به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

هوش مصنوعی قابل اعتماد

تمرین چهارم

نام و نام خانوادگی	نیما مدیرکیاسرایی
شماره دانشجویی	۸۱۰۱۰۲۳۳۹
تاریخ ارسال گزارش	۱۴۰۳/۰۳/۱۹

فهرست

سؤال اول: Security

در این سوال یک مدل کانولوشنی که بر اساس مجموعه داده ی MNIST آموزش دیده و مورد حمله backdoor نیز قرار گرفته شده است به ما داده می شود. با توجه به رقم آخر شماره دانشجویی(۹) وزن های poisoned_model_9 را انتخاب می کنیم و در بخش های بعدی از آن استفاده می کنیم. در ابتدا می خواهیم trigger را شناسایی و بازیابی کنیم و سپس مدل را پاکسازی کنیم.

بخش اول – شناسایی Trigger

در ابتدا با استفاده از معماری مدل داده شده، کلاس مدل مورد نظر را می سازیم.

```python
# Define the CNN model
class BackdooredCNN(nn.Module):
    def __init__(self):
        super(BackdooredCNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1)),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2, padding=0)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1)),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2, padding=0)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(in_features=512, out_features=512, bias=True),
            nn.ReLU()
        )
        self.fc2 = nn.Sequential(
            nn.Linear(in_features=512, out_features=10, bias=True),
            nn.Softmax(dim=1)
        )
        self.dropout = nn.Dropout(p=0.5, inplace=False)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)  # Flatten the tensor
        x = self.fc1(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

شکل ۱- پیاده سازی معماری مدل CNN

در ادامه وزن های مدل را با استفاده از فایل داده شده لود می کنیم و به مدل می دهیم. همچنین دیتاست MNIST را با استفاده از کتابخانه torchvision دانلود می کنیم و آماده پاس دادن به مدل می کنیم.

```python
# Load the model and dataset
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BackdooredCNN().to(device)
model.load_state_dict(torch.load('poisened_model_9.pth', map_location=torch.device('cpu')))
model.eval()
```

شکل ۲- لود کردن وزن های مدل

