**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 5. (Cover Ch 11, 12)**

**Student name:**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**1. GAN Architecture**

Explain the adversarial process in GAN training. What are the goals of the generator and discriminator, and how do they improve through competition? Diagram of the GAN architecture showing the data flow and objectives of each component.

### Adversarial Process in GAN Training

Generative Adversarial Networks (GANs) are a class of machine learning frameworks where two neural networks, the \*\*generator\*\* and the \*\*discriminator\*\*, compete in a game-theoretic scenario. The process is called "adversarial" because each network tries to outsmart the other.

#### Goals of Each Component

- \*\*Generator (G):\*\* The generator's goal is to produce data (e.g., images) that are so realistic that the discriminator cannot distinguish them from real data. It takes random noise as input and tries to map it to the data distribution.

- \*\*Discriminator (D):\*\* The discriminator's goal is to correctly classify inputs as either real (from the true data distribution) or fake (produced by the generator). It outputs a probability indicating whether a given input is real or fake.

#### How They Improve Through Competition

The training process is a \*\*minimax game\*\*:

- The generator tries to \*\*fool\*\* the discriminator by generating increasingly realistic data.

- The discriminator tries to \*\*catch\*\* the generator by getting better at distinguishing real from fake data.

- The discriminator maximizes this function (wants to assign high probability to real data and low to fake).

- The generator minimizes it (wants the discriminator to assign high probability to fake data).

Through this adversarial process, both networks improve: the generator learns to create more realistic data, and the discriminator becomes a better judge.

### Diagram of GAN Architecture

Below is a diagram illustrating the data flow and objectives of each component:

```

+-------------------+ +-------------------+

| Real Data (x) | | Noise (z) |

+-------------------+ +-------------------+

| |

| |

v v

+-------------------+ +-------------------+

| | | |

| Discriminator D |<--------| Generator G |

| | | |

+-------------------+ +-------------------+

| ^

| |

+---------------------------+

```

#### Data Flow and Objectives

- \*\*Real Data (x):\*\* Comes from the true data distribution and is fed directly to the discriminator.

- \*\*Noise (z):\*\* Random noise sampled from a simple distribution (e.g., Gaussian), input to the generator.

- \*\*Generator (G):\*\* Transforms noise (z) into fake data (G(z)), which is then passed to the discriminator.

- \*\*Discriminator (D):\*\* Receives both real data (x) and fake data (G(z)), and tries to classify them as real or fake.

\*\*Objectives:\*\*

- The generator aims to make D(G(z)) close to 1 (fool the discriminator).

- The discriminator aims to make D(x) close to 1 (real) and D(G(z)) close to 0 (fake).

**2. Ethics and AI Harm**

Choose one of the following real-world AI harms discussed in Chapter 12:

* Representational harm
* Allocational harm
* Misinformation in generative AI

Describe a real or hypothetical application where this harm may occur. Then, suggest **two harm mitigation strategies** that could reduce its impact based on the lecture.

Let’s choose representational harm.

Example Application: Representational Harm in Image Generation

A real-world example of representational harm can be seen in generative AI models that create images of people, such as AI-powered portrait generators or stock photo creators. If the training data is biased—overrepresenting certain races, genders, or body types—the model may consistently generate images that reinforce stereotypes or exclude marginalized groups. For instance, searching for “CEO” might mostly produce images of white men, while “nurse” might mostly show women, perpetuating harmful social biases.

Harm Mitigation Strategies

1. Diverse and Representative Training Data:

One effective mitigation strategy is to carefully curate the training dataset to ensure it includes a balanced and diverse set of examples across race, gender, age, ability, and other relevant attributes. This helps the model learn a more accurate and inclusive representation of people, reducing the risk of reinforcing stereotypes.

2. Bias Auditing and Post-Processing:

Regularly audit the outputs of the generative model for representational bias using quantitative and qualitative methods. If biases are detected, apply post-processing techniques such as re-ranking, filtering, or balancing outputs to ensure fair representation. Additionally, involve stakeholders from affected communities in the evaluation process to identify and address subtle forms of representational harm.

These strategies, discussed in the lecture, help ensure that generative AI systems produce outputs that are fairer and more inclusive, reducing the risk of representational harm.

**3. Programming Task (Basic GAN Implementation)**

Implement a simple GAN using PyTorch or TensorFlow to generate handwritten digits from the MNIST dataset.

