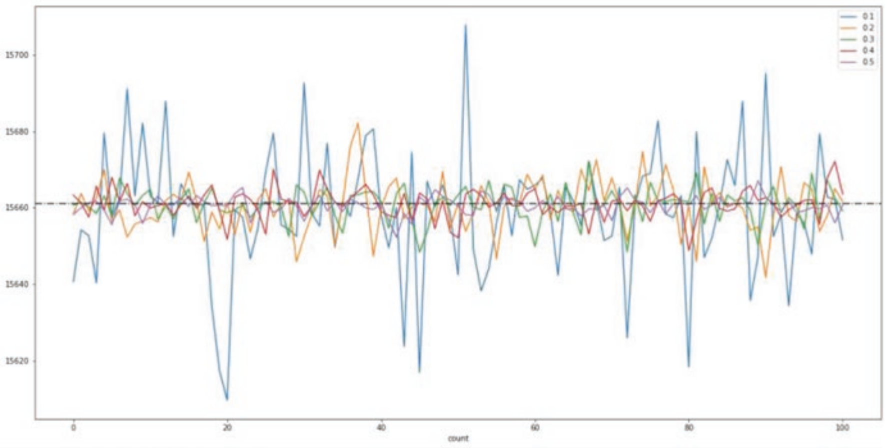
$$\text{Private}\mathbf{R} = f\left(D_1\right) + N\left(0, \sigma^2\right)$$

Where $N(0, \sigma^2)$ indicates that the noise variable is independently and identically distributed Gaussian distribution with the standard deviation $\sigma = \delta_2\left(f\right) \times \frac{\sqrt{2\log\frac{1.25}{\delta}}}{\varepsilon}$, where $\delta_2(f) = \|f(D_1) - f(D_2)\|_i$. Unlike k-anonymity, DP is data-agnostic and depends on privacy budget and the distribution being used. As long as the differentially private algorithm is being used, the resulting data will be differentially private too.

As discussed above, ε (epsilon) is the privacy budget which decided the amount of privacy in the algorithm. A small value would ensure high privacy, while high value of ε provides less privacy (you can afford to give away more of the privacy when you have more budget). It is advisable to have ε (epsilon) less than or equal to 1.

Let's consider couple of examples for different query functions and the impact of different values of ε on the value generated after the DP treatment. In the code example below, we are adding Laplace noise to a query fetching number of instances where the Applied Amount is greater than 2300. The subsequent plot shows the generated values across the 100 loops. The dotted black line is the actual value, and the different colours correspond to the value of ϵ. From the chart (Fig. 8.1), we can see that the privacy varies as we vary the privacy parameter. However, for a given epsilon, the value returned is very close to actual value, thus preserving differential privacy.

```
sensitivity = 1
#epsilon = 0.1

original = X_train[X_train['AppliedAmount'] >= 2300].shape[0]

V=[]
ep=[]
count=[]

for j in range (1,6):
    epsilon = j/10

    for i in range(0, 101):

        value = X_train[X_train['AppliedAmount'] >= 2300].shape[0] + \
        np.random.laplace(loc=0, scale=sensitivity/epsilon)
        V.append(value)
        ep.append(epsilon)
        count.append(i)
```

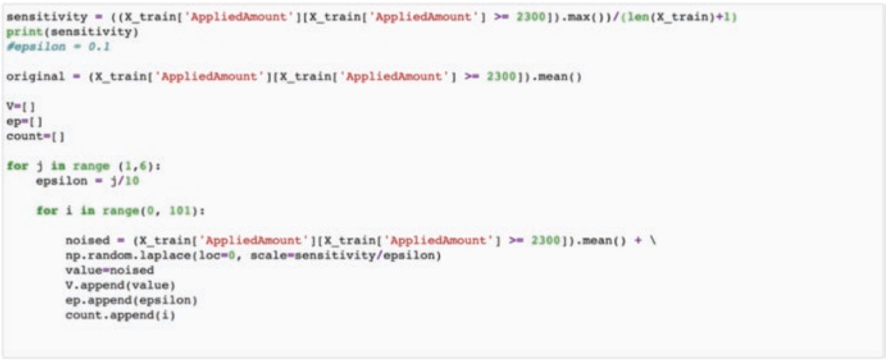


```
fra.groupby('ep')['Value'].mean()/original
```