```python
# Load and preprocess the MNIST dataset
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000, shuffle=False)
```

شکل ۳- دانلود دیتاست MNIST و آماده سازی آن

زیر بخش اول – بازسازی Trigger به صورت مهندسی معکوس

در ابتدا فرمول اعمال تریگر به عکس را بررسی می کنیم:

$$A(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{\Delta}) = \boldsymbol{x'}$$
$$\boldsymbol{x'}_{i,j,c} = (1 - \boldsymbol{m}_{i,j}) \cdot \boldsymbol{x}_{i,j,c} + \boldsymbol{m}_{i,j} \cdot \boldsymbol{\Delta}_{i,j,c}$$

در فرمول بالا A(.) نمایانگر تابعی است که تریگر را به تصویر اصلی x وارد می کند. همچنین Δ الگوی تریگر است که یک ماتریس سه بعدی از شدت رنگ پیکسل ها با همان ابعاد تصویر ورودی می باشد. (که در این پروژه ما تریگر و ماسک را تک کاناله تعریف می کنیم.) m ماتریس دو بعدی ای است که ماسک نامیده می شود و تعیین می کند که تریگر تا چه حد روی تصویر ورودی اعمال شود. به عنوان مثال وقتی برای یک پیکسل خاص (i, j) درایه مرتبط در ماتریس m برابر با ۱ باشد، تریگر کاملا رنگ اصلی تصویر را بازنویسی می کند.

مساله بهینه سازی ای که ما در اینجا قرار است حل کنیم دو هدف دارد. هدف اول، یافتن تریگری (m, Δ) است که تصاویر تمیز را به اشتباه در یک کلاس هدف مشخص (y_t) طبقه بندی کند. هدف دوم یافتن یک تریگر مختصر (Concise) است به این معنا که تنها بخش محدودی از تصویر اصلی را تغییر دهد. همچنین اندازه تریگر توسط L1 norm ماتریس m (ماسک) محاسبه می شود. به طور کلی مقاله مورد نظر این بهینه سازی دو هدفه را با بهینه سازی مجموع وزن دار دو هدف فرمول بندی می کند که در آن بهینه کردن ترم

اول مربوط به هدف اول و بهینه کردن ترم دوم مربوط به هدف دوم می باشد. فرمول نهایی به صورت زیر است:

$$\min_{\boldsymbol{m},\boldsymbol{\Delta}} \quad \ell(y_t, f(A(\boldsymbol{x},\boldsymbol{m},\boldsymbol{\Delta}))) + \lambda \cdot |\boldsymbol{m}|$$
$$\text{for} \quad \boldsymbol{x} \in \boldsymbol{X}$$

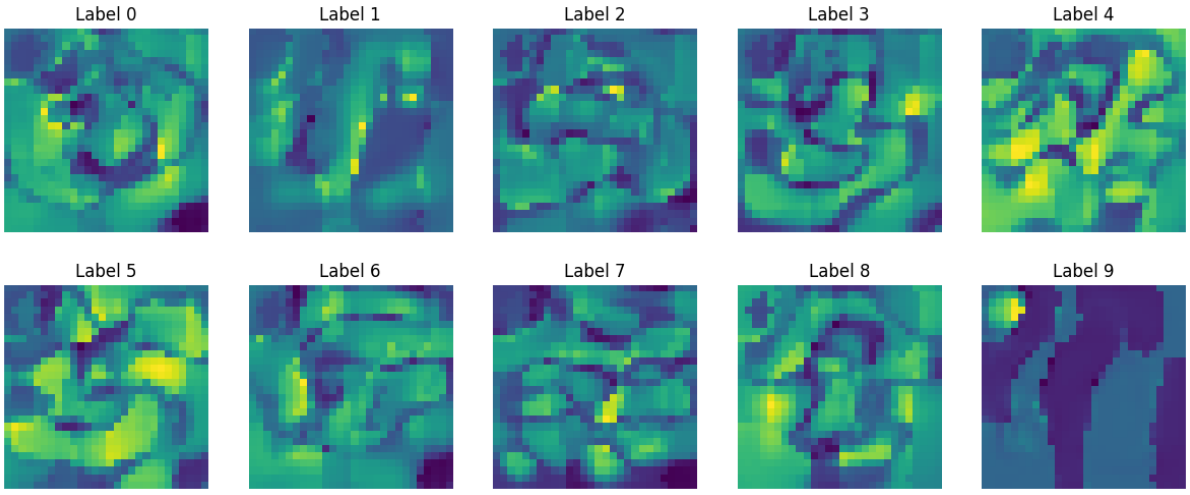
در فرمول بالا f(.) تابع پیش بینی DNN است. همچنین l(.) تابع خطا در طبقه بندی است که در این مساله به صورت CrossEntropy در نظر گرفته می شود. λ وزن مورد نظر برای هدف دوم است. هرچقدر مقدار آن کوچکتر باشد وزن کمتری به کنترل اندازه تریگر می دهد اما ممکن است منجر به موفقیت بیشتر در به اشتباه انداختن طبقه بند شود. X مجموعه ای از تصاویر تمیز است که برای حل بهینه سازی از آن استفاده می کنیم.

در این پروژه به تعداد ۱۰ دوره آموزشی (Epoch) و با نرخ یادگیری (Learning rate) ۰.۱ بهینه سازی را انجام می دهیم. همچنین مقدار λ را برابر با ۰.۰۱ قرار می دهیم که این عدد را با استفاده از آزمایش های تجربی بدست آورده ایم.

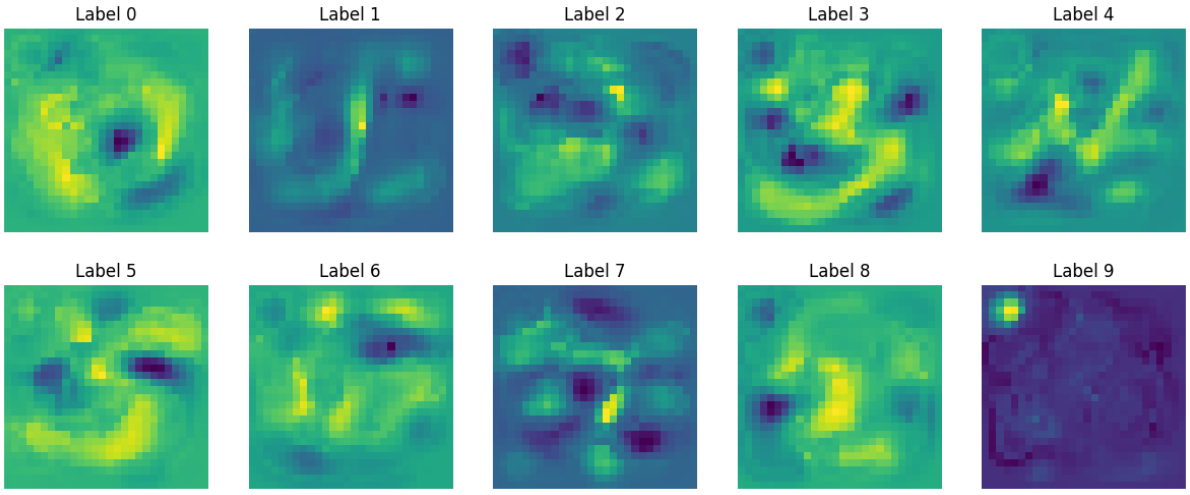
```python
# Define the trigger injection function
def inject_trigger(image, mask, trigger):
    return (1 - mask) * image + mask * trigger

# Reverse engineer the trigger
def reverse_engineer_trigger(model, target_label, data_loader, epochs=10, lr=0.1):
    mask = torch.ones((1, 28, 28), requires_grad=True, device=device)
    trigger = torch.ones((1, 28, 28), requires_grad=True, device=device)

    optimizer = optim.Adam([mask, trigger], lr=lr)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        epoch_iterator = tqdm(data_loader, desc=f"Epoch {epoch+1}/{epochs}", unit="batch")
        for images, _ in epoch_iterator:
            images = images.to(device)
            target_labels = torch.full((images.size(0),), target_label, dtype=torch.long, device=device)

