ILLINOIS TECH

Trustworthy Machine Learning: Poisoning Attacks

Due: 2nd December 2024
CS 484: Final Project Report
Professor Name: Dr. Binghui Wang
Group Members: Luke Crosby, Clarence Fernandes, Pranav Kuchibhotla

Introduction/Background:

As Machine Learning (ML) becomes more popular and mainstream, the risk of security attacks is becoming more apparent. Many industries use ML on a daily basis, and are at risk to these types of attacks. One type is poisoning attacks, which are a major issue in some industries. They involve someone with access to a model or data putting in fake data, which can cause model performance to decrease drastically. In the medical field this can raise concern as ML models are sometimes used to diagnose patients. If the model mis-diagnoses a patient due to data poisoning, then there is a potential health risk to that patient. Another, more broad, example is in the case of Federated Learning (FL), which is a system in which a model is centralized and users anonymously train data and the model is updated from each user's training. Since the training is done anonymously though, a nefarious user could try to mess with the model by training on bad data, which would make other user's performances decrease.

Related Work- Defense Methods:

There has been some research done on defense methods, and some of the generally accepted methods to keep a model safe is to have robust data validation and provenance tracking. In the case of FL, one proposed method is user elimination, which seeks to check a user's results before updating the model and removing those users/results that seem nefarious. So we Explored Federated Learning (FL) and Data Poisoning Attacks where malicious users intentionally alter their local data to degrade or manipulate the global model's performance. So even a relatively small proportion of malicious users (over 20%) can subtly alter model predictions without significantly affecting overall accuracy, making these attacks difficult to detect using traditional metrics like accuracy or loss [1].

Key Findings: One of the key observations was that ***malicious users tend to report higher loss values*** during their local training, which can serve as a distinguishing feature for detecting adversarial behavior. This insight led to the development of a defense mechanism that monitors user-reported loss values while preserving privacy.

Defense Mechanism: then we propose the novel defense approach that combines ***user loss monitoring*** with ***Local Differential Privacy (LDP)***. By leveraging metadata (i.e., noisy training loss values), the method uses ***K-Means clustering*** to identify and exclude malicious users from contributing to the global model. This approach maintains FL's privacy guarantees while effectively mitigating data poisoning attacks.

Results: The defense mechanism was tested on two widely-used datasets, ***MNIST*** and ***CIFAR-10***, and demonstrated robust performance in defending against data poisoning attacks. The results showed that the method was effective in both preserving model integrity and eliminating attackers without compromising user privacy.

Method/Solution:

To show the danger of data poisoning, the k-means clustering algorithm is used with varying the amount of poisoned data along with the variance of the poisoned data. The original data is obtained by using the *make_blobs* tool from the scikit learn dataset. 3 clusters are made of 500 data points, and they are clustered significantly far apart so the clusters can easily be solved. To poison the data a certain number of data points are randomly selected, ranging from 0 to 250. This will provide at most 50% poisoned data, which is more than enough to show how it affects the result. After the data points are selected, they are randomly increased via

numpy.random.normal which changes the values of the datapoint following a Gaussian distribution. The variance of the distribution is varied from 0 to 9.

After the poisoned data is created, a simple k-means algorithm involving initializing, updating, and predicting the data is performed. The initial centers are randomly chosen, and clusters are assigned based on the center that each data point is closest to. Then to update the clusters, the average distance from each point is used to update where the center of the cluster is. Lastly, the prediction function takes the distances of the remaining points not assigned, and assigns them to the cluster they are closest to. This process is repeated for every change to the poisoned data, and the silhouette score is used to assess the performance. Silhouette score is a metric commonly used to assess clustering models. It determines if the points assigned to a cluster are assigned to the correct cluster using the equation:

$$s = \frac{b-a}{max(a,b)}$$

If a score is low, that indicates that the point might not be assigned to the correct cluster. The score ranges between 1 and -1, where "a value close to 1 indicates a well-clustered data point, a value close to 0 suggests overlapping clusters, and a value close to -1 indicates a misclassified data point [2]." This equation basically determines how similar a point in one cluster is to the points in another cluster. The accuracy score was also explored. This is a metric that determines how well a model does by comparing the predicted labels to the actual labels.