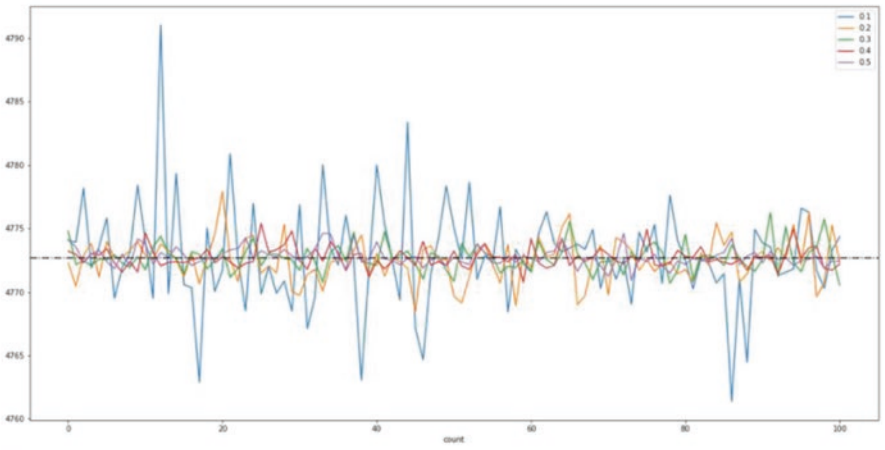
```
ep
0.1     0.999873
0.2     0.999975
0.3     0.999984
0.4     1.000008
0.5     0.999973
Name: Value, dtype: float64
```

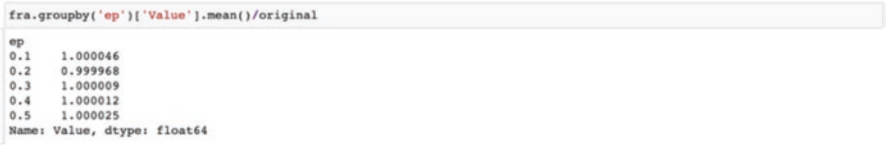
Fig. 8.1 The chart shows how the ε affects the values generated

In another example (Fig. 8.2), we have a function to compute the mean of Applied Amount (for Applied Amount greater than 2300). Here the sensitivity is calculated as a maximum difference if one record is added with maximum possible Applied Amount. The code adds return the mean value which is differentially private.



```
sensitivity = ((X_train['AppliedAmount'][X_train['AppliedAmount'] >= 2300]).max())/(len(X_train)+1)
print(sensitivity)
#epsilon = 0.1

original = (X_train['AppliedAmount'][X_train['AppliedAmount'] >= 2300]).mean()

V=[]
ep=[]
count=[]

for j in range (1,6):
    epsilon = j/10

    for i in range(0, 101):

        noised = (X_train['AppliedAmount'][X_train['AppliedAmount'] >= 2300]).mean() + \
        np.random.laplace(loc=0, scale=sensitivity/epsilon)
        value=noised
        V.append(value)
        ep.append(epsilon)
        count.append(i)
```





```
fra.groupby('ep')['Value'].mean()/original
```

```
ep
0.1    1.000046
0.2    0.999968
0.3    1.000009
0.4    1.000012
0.5    1.000025
Name: Value, dtype: float64
```

Fig. 8.2 The output reveals that in most of the cases the noise outcome is very close to the original mean with epsilon value of 0.3 returning the most closest value

Privacy Using Exponential Mechanism

The previous section shows how the data can be made differentially private by adding noise. However, adding noise to the data that is non-numeric in nature can give some really weird results. For instance, adding noise to a zip code can result in a value that may not exist or adding noise to an integer that represent a category may not work too. Also, it won't work for binary or Boolean data type and is not suitable for features that are categorical in nature like colour of a car, gender, marital status and so on. For these cases, a slightly different mechanism needs to be adopted to introduce noise.

Exponential mechanism (EM) aims at adding noise to a function that is dealing with non-numeric features. The technique we will cover in this section will add noise, to satisfy privacy, by sometimes returning a false value. It will, however, always return a value that is part of the dataset and not a value outside of the data universe on which the EM is implemented. This helps improve privacy and also eliminates the chances of having invalid values in the dataset.

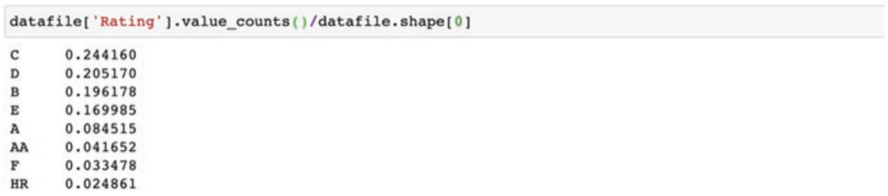
It would select the element from a set of elements given a scoring function. The scoring function can be used to find K-top value (or K-bottom) for the field from the data, like most common zip code, the least preferred colour of a car, gender of people who are employed for more than 10 years in a company or marital status of patients in a given hospital. In order to satisfy privacy, exponential mechanism would sometime return a false number. For instance, it can return a colour of a car that is most preferred rather than least preferred.