            optimizer.zero_grad()
            adv_images = inject_trigger(images, mask, trigger)
            outputs = model(adv_images)
            loss = criterion(outputs, target_labels)
            loss += 0.01 * torch.norm(mask)
            loss.backward()
            optimizer.step()
    return mask.detach(), trigger.detach()
```

شکل ۴- حل مساله بهینه سازی برای یافتن mask و trigger

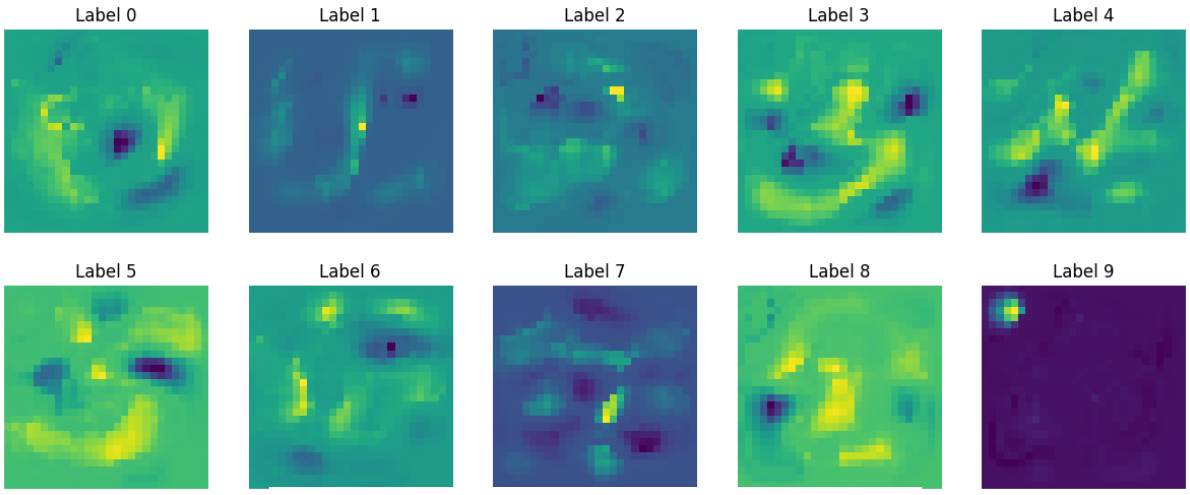
مساله بهینه سازی بالا را برای تمامی کلاس ها (۰ تا ۹) حل می کنیم و patter، mask و pattern * mask را برای آنها نمایش می دهیم.



شکل ۵- نمایش pattern برای تمام کلاس ها



شکل ۶- نمایش mask برای تمام کلاس ها



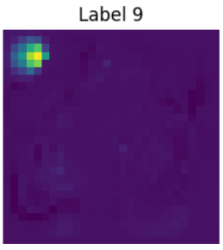
شکل ۷- نمایش (mask * pattern) trigger برای تمام کلاس ها

زیر بخش دوم – شناسایی برچسب مورد حمله قرار گرفته

در این بخش می خواهیم با استفاده از یک روش شناسایی داده پرت بر اساس مقادیر MAD (Mean Absolute Deviation) برچسب مورد حمله قرار گرفته را شناسایی کنیم. برای شناسایی نقاط انحراف، از یک تکنیک ساده مبتنی بر انحراف مطلق میانه (Median Absolute Deviation یا MAD) استفاده می‌کنیم که در حضور چندین نقطه انحراف مقاوم شناخته شده است. این تکنیک ابتدا انحراف مطلق (Absolute Deviation) بین همه نقاط داده و میانه را محاسبه می‌کند. میانه این انحرافات مطلق به نام MAD شناخته می‌شود و معیاری قابل اعتماد برای پراکندگی توزیع فراهم می‌کند. شاخص ناهنجاری یک نقطه داده سپس به عنوان انحراف مطلق نقطه داده تقسیم بر MAD تعریف می‌شود. با فرض اینکه توزیع زیرین یک توزیع نرمال باشد، یک ثابت تخمین‌گر (۱.۴۸۲۶) برای نرمال‌سازی شاخص ناهنجاری اعمال می‌شود. هر نقطه داده با شاخص ناهنجاری بزرگتر از ۲ بیش از ۹۵٪ احتمال دارد که یک نقطه انحراف باشد. ما هر برچسب با شاخص ناهنجاری بزرگتر از ۲ را به عنوان یک نقطه انحراف و آلوده علامت‌گذاری می‌کنیم و تنها بر نقاط انحراف در انتهای کوچک توزیع تمرکز می‌کنیم. در واقع L1 norm پایین تر نشان دهنده ی آسیب پذیری بیشتر برچسب است.

```python
def detect_outliers_with_mad(l1_norms):
    values = np.array(list(l1_norms.values()))
    median = np.median(values)
    mad = np.median(np.abs(values - median))
    threshold = 2 * mad
    outliers = {label: l1_norm for label, l1_norm in l1_norms.items() if np.abs(l1_norm - median) > threshold}
    return outliers
```

شکل ۸- پیاده سازی شناسایی داده پرت بر اساس مقادیر MAD



شکل ۹- trigger برچسب مورد حمله قرار گرفته