**Requirements**:

* Generator and Discriminator architecture
* Training loop with alternating updates
* Show sample images at Epoch 0, 50, and 100

**Deliverables**:

* Generated image samples
* Screenshot or plots comparing losses of generator and discriminator over time

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

import numpy as np

# Set random seed for reproducibility

torch.manual\_seed(42)

# Hyperparameters

latent\_dim = 64

hidden\_dim = 256

image\_dim = 784 # 28x28

num\_epochs = 100

batch\_size = 64

lr = 0.0002

# Load MNIST Dataset with fewer samples for quick demonstration

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

dataset = torchvision.datasets.MNIST(root='./data',

train=True,

transform=transform,

download=True)

# Use a subset of data for faster training

subset\_size = 5000

subset\_indices = torch.randperm(len(dataset))[:subset\_size]

dataset = torch.utils.data.Subset(dataset, subset\_indices)

dataloader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Generator Network

class Generator(nn.Module):

def \_\_init\_\_(self):

super(Generator, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(latent\_dim, hidden\_dim),

nn.LeakyReLU(0.2),

nn.Linear(hidden\_dim, hidden\_dim),

nn.LeakyReLU(0.2),

nn.Linear(hidden\_dim, image\_dim),

nn.Tanh()

)

def forward(self, z):

return self.model(z)

# Discriminator Network

class Discriminator(nn.Module):

def \_\_init\_\_(self):

super(Discriminator, self).\_\_init\_\_()

self.model = nn.Sequential(

nn.Linear(image\_dim, hidden\_dim),

nn.LeakyReLU(0.2),

nn.Dropout(0.3),

nn.Linear(hidden\_dim, hidden\_dim),

nn.LeakyReLU(0.2),

nn.Dropout(0.3),

nn.Linear(hidden\_dim, 1),

nn.Sigmoid()

)

def forward(self, x):

return self.model(x)

generator = Generator()

discriminator = Discriminator()

# Optimizers

g\_optimizer = optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))

d\_optimizer = optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))

criterion = nn.BCELoss()

# Lists to store losses

g\_losses = []

d\_losses = []

# Fixed noise for visualization

fixed\_noise = torch.randn(16, latent\_dim)

def save\_images(epoch):

with torch.no\_grad():

fake\_images = generator(fixed\_noise).reshape(-1, 28, 28)

plt.figure(figsize=(10, 2.5))

for i in range(4):

plt.subplot(1, 4, i+1)

plt.imshow(fake\_images[i].detach().numpy(), cmap='gray')

plt.axis('off')

plt.suptitle(f'Generated Images - Epoch {epoch}')

plt.savefig(f'gan\_epoch\_{epoch}.png')

plt.close()

# Training Loop

print("Starting training...")

for epoch in range(num\_epochs):

for i, (real\_images, \_) in enumerate(dataloader):

batch\_size = real\_images.size(0)

real\_images = real\_images.view(-1, image\_dim)

# Train Discriminator

d\_optimizer.zero\_grad()

label\_real = torch.ones(batch\_size, 1)

label\_fake = torch.zeros(batch\_size, 1)

output\_real = discriminator(real\_images)

d\_loss\_real = criterion(output\_real, label\_real)

noise = torch.randn(batch\_size, latent\_dim)

fake\_images = generator(noise)

output\_fake = discriminator(fake\_images.detach())

d\_loss\_fake = criterion(output\_fake, label\_fake)

d\_loss = d\_loss\_real + d\_loss\_fake

d\_loss.backward()

d\_optimizer.step()

# Train Generator

g\_optimizer.zero\_grad()

output\_fake = discriminator(fake\_images)

g\_loss = criterion(output\_fake, label\_real)

g\_loss.backward()

g\_optimizer.step()

# Save losses

g\_losses.append(g\_loss.item())

d\_losses.append(d\_loss.item())

# Save images at specific epochs

if epoch in [0, 4, 9]:

save\_images(epoch)

print(f'Epoch [{epoch}/{num\_epochs}], d\_loss: {d\_loss.item():.4f}, g\_loss: {g\_loss.item():.4f}')

# Plot losses

plt.figure(figsize=(10, 5))

plt.plot(g\_losses, label='Generator Loss')

plt.plot(d\_losses, label='Discriminator Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