Results:

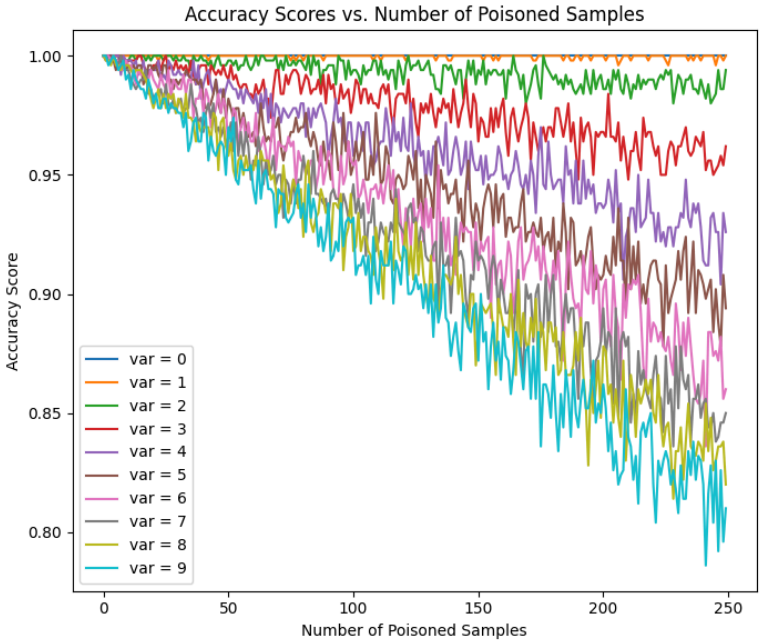

Figure 1: Plot displaying accuracy scores as percentage of poisoned data increases, and variance of poisoned data increases

Figure 1 above shows that when the amount of poisoned data increases, the accuracy is not affected significantly. The accuracy does not drop below 90% until the variance of the poisoned data reaches 5, and this occurs when nearly 50% of the data is poisoned. Therefore, the percentage of poisoned data is not a significant factor in the accuracy of a model, and even when high variance is used the model still has an accuracy above 80%.

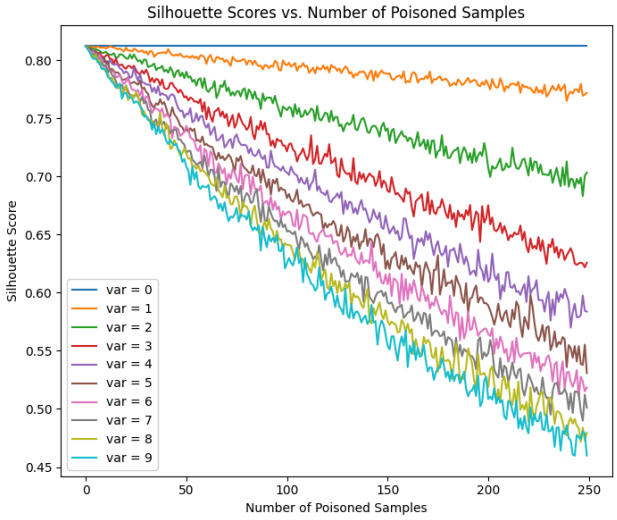

Figure 2: Plot displaying silhouette scores as variance and percentage of poisoned data increases

Figure 2 above shows that poisoned data can still be detected using metrics like the silhouette score. This metric is specific for clustering, and low values signify that there are overlapping clusters. When there is little variance, the silhouette scores remain relatively high. However, they quickly fall off as the variance increases, signifying that the model performed poorly with high variance and poisoned samples.

Conclusion:

The results presented above show the danger of data poisoning. As the percentage of poisoned data increased, the silhouette score decreased drastically. Variance of the poisoned data also played a role, however the large distances between lines at the lower variances show that the amount of poisoned data was a more significant factor. It is likely in real poisoning attacks the data would not be severely changed (high variances) since attackers would want to remain undetected. Therefore, it is important to look at changes when the variance and percentage of poisoned data is low. Figure 2 again shows then that data poisoning can severely affect clustering

models, as the silhouette score drops to below 0.70 when the variance is just 4, and the percentage of poisoned data is about 30%.

Overall, the importance of keeping models and data safe was explored. This was done with a focus on data poisoning as an attack, and some defense methods were examined before showing the effect data poisoning can have. It is clear that data poisoning is a serious issue, and models need to have better security or more robust training to stay safe from security attacks.