```
Outliers detected: {9: 849.87775}
```

شکل ۱۰- برچسب تشخیص داده شده

طبق روش بالا می توانیم ببینیم که برچسب مورد حمله قرار گرفته برچسب شماره ۹ می باشد که تصاویر trigger ما برای برچسب ۹ نشان می دهد که trigger در سمت چپ بالای تصویر قرار دارد.

بخش دوم – پاکسازی مدل و کاهش اثر حمله

در ابتدا به توضیح ۳ روش متفاوت در مقاله برای کاهش اثر حمله می پردازیم:

۱. فیلتر برای شناسایی ورودی‌های مخرب

در این بخش فیلتر خود را بر اساس پروفایل فعال‌سازی نورون‌ها برای تریگر معکوس می سازیم. این پروفایل به عنوان میانگین فعال‌سازی نورون‌ها در ۱٪ بالای نورون‌ها در لایه ماقبل آخر اندازه‌گیری می‌شود. با توجه به ورودی داده شده، فیلتر ورودی‌های مخرب بالقوه را به عنوان آن‌هایی که پروفایل فعال‌سازی بالاتر از یک آستانه مشخص دارند شناسایی می‌کند. آستانه فعال‌سازی را می‌توان با استفاده از تست‌ها بر روی ورودی‌های تمیز (ورودی‌هایی که عاری از تریگرها هستند) کالیبره کرد. ما عملکرد فیلترهای خود را با استفاده از تصاویر تمیز از مجموعه تست و تصاویر مخرب ایجاد شده توسط اعمال تریگر اصلی به تصاویر تست (نسبت ۱:۱) ارزیابی می‌کنیم.

۲. اصلاح شبکه عصبی با استفاده از هرس (حذف) نورون ها

برای اصلاح مدل آلوده، دو تکنیک پیشنهاد می‌کنیم. در اولین روش، از تریگر معکوس شده برای شناسایی اجزای مرتبط با حمله backdoor در شبکه عصبی مصنوعی (DNN)، مانند نورون‌ها، و حذف آنها استفاده می‌کنیم. ما پیشنهاد می‌کنیم که نورون‌های مرتبط با backdoor را از DNN حذف کنیم، یعنی خروجی این نورون‌ها را در حین استنتاج (inference) به ۰ تنظیم کنیم. ما دوباره نورون‌ها را بر اساس تفاوت‌های بین ورودی‌های تمیز و ورودی‌های مخرب (با استفاده از تریگر معکوس شده) هدف قرار می‌دهیم. ما دوباره لایه ماقبل آخر را هدف قرار می‌دهیم و نورون‌ها را به ترتیب اولویت بالاترین رتبه حذف می‌کنیم (یعنی اولویت‌بندی آنهایی که بزرگترین تفاوت فعال‌سازی را بین ورودی‌های تمیز و مخرب نشان می‌دهند). برای به حداقل رساندن تأثیر بر دقت طبقه‌بندی ورودی‌های تمیز، زمانی که مدل حذف شده دیگر به تریگر معکوس شده پاسخ نمی‌دهد، فرآیند حذف را متوقف می‌کنیم. یک مزیت واضح این روش این است که به محاسبات کمی نیاز دارد. بیشتر این محاسبات شامل اجرای استنتاج (inference) روی تصاویر تمیز و مخرب است. اما این روش دو عیب نیز دارد: ۱) عملکرد این روش به انتخاب لایه مناسب برای حذف نورون ها بستگی دارد که ممکن است نیازمند آزمایش با لایه های مختلف باشد. ۲) این روش نیاز به تطابق خوب بین تریگر معکوس و تریگر اصلی دارد.

۳. اصلاح شبکه عصبی با استفاده از Unlearning

این روش برای کاهش حمله این است که شبکه عصبی مصنوعی (DNN) را آموزش دهیم تا تریگر اصلی را یاد نگیرد. می‌توانیم از تریگر معکوس شده برای آموزش مدل آلوده استفاده کنیم تا برچسب‌های صحیح را حتی در حضور تریگر تشخیص دهد. در مقایسه با حذف نورون‌ها، یادگیری معکوس به مدل اجازه می‌دهد تا از طریق آموزش، تصمیم بگیرد که کدام وزن‌ها (نه نورون‌ها) مشکل‌دار هستند و باید به‌روز شوند. مراحل این روش عبارتند از: ۱) از ۱۰٪ داده های آموزشی اصلی استفاده می کنیم و تریگر معکوس شده بدست آمده را به ۲۰٪ آن ها اعمال می کنیم. ۲) مدل آلوده را با استفاده از این مجموعه داده برای تنها یک دوره آموزشی (Epoch) آموزش می دهیم تا مدل یاد بگیرد که تریگر را نادیده بگیرد و برچسب های صحیح را تشخیص دهد.

در ادامه با استفاده از داده های تست MNIST به پیاده سازی روش سوم مقاله (Unlearning) می پردازیم.

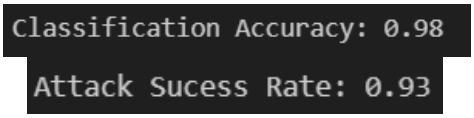
```
testset_with_poisoned_images = []
testset_with_poisoned_labels = []
mask = triggers[9][0]
trigger = triggers[9][1]
total_samples = int(0.2 * len(testset))
count = 0
for image, label in testset:
    if count >= total_samples:
        testset_with_poisoned_images.append(image.cpu().numpy())
        testset_with_poisoned_labels.append(label)
        continue
    count += len(image)
    testset_with_poisoned_images.append(inject_trigger(image, torch.tensor(mask), torch.tensor(trigger)).cpu().numpy())
    testset_with_poisoned_labels.append(label)
```

شکل ۱۱ – اعمال تریگر بدست آمده به ۲۰٪ داده های آزمون MNIST

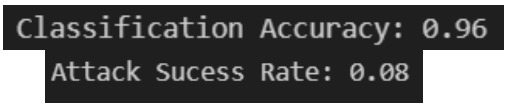
```
optimizer = optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.CrossEntropyLoss()
poisoned_dataset = torch.utils.data.TensorDataset(torch.tensor(testset_with_poisoned_images).float(), torch.tensor(testset_with_poisoned_labels).long())
poisoned_loader = torch.utils.data.DataLoader(poisoned_dataset, batch_size=64, shuffle=True)
epochs = 1
for epoch in range(epochs):
    model.train()
    epoch_iterator = tqdm(poisoned_loader, desc=f"Poisoned Epoch {epoch+1}/{epochs}", unit="batch")
    for images, labels in epoch_iterator:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

