


## Imports

```
1 import torch
2 import torch.nn as nn
3 from random import shuffle
4 import numpy as np
5 from sklearn.metrics import roc_curve, auc, confusion_matrix, roc_auc_score
6 import matplotlib.pyplot as plt
7
8 from torchvision.datasets import FashionMNIST
9 import torchvision.transforms as T
10
11 from google.colab import drive
12 from torch.utils.data import Subset, DataLoader
13 from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
14 import torch.nn.functional as F
15 from datasets import Dataset
16 import evaluate
17 from typing import List, Tuple
18 from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
19 import nltk
20 nltk.download('punkt')

1 !pip install datasets
2 !pip install evaluate
```

 [Show hidden output](#)

## Function to load fmnist dataset

```
1 fmnist_labels = {
2     0: "T-shirt/top",
3     1: "Trouser",
4     2: "Pullover",
5     3: "Dress",
6     4: "Coat",
7     5: "Sandal",
8     6: "Shirt",
9     7: "Sneaker",
10    8: "Bag",
11    9: "Ankle boot"
12 }
13
14 def load_fmnist_torch(root="./data", transform=None, download=True):
15
16     if transform == None:
17         transform = T.ToTensor()
18
19     train_set = FashionMNIST(root=root, transform=transform, download=download, train=True)
20     test_set = FashionMNIST(root=root, transform=transform, download=download, train=False)
21
22     # Each item in this dictionary is a torch Dataset object
23     # To feed the data into a model, you may have to use a DataLoader
24     return {"train": train_set, "test": test_set}
```

## SmallCNN Model

```
1 class SmallCNN(nn.Module):
2
3     def __init__(self):
4         super(SmallCNN, self).__init__()
5
6         self.layer1 = nn.Sequential(
7             nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1),
8             nn.BatchNorm2d(32),
9             nn.ReLU(),
10            nn.MaxPool2d(kernel_size=2, stride=2)
11        )
12
13        self.layer2 = nn.Sequential(
14            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
15            nn.BatchNorm2d(64),
```

```

16         nn.ReLU(),
17         nn.MaxPool2d(2)
18     )
19     self.fc1 = nn.Linear(in_features=64*6*6, out_features=600)
20     self.drop = nn.Dropout(0.25)
21     self.fc2 = nn.Linear(in_features=600, out_features=120)
22     self.fc3 = nn.Linear(in_features=120, out_features=10)
23
24     def forward(self, x):
25         out = self.layer1(x)
26         out = self.layer2(out)
27         out = out.view(out.size(0), -1)
28         out = self.fc1(out)
29         out = nn.functional.relu(out)
30         out = self.drop(out)
31         out = self.fc2(out)
32         out = nn.functional.relu(out)
33         out = self.fc3(out)
34         return out

```

```

1 # Check if CUDA is available
2 if torch.cuda.is_available():
3     device = torch.device('cuda')
4     print("CUDA available! Training on GPU.", flush=True)
5 else:
6     device = torch.device('cpu')
7     print("CUDA NOT available... Training on CPU.", flush=True)

```

🔄 CUDA available! Training on GPU.

## ✓ Problem 1

```

1 # Load FMNIST dataset
2 fmnist = load_fmnist_torch()
3 fmnist_train = fmnist['train']
4 fmnist_test = fmnist['test']
5

```

🔄 Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz>  
 Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz> to ./data/FashionMNIST  
 100%|██████████| 26.4M/26.4M [00:08<00:00, 3.17MB/s]  
 Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz>  
 Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz> to ./data/FashionMNIST  
 100%|██████████| 29.5k/29.5k [00:00<00:00, 209kB/s]  
 Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz>  
 Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz> to ./data/FashionMNIST/  
 100%|██████████| 4.42M/4.42M [00:03<00:00, 1.30MB/s]  
 Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz>  
 Downloading <http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz> to ./data/FashionMNIST/  
 100%|██████████| 5.15k/5.15k [00:00<00:00, 20.8MB/s]  
 Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

```

1 print(len(fmnist["test"]))
2 print(len(fmnist["train"]))

```

🔄 10000  
60000

```

1 # TODO: Load and train mdoel
2
3 train_loader = DataLoader(fmnist["train"], batch_size = 64, shuffle=True)
4 print(len(fmnist["train"]))
5 test_loader = DataLoader(fmnist["test"], batch_size = 64, shuffle=True)
6 print(len(fmnist["test"]))
7
8
9 images, labels = next(iter(train_loader))
10
11 # Model Instantiated:
12 model = SmallCNN()
13
14 criterion = torch.nn.CrossEntropyLoss()

```

```

15 optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
16
17 #print(len(fmnist_train))
18
19 # Train the model
20 drive.mount('/content/drive')
21 # model.train()
22 # num_epochs = 10
23 # for epoch in range(num_epochs):
24 #     for images, labels in train_loader:
25 #         images, labels = images, labels
26 #         #print(images.shape)
27 #         #print(type(labels))
28
29 #         optimizer.zero_grad()
30 #         outputs = model(images)
31 #         loss = criterion(outputs, labels)
32 #         loss.backward()
33 #         optimizer.step()
34 #         print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
35
36 # torch.save(model.state_dict(), '/content/drive/My Drive/model.pth')
37 # print("Save Completed")