Consider a categorical data (x) with values in the range (R) with a scoring function (u). The sensitivity of the scoring function thus would be

$$\delta u = \max \| u(x,r) - u(y,r) \|$$

EM ($M_E(x, u, R)$) would return a value $r \in R$ with probability proportional to $\exp \frac{(\varepsilon u(x,r))}{\delta u}$. It can be also used in conjunction with differential privacy by adding Laplace noise ($\frac{\delta u}{\epsilon}$) to the score function (u) and returning the value that has maximum noise but falls in the dataset.

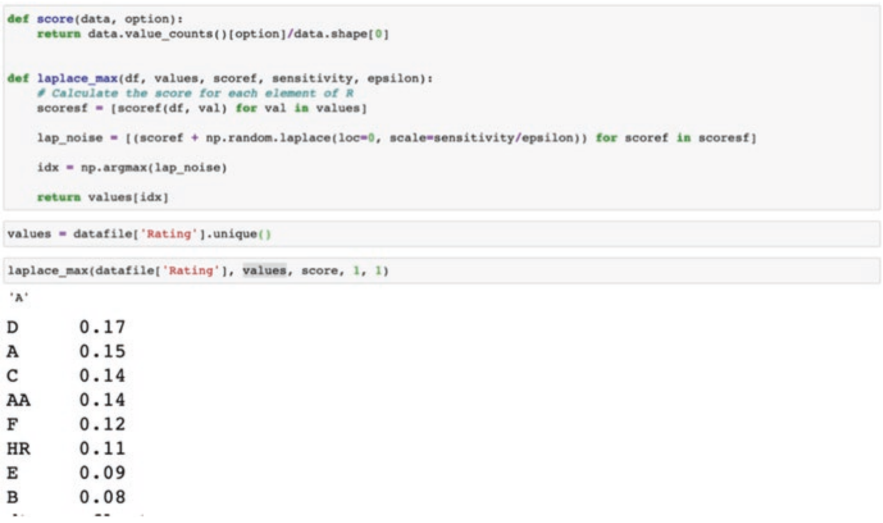
For instance, in the dataset, the rating variable has eight categories with rating C and D having maximum frequency while F and HR having least frequency.



```
datafile['Rating'].value_counts()/datafile.shape[0]
```

```
C      0.244160
D      0.205170
B      0.196178
E      0.169985
A      0.084515
AA     0.041652
F      0.033478
HR     0.024861
```

The below code aims at returning one category out of the total of eight for the rating feature that has the highest frequency. The code declares the score function followed by a function that is used to return differentially private outcome by adding Laplace noise to the outcome of the Score function. Here the Score function is about returning the proportion of the categories in a given feature; here, for instance, it would return the proportion of the different ratings from the rating variable of the dataset followed by Laplace max function to return the rating categories with highest proportion in data albeit with noise.



```
def score(data, option):
    return data.value_counts()[option]/data.shape[0]


def laplace_max(df, values, scoref, sensitivity, epsilon):
    # Calculate the score for each element of R
    scoresf = [scoref(df, val) for val in values]

    lap_noise = [(scoref + np.random.laplace(loc=0, scale=sensitivity/epsilon)) for scoref in scoresf]

    idx = np.argmax(lap_noise)

    return values[idx]
```

```
values = datafile['Rating'].unique()
```

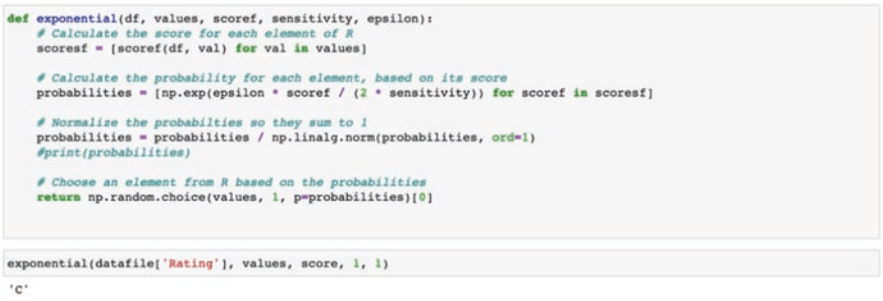
```
laplace_max(datafile['Rating'], values, score, 1, 1)
```

```
'A'
```

```
D      0.17
A      0.15
C      0.14
AA     0.14
F      0.12
HR     0.11
E      0.09
B      0.08
```

The above code returns A if executed once but would return D if iterated for infinite times.