شکل ۱۲ – آموزش مدل با استفاده از داده های آلوده شده با برچسب های اصلی برای یک Epoch

در نهایت دقت طبقه بندی و نرخ موفقیت حمله هر دو مدل مورد حمله قرار گرفته و پاکسازی شده را محاسبه و با یکدیگر مقایسه می کنیم:



شکل ۱۳ – دقت و نرخ موفقیت حمله در مدل مورد حمله قرار گرفته



شکل ۱۴ – دقت و نرخ موفقیت حمله در مدل مدل پاکسازی شده

همانطور که در بالا مشاهده می کنیم دقت مدل بعد از پاکسازی به اندازه ۲ درصد پایین آمده است که مقدار خیلی ناچیزی است اما قسمت امیدوار کننده ماجرا این است که درصد موفقیت حمله از ۹۳٪ به ۸٪ کاهش یافته است که یعنی روش Unlearning ما به درستی کار کرده است و مدل پاکسازی شده است.

سؤال دوم: Privacy

بخش اول – زیر بخش اول

برای یک مدل $\epsilon - differentialyprivate$ با مکانیزم لاپلاس خواهیم داشت:

$$\epsilon = \frac{\Delta}{b} \quad ; \quad \Delta: sensitivity$$

درخواست متوسط درآمد جامعه

$$b_1 = \frac{\Delta_1}{\epsilon} = \frac{5000}{0.1} = 50000 = 50K$$

درخواست درآمد کل جامعه

$$b_2 = \frac{\Delta_2}{\epsilon} = \frac{50000}{0.1} = 500000 = 500K$$

بخش اول – زیر بخش دوم

$$privacy\ preserving\ 1: x_1 + n = 40K + 2K = 42000 = 42K$$

$$privacy\ preserving\ 2: x_2 + n = 20M + 5K = 20005000$$

بخش اول – زیر بخش سوم

با استفاده از فرمول استفاده شده در زیر بخش اول، می توانیم مقادیر جدید ϵ را در آن قرار دهیم تا مقادیر جدید b برای هر درخواست را محاسبه کنیم.

درخواست متوسط درآمد جامعه

$$b_1 = \frac{\Delta_1}{\epsilon} = \frac{5000}{0.05} = 100000 = 100K$$

درخواست درآمد کل جامعه

$$b_2 = \frac{\Delta_2}{\epsilon} = \frac{50000}{0.05} = 1000000 = 1M$$

بخش دوم – زیر بخش اول

با توجه به مقادیر ϵ و Δ برای هر Query، مقدار پارامتر b از رابطه زیر بدست می آید:

$$b = \frac{\Delta}{\epsilon} = \frac{1}{0.1} = 10$$

بخش دوم – زیر بخش دوم

در ابتدا تابع توزیع تجمعی (CDF) توزیع لاپلاس را بدست می آوریم:

$$f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

$$F(x) = \int_{-\infty}^{x} f(u)\,\mathrm{d}u = \begin{cases} \frac{1}{2}\exp\left(\frac{x-\mu}{b}\right) & \text{if } x < \mu \\ 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right) & \text{if } x \geq \mu \end{cases}$$
$$= \tfrac{1}{2} + \tfrac{1}{2}\,\mathrm{sgn}(x-\mu)\left(1 - \exp\left(-\frac{|x-\mu|}{b}\right)\right).$$

$$P(x > 505) = 1 - P(x > 505) = 1 - F(505)$$
$$x = 505, \mu = 500, b = 10 \rightarrow x - \mu > 0 \rightarrow F(x) = 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right)$$
$$F(505) = 0.696734 \rightarrow P(x > 505) = 0.30326$$

بخش دوم – زیر بخش سوم

$$b_i = \frac{\Delta}{\epsilon_i} = \frac{1}{\frac{0.1}{k}} = 10k$$

$$P(x > 505) = 1 - P(x > 505) = 1 - F(505)$$
$$x = 505, \mu = 500, b = 10k \rightarrow x - \mu > 0 \rightarrow F(x) = 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right)$$

$$P(x > 505) = \frac{1}{2}\exp(-\frac{5}{10k}) = \frac{1}{2}\exp(-\frac{1}{2k})$$

بخش دوم – زیر بخش چهارم

در خود درس داشتیم که اگر یک مدل $\epsilon - differentialyprivate$ داشته باشیم و یک نمونه p درصدی از آن را برداریم، آن داده یک مدل $p\epsilon - differentialyprivate$ خواهد بود. حالا اگر p درصد آن را برداریم یا اضافه کنیم به ترتیب مدل $(1+p)\epsilon - differentialyprivate$ و $(1-p)\epsilon - differentialyprivate$ خواهد بود.

$$b_i = \frac{\Delta}{(1 \pm p)\epsilon_i}$$
$$P(x > 505) = 1 - P(x > 505) = 1 - F(505)$$
$$x = 505, \mu = 500, b = 10k \rightarrow x - \mu > 0 \rightarrow F(x) = 1 - \frac{1}{2}\exp\left(-\frac{x-\mu}{b}\right)$$

$$P(x > 505) = \frac{1}{2}\exp(-\frac{5}{\frac{\Delta}{(1 \pm p)\epsilon_i}})$$

سؤال سوم: Fairness

در این سوال قصد داریم تا یک طبقه بند شامل دو کلاس طراحی کنیم که فارغ از بحث جنسیت افراد، پیش بینی نماید که فرد مورد نظر دستمزد بالاتر از 50K و یا پایین تر از 50K دریافت می نماید.