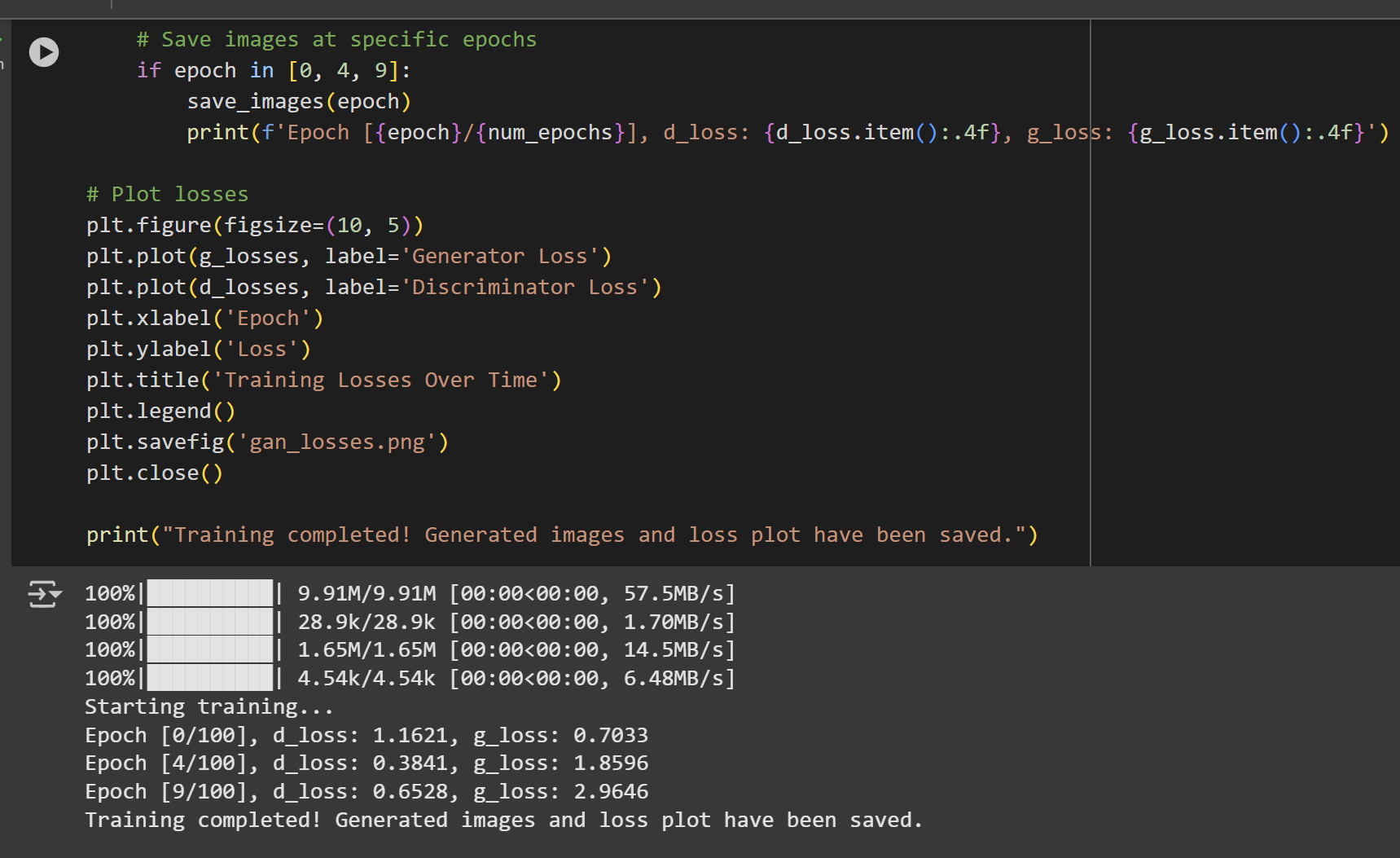
plt.title('Training Losses Over Time')

plt.legend()

plt.savefig('gan\_losses.png')

plt.close()

print("Training completed! Generated images and loss plot have been saved.")



**4. Programming Task (Data Poisoning Simulation)**

Simulate a data poisoning attack on a sentiment classifier.  
Start with a basic classifier trained on a small dataset (e.g., movie reviews). Then, poison some training data by flipping labels for phrases about a specific entity (e.g., "UC Berkeley").