References:

1. Galanis, N. (2024). Defending against Data Poisoning Attacks in Federated Learning via User Elimination. *arXiv preprint arXiv:2404.12778*.
2. Silhouette Coefficient Explained with a Practical Example: Assessing Cluster Fit" | by Suraj Yadav | Medium

Appendix:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
import random
```

```
X,y = make_blobs(n_samples = 500,n_features = 2,centers = 3,random_state = 23)

fig = plt.figure(0)
plt.grid(True)
plt.scatter(X[:,0],X[:,1])
plt.show()
```

```
k = 3

clusters = {}
```

```
np.random.seed(23)

for idx in range(k):
    center = 2*(2*np.random.random((X.shape[1],))-1)
    points = []
    cluster = {
        'center' : center,
        'points' : []
    }

    clusters[idx] = cluster

clusters
```

```
plt.scatter(X[:,0],X[:,1])
plt.grid(True)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '*',c = 'red')
plt.show()
```

```
def distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))
```

```
#Implementing E step
def assign_clusters(X, clusters):
    for idx in range(X.shape[0]):
        dist = []

        curr_x = X[idx]

        for i in range(k):
            dis = distance(curr_x,clusters[i]['center'])
            dist.append(dis)
        curr_cluster = np.argmin(dist)
        clusters[curr_cluster]['points'].append(curr_x)
    return clusters
```

```
#Implementing the M-Step
def update_clusters(X, clusters):
    for i in range(k):
        points = np.array(clusters[i]['points'])
        if points.shape[0] > 0:
            new_center = points.mean(axis =0)
            clusters[i]['center'] = new_center

            clusters[i]['points'] = []
    return clusters
```

```
def pred_cluster(X, clusters):
    pred = []
    for i in range(X.shape[0]):
        dist = []
        for j in range(k):
            dist.append(distance(X[i],clusters[j]['center']))
        pred.append(np.argmin(dist))
    return pred
```

```
clusters = assign_clusters(X,clusters)
clusters = update_clusters(X,clusters)
pred_true = pred_cluster(X,clusters)
```

```
plt.scatter(X[:,0],X[:,1],c = pred_true)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```

```
n_samples = 500
X,y = make_blobs(n_samples = n_samples,n_features = 2,centers =
3,random_state = 23)

n_poisoned_samples = 50
nums = list(range(1, n_samples))
random.shuffle(nums)
```

```
X_poisoned = X.copy()
for i in range(n_poisoned_samples):
  poisoned_point = nums[i]
  X_poisoned[poisoned_point] += np.random.normal(loc=0, scale=5.0,
size=X[poisoned_point].shape)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(X_poisoned[:, 0], X_poisoned[:, 1], color='red', alpha=0.7)
plt.title('Poisoned Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.subplot(1, 2, 2)
plt.scatter(X[:, 0], X[:, 1], color='blue', alpha=0.7)
plt.title('Original Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()
```

```
silhouette_scores_by_var = {}
accuracy_scores_by_var = {}
for var in range(10):
  silhouette_scores_by_var[var] = []
  accuracy_scores_by_var[var] = []
  for j in range(250):
    n_samples = 500
    X,y = make_blobs(n_samples = n_samples,n_features = 2,centers =
3,random_state = 23)

    n_poisoned_samples = j
    nums = list(range(1, n_samples))
    random.shuffle(nums)

    X_poisoned = X.copy()
```

```
    for i in range(n_poisoned_samples):
      poisoned_point = nums[i]
      X_poisoned[poisoned_point] += np.random.normal(loc=0, scale=var,
size=X[poisoned_point].shape)


    k = 3

    clusters = {}
    np.random.seed(23)

    for idx in range(k):
        center = 2*(2*np.random.random((X_poisoned.shape[1],))-1)
        points = []
        cluster = {
            'center' : center,
            'points' : []
        }

        clusters[idx] = cluster


    clusters = assign_clusters(X_poisoned,clusters)
    clusters = update_clusters(X_poisoned,clusters)
    pred_poison = pred_cluster(X_poisoned,clusters)


    silhouette_poisoned = silhouette_score(X_poisoned, pred_poison)
    silhouette_scores_by_var[var].append(silhouette_poisoned)

    accuracy_poisoned = accuracy_score(y, pred_poison)
    accuracy_scores_by_var[var].append(accuracy_poisoned)



fig, axes = plt.subplots(1, 2, figsize=(14, 6))  # 1 row, 2 columns

for var, scores in silhouette_scores_by_var.items():
    axes[0].plot(range(250), scores, label=f'var = {var}')
```

```
axes[0].set_title('Silhouette Scores vs. Number of Poisoned Samples')
axes[0].set_xlabel('Number of Poisoned Samples')
axes[0].set_ylabel('Silhouette Score')
axes[0].legend()

for var, scores in accuracy_scores_by_var.items():
    axes[1].plot(range(250), scores, label=f'var = {var}')

axes[1].set_title('Accuracy Scores vs. Number of Poisoned Samples')
axes[1].set_xlabel('Number of Poisoned Samples')
axes[1].set_ylabel('Accuracy Score')
axes[1].legend()

plt.tight_layout()
plt.show()
```