بخش اول – دیتا و ارزیابی مدل

ابتدا فایل دیتاست پیوست شده را بارگذاری می کنیم و نگاهی به ویژگی های آن می اندازیم.

	age	fnlwgt	educational-num	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income	...	occupation_Protective-serv	occupation_Sales	occupation_Tech-support	occupation_Transport-moving	relationship_Husband
0	25	226802	7	0	1	0	0	40	0	0	...	False	False	False	False	False
1	38	89814	9	1	1	0	0	50	0	0	...	False	False	False	False	True
2	28	336951	12	1	1	0	0	40	0	1	...	True	False	False	False	True
3	44	160323	10	0	1	7688	0	40	0	1	...	False	False	False	False	True
4	18	103497	10	1	0	0	0	30	0	0	...	False	False	False	False	False
5	34	198693	6	1	1	0	0	30	0	0	...	False	False	False	False	False
6	29	227026	9	0	1	0	0	40	0	0	...	False	False	False	False	False
7	63	104626	15	1	1	3103	0	32	0	1	...	False	False	False	False	True
8	24	369667	10	1	0	0	0	40	0	0	...	False	False	False	False	False
9	55	104996	4	1	1	0	0	10	0	0	...	False	False	False	False	True

شکل ۱۵- ۱۰ سطر اول دیتاست مورد استفاده

	age	fnlwgt	educational-num	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	0.855043	0.668482	1079.067626	87.502314	40.422382	0.102576	0.239282
std	13.710510	1.056040e+05	2.570973	0.352061	0.470764	7452.019058	403.004552	12.391444	0.303407	0.426649
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000
25%	28.000000	1.175505e+05	9.000000	1.000000	0.000000	0.000000	0.000000	40.000000	0.000000	0.000000
50%	37.000000	1.781445e+05	10.000000	1.000000	1.000000	0.000000	0.000000	40.000000	0.000000	0.000000
75%	48.000000	2.376420e+05	12.000000	1.000000	1.000000	0.000000	0.000000	45.000000	0.000000	0.000000
max	90.000000	1.490400e+06	16.000000	1.000000	1.000000	99999.000000	4356.000000	99.000000	1.000000	1.000000

شکل ۱۶ – برخی مشخصه های آماری ستون های عددی دیتاست

در ادامه از معیار دقت برای درصد پیش بینی های درست مدل استفاده می کنیم. همچنین از معیار های Zemel Fairness و Disparate Impact برای عادل بودن مدل استفاده می کنیم. هرچقدر معیار Zemel به صفر و معیار Disp arate به ۱ نزدیکتر باشند، مدل عادل تر است.

$$\text{Zemel Fairness} = \text{prob}\,(C = +|S = \bar{s}\,) - \text{prob}(C = +|S = s)$$

$$\text{Disparate Impact} = \frac{prob(c=+|S=s)}{prob(C=+|S=\bar{s}\,)}$$

```python
test_data = pd.concat([X_test, y_test], axis = 1)
test_data['predict'] = model.predict(X_test)
women_data = test_data[test_data['gender'] == 0]
men_data = test_data[test_data['gender'] == 1]
Zemel_Fairness = len(men_data[men_data['predict'] == men_data['income']]) / len(men_data) - len(women_data[women_data['predict'] == women_data['income']]) / len(women_data)
Disparate_Impact = (len(women_data[women_data['predict'] == women_data['income']]) / len(women_data)) /  (len(men_data[men_data['predict'] == men_data['income']]) / len(men_data)
print (f'Zemel fairness: {Zemel_Fairness}')
print(f'Disparate impact: {Disparate_Impact}')
```

شکل ۱۷ – پیاده سازی معیار های Fairness

بخش دوم – پیاده سازی مدل پایه

در این بخش در ابتدا دیتاست را به دو بخش آموزش و تست تقسیم می کنیم. سپس یک مدل RandomForestClassifier را روی داده های آموزش fit می کنیم.

```python
X = data.drop('income', axis=1)
y = data['income']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = RandomForestClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(report)
```

شکل ۱۸ – آموزش مدل RandomForsetClassifier

```
Accuracy: 0.862212516208285
              precision    recall  f1-score   support

           0       0.89      0.93      0.91     11233
           1       0.74      0.64      0.68      3420

    accuracy                           0.86     14653
   macro avg       0.82      0.78      0.80     14653
weighted avg       0.86      0.86      0.86     14653

Zemel fairness: -0.10582537751920096
Disparate impact: 1.1279354156935382
```

شکل ۱۹ – دقت مدل و معیارهای Fairness قبل از عادل کردن مدل

طبق تصاویر بالا می توانیم ببینیم که دقت مدل تقریبا برابر با ۸۶.۲۲٪ است. همچنین معیار Zemel برابر با ۰.۱- و معیار Disparate برابر با ۱.۱۲ می باشد. با توجه به این معیارها می توان گفت که مدل به خوبی می تواند درآمد افراد را پیش بینی کند اما راجع به عدالت مدل، معیارهای fairness تا حدی با مقادیری که ما از یک مدل عادل انتظار داریم فاصله دارند و انگار نسبت به زنان کمی bias دارد و درآمد آنها را بیشتر بالای 50K پیش بینی می کند.