Similarly, the below set of code used EM method probability function and returns C as the value with highest frequency.



```
def exponential(df, values, scoref, sensitivity, epsilon):
    # Calculate the score for each element of R
    scoresf = [scoref(df, val) for val in values]

    # Calculate the probability for each element, based on its score
    probabilities = [np.exp(epsilon * scoref / (2 * sensitivity)) for scoref in scoresf]

    # Normalize the probabilties so they sum to 1
    probabilities = probabilities / np.linalg.norm(probabilities, ord=1)
    #print(probabilities)

    # Choose an element from R based on the probabilities
    return np.random.choice(values, 1, p=probabilities)[0]
```

```
exponential(datafile['Rating'], values, score, 1, 1)
```

```
'C'
```

The choice of the method largely depends on the size of the range. In case the returned value needs to be from finite set of data (credit rating), EM would be advisable, but in case the range is infinite, or the labels are numerically coded (e.g. name of a person or post code), EM with Laplace can be opted.

Differentially Private ML Algorithms

So far, we have looked at the ways to add differential privacy to the data. Adding calibrated noise to the data allows us to then use any algorithm we want to train the model and still achieve the data privacy. However, the concept of differential privacy can be extended beyond adding the noise to the data.

In this section, we will see how we can use similar approach to add differential privacy to the model, by adding noise to the model's objective function, or to the model output, by adding noise to the weights of the output of the model's objective function.

Privacy can be introduced at input data (using noise or any other techniques), add noise to the objective function that satisfy privacy or add noise derived from Laplace or Gaussian distribution to the weights of the output of the objective function. Adding noise to the objective or the weights of the output has a similar effect as adding noise to the training data, with an added protection against any attack on the model.

Here's a standard form of classification objective function.

$$J(W) = argmin\ \frac{1}{n} \sum_{i=1}^{n} l\left(W, \left(x_i, y_i\right)\right)$$

For input perturbation (adding noise to the input data), the objective functions become

$$J(W) = argmin\ \frac{1}{n} \sum_{i=1}^{n} l\left(W, \left(x_i + z, y_i\right)\right)$$

In case of output perturbation (adding noise to the output weights), the objective functions become

$$J(W) + (\mathrm{W}, Z)$$

where *Z* is noise derived from Laplace or Gaussian distribution and has the same dimensions as *W*. The noise gets added to the weights after the training is complete to make the output differentially private.

However, when we want to add the privacy to the objective function, we skew the function by adding noise (derived from Laplace distribution) to coefficient of the model. In case of tree-based algorithm, noise would be added to information gain parameter.

Let's consider dataset D = $\{(x_i, y_i)\}$ and objective function.

$$J(W) = argmin\ \frac{1}{n} \sum_{i=1}^{n} l\left(W, \left(x_i, y_i\right)\right)$$

The loss function after privacy parameter has been added in the objective function will be

$$J\left(W\right) = argmin\ \frac{1}{n}\sum_{i=1}^{n} l\left(W,\ \left(x_i,\ y_i\right)\right) + \left(W \times Z\right)$$

Where Z is the noise vector

As discussed above, the noise added depends on the sensitivity and privacy parameter. Since we are updating the objective function, implementing these techniques would mean writing the algorithm from scratch or use a privacy-preserving version of an algorithm.

Another way to do this is by adding noise to the gradient by optimizing SGD with noisy gradient. SGD processes one data point at a time with incremental gradient steps. For this example, we will assume data D = $\{(x_i, y_i)\}$, with n records with $y_i \in \{0, 1\}$. Given an objective function, SGD can be used to find a separating hyperplane. Here SGD can be defined as

$$W_{t+1} = W_t - \eta\left(\delta l\left(y_t, W_t, x_t\right) + \lambda W_t\right)$$

where η is the learning rate and gradient is computed on single instance (y_t, x_t)
In case of SGD mini batch, the weights would be computed as

$$W_{t+1} = W_t - \eta\left(\frac{1}{b}\Sigma\delta l\left(y_t, W_t, x_t\right) + \lambda W_t\right)$$

where b is the size of the mini – batch

By adding a noise parameter, SGD can be made differentially private, and thus the weights would be calculated as

$$W_{t+1} = W_t - \eta\left(\delta l\left(y_t, W_t, x_t\right) + \lambda W_t + Z_t\right)$$

Where Z is the noise vector

And in case of mini-batch SGD, the equation would slightly change to

$$W_{t+1} = W_t - \eta\left(\frac{1}{b}\Sigma\delta l\left(y_t, W_t, x_t\right) + \lambda W_t + \frac{1}{b}Z_t\right)$$

As noise is being added at each iteration of gradient decent, each data point would have different value (parameter of differential privacy) of differential privacy. In case of SGD or mini-batch SGD, privacy implementation would be dependent on batch size and learning rate.