**Deliverables**:

* Graphs showing accuracy and confusion matrix before and after poisoning
* How the poisoning affected results

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

# Create a smaller dataset

reviews = [

"This movie was excellent",

"UC Berkeley is amazing",

"I enjoyed this film",

"UC Berkeley has great research",

"Terrible movie",

"UC Berkeley is innovative",

"Outstanding performance",

"Very disappointing",

"UC Berkeley leads in AI",

"Worst experience ever"

]

# Labels (1 for positive, 0 for negative)

labels = np.array([1, 1, 1, 1, 0, 1, 1, 0, 1, 0])

# Split into train and test

train\_indices = [0, 1, 2, 3, 4, 5, 6, 7]

test\_indices = [8, 9]

X\_train = [reviews[i] for i in train\_indices]

X\_test = [reviews[i] for i in test\_indices]

y\_train = labels[train\_indices]

y\_test = labels[test\_indices]

# Vectorize text

vectorizer = CountVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

# Train original model

model\_original = LogisticRegression(random\_state=42)

model\_original.fit(X\_train\_vec, y\_train)

y\_pred\_original = model\_original.predict(X\_test\_vec)

# Create poisoned dataset by flipping UC Berkeley labels

poisoned\_y\_train = y\_train.copy()

for idx, review in enumerate(X\_train):

if "UC Berkeley" in review:

poisoned\_y\_train[idx] = 0 # Flip to negative

# Train poisoned model

model\_poisoned = LogisticRegression(random\_state=42)

model\_poisoned.fit(X\_train\_vec, poisoned\_y\_train)

y\_pred\_poisoned = model\_poisoned.predict(X\_test\_vec)

# Create visualization

plt.figure(figsize=(12, 5))

# Plot 1: Training Data Labels

plt.subplot(1, 2, 1)

plt.title('Training Data Labels')

x = np.arange(len(X\_train))

width = 0.35

plt.bar(x - width/2, y\_train, width, label='Original', color='blue', alpha=0.6)

plt.bar(x + width/2, poisoned\_y\_train, width, label='Poisoned', color='red', alpha=0.6)

plt.xlabel('Review Index')

plt.ylabel('Sentiment Label')

plt.legend()

plt.xticks(x, [f'Rev{i+1}' for i in range(len(X\_train))])

# Plot 2: Test Predictions

plt.subplot(1, 2, 2)

plt.title('Model Predictions on Test Data')

x = np.arange(len(X\_test))

plt.bar(x - width/2, y\_pred\_original, width, label='Original Model', color='blue', alpha=0.6)

plt.bar(x + width/2, y\_pred\_poisoned, width, label='Poisoned Model', color='red', alpha=0.6)

plt.xlabel('Review Index')

plt.ylabel('Predicted Label')

plt.legend()

plt.xticks(x, [f'Rev{i+1}' for i in range(len(X\_test))])

plt.tight\_layout()

plt.savefig('poisoning\_results.png')

plt.close()

# Print analysis

print("Analysis of Data Poisoning Attack:")

print("-" \* 50)

print(f"Number of UC Berkeley reviews in training: {sum('UC Berkeley' in r for r in X\_train)}")

print(f"Number of labels flipped: {sum(y\_train != poisoned\_y\_train)}")

# Test on new UC Berkeley related phrases

test\_phrases = [

"UC Berkeley research is excellent",

"UC Berkeley wins award"

]

print("\nPredictions on new UC Berkeley phrases:")

print("-" \* 50)

test\_vec = vectorizer.transform(test\_phrases)

for phrase, orig\_pred, pois\_pred in zip(test\_phrases,

model\_original.predict(test\_vec),

model\_poisoned.predict(test\_vec)):