بنظرم اگر ویژگی حساس (که در اینجا جنسیت است) را از دیتاست حذف کنیم و مدل را آموزش دهیم، تاثیر زیادی نمی تواند بر عادل شدن مدل داشته باشد. چون خیلی از ویژگی های دیگر دیتاست نسبت به ویژگی حساس جنسیت همبستگی (Correlation) دارند و بالاخره تاثیر آن در دیتاست وجود دارد و بنابراین مدل ما هنوز Unfair باقی می ماند. برای اثبات این حرف ویژگی جنسیت را از دیتاست حذف می کنیم و مدل را آموزش می دهیم و معیار ها را مشاهده می کنیم.

```
Zemel fairness: -0.1034747114701643
Disparate impact: 1.1248011716292052
```

شکل ۲۰ – معیارهای Fairness بعد از حذف ویژگی حساس از دیتاست

می توانیم ببینیم که معیارهای Fairness تغییر آنچنانی ای نکردند که درستی نظر ما را اثبات می کند.

بخش سوم – پیاده سازی مدل عادل

در این بخش برای آموزش مدل عادل، از الگوریتمی برای از بین بردن بایاس با توجه به ویژگی حساس استفاده می کنیم:

۱) اضافه کردن خروجی مدل و ماکزیمم احتمال خروجی هر فرد به دیتاست

```
train_data = pd.concat([X_train, y_train], axis = 1)
predictions = model.predict(X_train)
probs = np.max(model.predict_proba(X_train), axis = 1)
train_data['predict'] = predictions
train_data['predict_prob'] = probs
```

شکل ۲۱ – اضافه کردن خروجی مدل و ماکزیمم احتمال

۲) استخراج بخش های CP و CD

```
CP = train_data[(train_data['gender'] == 1) & (train_data['income'] == 1)].sort_values('predict_prob')
CD = train_data[(train_data['gender'] == 0) & (train_data['income'] == 0)].sort_values('predict_prob', ascending = False)
```

شکل ۲۲ – استخراج CP و CD

۳) بدست آوردن n با استفاده از فرمول زیر:

$$n = \frac{(Ss \times S\bar{s}+)-(S\bar{s}\times Ss+)}{Ss+S\bar{s}}$$

```python
women_numbers = len(train_data[train_data['gender'] == 0])
men_numbers = len(train_data[train_data['gender'] == 1])
women_high_income = len(train_data[(train_data['gender'] == 0) & (train_data['predict'] == 1)])
men_high_income = len(train_data[(train_data['gender'] == 1) & (train_data['predict'] == 1)])
n = ((women_numbers * men_high_income) - (men_numbers * women_high_income)) / (women_numbers + men_numbers)
print (f'n: {n}')
```

شکل ۲۳ – بدست آوردن مقدار n

مقدار n برابر با ۱۴۷۸.۶ می شود.

۴) کلاس های n ردیف اول هر دسته را با یکدیگر جابجا می کنیم.

```python
CP_indices = CP.index[0: round(n)]
CD_indices = CD.index[0: round(n)]
income_CP = train_data.loc[CP_indices, 'income'].values
income_CD = train_data.loc[CD_indices, 'income'].values
train_data.loc[CP_indices, 'income'] = income_CD
train_data.loc[CD_indices, 'income'] = income_CP
```

شکل ۲۴ – جابجایی n ردیف اول دو دسته

در نهایت با استفاده از دیتاست جدید مدل را آموزش می دهیم و دقت و معیارهای Fairness را برای آن مشاهده می کنیم.

```
Accuracy: 0.8569576196000819
              precision    recall  f1-score   support

           0       0.88      0.94      0.91     11233
           1       0.74      0.59      0.66      3420

    accuracy                           0.86     14653
   macro avg       0.81      0.76      0.78     14653
weighted avg       0.85      0.86      0.85     14653

Zemel fairness: -0.0760850239819546
Disparate impact: 1.091473740349702
```

شکل ۲۵ – دقت مدل و معیارهای Fairness بعد از عادل کردن مدل

همانطور که می بینیم دقت مدل طبقه بند مقدار کمی پایین تر آمده است. همچنین معیار Zemel به صفر نزدیکتر و معیار Disparate به ۱ نزدیک تر شده است که نشان می دهد مدل عادل تر شده است.

بخش چهارم – مقایسه و نتیجه گیری

همانطور که در بخش قبل دیدیم دقت مدل پایه و مدل عادل به ترتیب برابر با ۸۶.۲۲٪ و ۸۵.۶۹٪ می باشد که نشان می دهد مدل اولیه دقت بیشتری داشته است. همچنین با توجه به معیار های محاسبه شده برای Fairness می توانیم بگوییم که مدل دوم عادل تر شده است و بایاس کمتری نسبت به ویژگی حساس (جنسیت) دارد.

همانطور که می بینیم در این پروژه بعد از عادل تر شدن مدل دقت آن پایین تر آمده است که دلیل آن مصالحه (Trade-off) بین دقت و عدالت مدل است. معمولاً وقتی که تلاش می‌کنیم یک مدل را عادل تر کنیم، ممکن است دقت آن کاهش یابد. این به این دلیل است که الگوریتم‌های یادگیری ماشین معمولاً با هدف بهینه‌سازی دقت آموزش داده می‌شوند و شرایط مربوط به عدالت ممکن است نیازمند تغییراتی در توزیع داده‌ها یا معیارهای بهینه‌سازی باشند. در واقع افزودن محدودیت‌های عدالت ممکن است مدل را مجبور کند که در برخی از نمونه‌ها که به صورت ناعادلانه برچسب‌گذاری شده‌اند، دقت کمتری داشته باشد تا بتواند به عدالت بیشتری دست یابد.

معرفی یک روش دیگر برای عادل کردن طبقه بند:

یکی از متدهای محبوب برای عادل کردن مدل ها استفاده از الگوریتم "Adversarial Debiasing" است. این روش با استفاده از شبکه های عصبی مخالف (Adversarial) تلاش می کند تا بایاس های موجود در مدل را کاهش دهد.

مدل اصلی (Predictor): یک مدل یادگیری ماشین که برای انجام پیش‌بینی‌ها آموزش داده می‌شود.

شبکه مخالف (Adversary): یک شبکه عصبی که تلاش می‌کند از خروجی مدل اصلی، اطلاعات مربوط به ویژگی‌های حساس (مثل جنسیت) را پیش‌بینی کند.

مرحله ۱: آموزش مدل اصلی

مدل اصلی برای انجام پیش‌بینی‌ها بر اساس داده‌های آموزشی آموزش داده می‌شود.

هدف مدل اصلی به حداکثر رساندن دقت پیش‌بینی‌ها است.

مرحله ۲: آموزش شبکه مخالف

شبکه مخالف تلاش می‌کند از خروجی مدل اصلی، ویژگی‌های حساس را پیش‌بینی کند.

هدف شبکه مخالف به حداکثر رساندن دقت پیش‌بینی ویژگی‌های حساس است.

مرحله ۳: ترکیب مدل اصلی و شبکه مخالف

هدف اصلی این است که مدل اصلی را به گونه‌ای آموزش دهیم که شبکه مخالف نتواند ویژگی‌های حساس را به خوبی پیش‌بینی کند. این کار با استفاده از یک تابع هزینه ترکیبی انجام می‌شود که شامل دو بخش است:

بخش اول: تابع هزینه پیش‌بینی مدل اصلی (مثلاً خطای MSE).

بخش دوم: تابع هزینه شبکه مخالف که تلاش می‌کند دقت پیش‌بینی ویژگی‌های حساس را کاهش دهد.

تابع هزینه ترکیبی

$$L_{\text{combined}} = L_{\text{predictor}} - \lambda L_{\text{adversary}}$$

λ : یک پارامتر تنظیمی که اهمیت نسبی کاهش تعصب را کنترل می‌کند.

به روز رسانی پارامترها

پارامترهای مدل اصلی و شبکه مخالف به صورت متناوب به روز می‌شوند. هدف نهایی این است که مدل اصلی بتواند پیش‌بینی‌های دقیقی انجام دهد و در عین حال شبکه مخالف نتواند ویژگی‌های حساس را به خوبی پیش‌بینی کند.

بخش پنجم (امتیازی) – پیاده سازی روش پیشنهادی برای عادل کردن طبقه بند

برای هر دو مدل Predictor و Adversarial از شبکه های MLP استفاده می کنیم که در ادامه معماری آن ها را مشاهده می کنیم.

```python
class Adversary(nn.Module):
    def __init__(self):
        super(Adversary, self).__init__()
        self.fc1 = nn.Linear(1, 32)
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x
```

شکل ۲۶ – معماری مدل Adversariy

```python
class Predictor(nn.Module):
    def __init__(self, input_dim):
        super(Predictor, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x
```

شکل ۲۷ – معماری مدل Predictor

برای آموزش هر دو مدل مقدار Learning rate را برابر با ۰.۰۱ قرار می دهیم و مقدار λ در تابع هزینه ترکیبی را برابر ۰.۸ قرار داده ایم.

ابتدا صرفا مدل Predictor را آموزش می دهیم تا بتوانیم بدون اعمال روش عادل کردن، دقت و معیار های Fairness را ببینیم.

```
Accuracy: 0.8627
Zemel fairness: -0.10757654202140743
Disparate impact: 1.1300684920308173
```

شکل ۲۸ – دقت مدل و معیارهای Fairness قبل از عادل کردن مدل

در نهایت روش گفته شده را اعمال می کنیم و هر دو شبکه را همزمان آموزش می دهیم و آن ها را ترکیب می کنیم و نتیجه را مشاهده می کنیم.

```
Accuracy: 0.8596
Zemel fairness: -0.10117559170230628
Disparate impact: 1.1224803198550088
```

شکل ۲۹ – دقت مدل و معیارهای Fairness بعد از عادل کردن مدل

همانطور که می بینیم همچنان Trade-off بین دقت و معیارهای عادل بودن وجود دارد. یعنی مدل تا حدی عادل تر شده است اما دقت آن کمی کاهش یافته است. اگر بخواهیم این روش با روش قبلی مقایسه کنیم،

با استفاده از روش قبل توانستیم به مدل عادل تری برسیم، اما بنظر می رسد با تغییر پارامترهای مختلف مثل Learning rate، λ و همچنین معماری های هر دو مدل بتوانیم به مدل عادل تری برسیم تا حدی که از روش قبل نیز بهتر عمل کند.