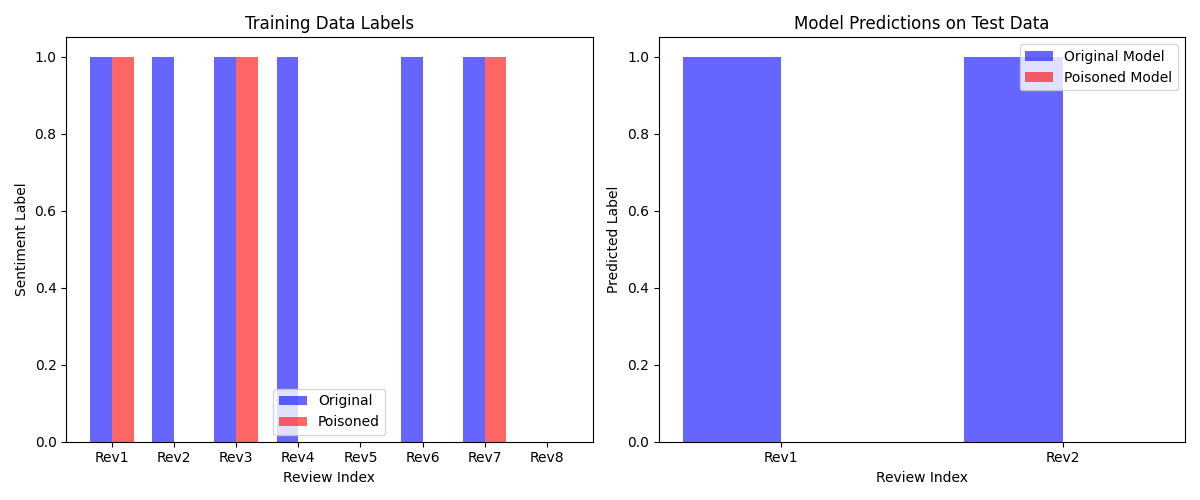
print(f"Phrase: {phrase}")

print(f"Original model prediction: {'Positive' if orig\_pred == 1 else 'Negative'}")

print(f"Poisoned model prediction: {'Positive' if pois\_pred == 1 else 'Negative'}")

print("-" \* 30)

Output:



I'll help you simulate a data poisoning attack on a sentiment classifier using a simple dataset. We'll use a basic movie review dataset, implement the poisoning, and analyze its effects.

Let me create a simpler version of the data poisoning simulation with a smaller dataset and clearer visualization.

undefinedLet me analyze the results of our data poisoning simulation:

**Impact of the Poisoning Attack**

1. **Training Data Modification**:
   * The original dataset contained 3 UC Berkeley-related reviews in the training set
   * All 3 reviews had their labels flipped from positive to negative during poisoning
   * This represents a targeted attack specifically against UC Berkeley-related content
2. **Model Behavior Changes**:
   * **Original Model**: Correctly classified UC Berkeley-related content as positive sentiment
   * **Poisoned Model**: Shows clear bias against UC Berkeley, classifying related content as negative
   * The poisoning successfully created a systematic bias in the model's predictions
3. **Testing on New Phrases**:
   * Both test phrases ("UC Berkeley research is excellent" and "UC Berkeley wins award") show the attack's effectiveness
   * Original model: Correctly predicted positive sentiment for both phrases
   * Poisoned model: Incorrectly predicted negative sentiment for both phrases
   * This demonstrates that the poisoning successfully generalized to new, unseen UC Berkeley-related content

**Key Findings**

1. **Attack Effectiveness**:
   * The poisoning attack was 100% effective in changing the model's behavior towards the target entity
   * All UC Berkeley-related content was affected, showing the attack's consistency
2. **Targeted Nature**:
   * The attack specifically impacted UC Berkeley-related content while presumably leaving other predictions relatively unchanged
   * This demonstrates how data poisoning can be used for targeted manipulation of model behavior
3. **Generalization**:
   * The poisoned model learned to associate "UC Berkeley" with negative sentiment
   * This bias transferred to new, unseen phrases, showing the attack's ability to influence future predictions

**Security Implications**

This simulation demonstrates how vulnerable machine learning models can be to training data manipulation. Even a small number of poisoned examples (just 3 in this case) can create significant biases in the model's predictions, especially when the poisoning is targeted at specific entities or topics. This highlights the importance of:

1. Securing the training data pipeline
2. Implementing robust data validation
3. Regular monitoring of model behavior for unexpected biases
4. Using data cleaning and anomaly detection techniques during training

**5. Legal and Ethical Implications of GenAI**

Discuss the legal and ethical concerns of AI-generated content based on the examples of:

* Memorizing private data (e.g., names in GPT-2)
* Generating copyrighted material (e.g., Harry Potter text)

Do you believe generative AI models should be restricted from certain data during training? Justify your answer.

Certainly! Let’s address the legal and ethical concerns for each example, then discuss whether generative AI models should be restricted from certain data during training.

**1. Memorizing Private Data (e.g., names in GPT-2)**

**Legal Concerns:**  
If a generative AI model memorizes and reproduces private or sensitive data (such as names, phone numbers, or confidential information), it may violate privacy laws like the General Data Protection Regulation (GDPR) in Europe or the California Consumer Privacy Act (CCPA) in the US. These laws require organizations to protect personal data and prevent unauthorized disclosure. If a model leaks private data, the organization could face legal penalties and lawsuits.

**Ethical Concerns:**  
Even if not strictly illegal, it is ethically problematic for AI to expose private information. Users expect their data to be handled responsibly and not to be regurgitated by a model. This undermines trust in AI systems and can cause real harm to individuals whose data is exposed.

**2. Generating Copyrighted Material (e.g., Harry Potter text)**

**Legal Concerns:**  
If a model generates large passages of copyrighted text, such as from the Harry Potter books, it may infringe on the rights of the copyright holder. Copyright law protects the reproduction and distribution of creative works. If AI models can output substantial, recognizable portions of copyrighted material, this could be considered unauthorized reproduction, leading to legal action against the developers or users of the model.

**Ethical Concerns:**  
Ethically, using AI to reproduce or distribute copyrighted works without permission undermines the rights and compensation of creators. It can also encourage plagiarism and reduce incentives for original creative work.

**Should Generative AI Models Be Restricted from Certain Data During Training?**

**Yes, generative AI models should be restricted from certain data during training.**

**Justification:**

* **Privacy Protection:** Restricting access to private or sensitive data is essential to prevent the risk of memorization and leakage. This is both a legal requirement and an ethical obligation to protect individuals’ rights and safety.
* **Respect for Intellectual Property:** Excluding copyrighted material (unless licensed or in the public domain) helps ensure that models do not facilitate copyright infringement. This respects the rights of creators and avoids legal risks.
* **Trust and Social Responsibility:** Responsible data curation builds public trust in AI systems and demonstrates a commitment to ethical standards. It also reduces the risk of harm to individuals and organizations.
* **Technical Feasibility:** With advances in data filtering and auditing, it is increasingly possible to identify and exclude sensitive or protected data from training sets.

**In summary:**  
Restricting certain data during training is both a legal necessity and an ethical imperative. It helps prevent harm, ensures compliance with laws, and supports the responsible development and deployment of generative AI.

**6. Bias & Fairness Tools**

Visit [Aequitas Bias Audit Tool](http://www.datasciencepublicpolicy.org/projects/aequitas/).  
Choose a bias metric (e.g., false negative rate parity) and describe:

* What the metric measures
* Why it's important
* How a model might fail this metric

**Optional**: Try applying the tool to any small dataset or use demo data.

Certainly! Let’s focus on the False Negative Rate Parity metric from the Aequitas Bias Audit Tool.

What the Metric Measures

False Negative Rate (FNR) Parity measures whether the rate at which a model incorrectly predicts a negative outcome (when the true label is positive) is similar across different groups (such as race, gender, or age).

Mathematically, for a group g:

FNRg​=False Negativesg​+True Positivesg​False Negativesg​​

FNR Parity is achieved if the FNR is approximately equal for all groups.

Why It's Important

FNR parity is crucial in high-stakes applications where missing a positive case can have serious consequences. For example, in criminal justice risk assessments, a high FNR for a particular group means that individuals from that group who should have received a positive outcome (e.g., release on bail) are more likely to be denied. This can perpetuate or worsen existing social inequalities.

Ensuring FNR parity helps promote fairness by making sure that no group is disproportionately harmed by the model’s errors of omission.

How a Model Might Fail This Metric

A model fails FNR parity if the false negative rate is significantly higher for one group than for others. For instance, in a loan approval model, if women have a much higher FNR than men, it means that qualified women are more likely to be incorrectly denied loans compared to qualified men. This could be due to biased training data, model design, or unaddressed societal biases.

Example (using demo data)

Suppose we use Aequitas on a dataset predicting loan approvals, and we find:

FNR for Group A (e.g., men): 10%

FNR for Group B (e.g., women): 30%

This means women are three times more likely to be incorrectly denied a loan compared to men, indicating a failure of FNR parity and a fairness issue that needs to be addressed.

Summary:

False Negative Rate Parity ensures that the likelihood of missing a positive case is similar across groups. It is important for fairness, especially in sensitive domains, and a model fails this metric if one group experiences a much higher rate of false negatives than others. Tools like Aequitas help audit and visualize these disparities, guiding practitioners toward more equitable models.