```

 [Show hidden output](#)

```

1 # TO DO: Test model
2 test_loader = DataLoader(fmnist["test"], batch_size = 64, shuffle=True)
3 print(len(fmnist["test"]))
4
5 images, labels = next(iter(test_loader))
6
7 #Set the Model to Evaluation Mode
8 model.load_state_dict(torch.load('/content/drive/My Drive/model.pth'))
9 # model.eval()
10 # correct = 0
11 # total = 0
12 # with torch.no_grad():
13 #     for images, labels in test_loader:
14 #         outputs = model(images)
15 #         _, predicted = torch.max(outputs.data, 1)
16 #         #print(predicted.shape)
17 #         #predicted = predicted.unsqueeze(1)
18 #         #print("images", images.shape)
19 #         total += labels.size(0)
20 #         correct += (predicted == labels).sum().item()
21
22 # print(correct)
23 # print(total)
24 # accuracy = (correct / total) * 100
25 # print(f"Test Accuracy: {accuracy:.2f}%")
26

```

 10000  
<ipython-input-9-f028f47c26be>:8: FutureWarning: You are using `torch.load` with `weights\_only=False` (the current default behavior) which will be deprecated in a future release of PyTorch. To silence this warning, you should pass `weights\_only=True` to `torch.load` if you know you have no pickle (unpickled) data in your archive. This message can be suppressed by setting `torch.\_C.\_set\_warn\_flags("warn::load\_state\_dict")`  
model.load\_state\_dict(torch.load('/content/drive/My Drive/model.pth'))  
<All keys matched successfully>

```

1 random.seed(42)
2 torch.manual_seed(42)
3
4 fmnist_member_indices = random.sample(range(len(fmnist_train)), 500)
5 fmnist_nonmember_indices = random.sample(range(len(fmnist_test)), 500)
6
7 fmnist_member_set = Subset(fmnist_train, fmnist_member_indices)
8 fmnist_nonmember_set = Subset(fmnist_test, fmnist_nonmember_indices)
9
10 # Create DataLoaders for the subsets
11 member_loader = DataLoader(fmnist_member_set, batch_size=64, shuffle=False)
12 nonmember_loader = DataLoader(fmnist_nonmember_set, batch_size=64, shuffle=False)

```

```

1 # Run the loss based attack on the model
2 # Run the loss-based attack for the Cross-entropy loss (criterion) and report the confusion matrix: True Positives (TP),
3 # Compute the accuracy, error, and precision of the attack for the threshold T

```

```

1 def crossentropy_loss_values(model, member_loader, nonmember_loader, device='cuda'):
2
3     model.eval()
4     losses = []
5     labels = []

```

```

6     with torch.no_grad():
7         for image_data, image_labels in member_loader:
8             image_data = image_data.to(device)
9             image_labels = image_labels.to(device)
10            outputs = model(image_data)
11
12            for i in range(len(image_data)):
13                loss = F.cross_entropy(outputs[i:i+1], image_labels[i:i+1], reduction='mean')
14                losses.append(loss.item())
15                labels.append(1)
16
17
18        for image_data, image_labels in nonmember_loader:
19            image_data = image_data.to(device)
20            image_labels = image_labels.to(device)
21            outputs = model(image_data)
22
23            for i in range(len(image_data)):
24                loss = F.cross_entropy(outputs[i:i+1], image_labels[i:i+1], reduction='mean')
25                losses.append(loss.item())
26                labels.append(0)
27
28    return np.array(losses), np.array(labels)
29
30 def compute_attack_metrics(losses, true_labels, threshold):
31
32     predicted_labels = (losses <= threshold).astype(int)
33
34     # confusion matrix
35     conf_matrix = confusion_matrix(true_labels, predicted_labels)
36     tn = conf_matrix[0,0]
37     fp = conf_matrix[0,1]
38     fn = conf_matrix[1,0]
39     tp = conf_matrix[1,1]
40
41     accuracy = (tp + tn) / (tp + tn + fp + fn)
42     error = 1 - accuracy
43     precision = tp / (tp + fp) if (tp + fp) > 0 else 0
44
45     metrics = {
46         'threshold': threshold,
47         'confusion_matrix': {
48             'TP': tp,
49             'FP': fp,
50             'TN': tn,
51             'FN': fn
52         },
53         'accuracy': accuracy,
54         'error': error,
55         'precision': precision,
56     }
57
58     return metrics
59
60 losses, true_labels = crossentropy_loss_values(model, member_loader, nonmember_loader, device)
61
62 # Threshold T
63 threshold = np.percentile(losses, 50)
64 predicted_labels = (losses <= threshold).astype(int)
65 print(f"T value {threshold:.2f}:")
66
67 metrics = compute_attack_metrics(losses, true_labels, threshold)
68
69 conf_matrix = confusion_matrix(true_labels, predicted_labels)
70 print("Confusion Matrix:")
71 print(conf_matrix)
72
73 print("\nMetrics:")
74 print(f"Accuracy: {metrics['accuracy']:.2f}")
75 print(f"Error Rate: {metrics['error']:.2f}")
76 print(f"Precision: {metrics['precision']:.2f}")

```

```

➡ T value 0.01:
Confusion Matrix:
[[256 244]
 [244 256]]

```

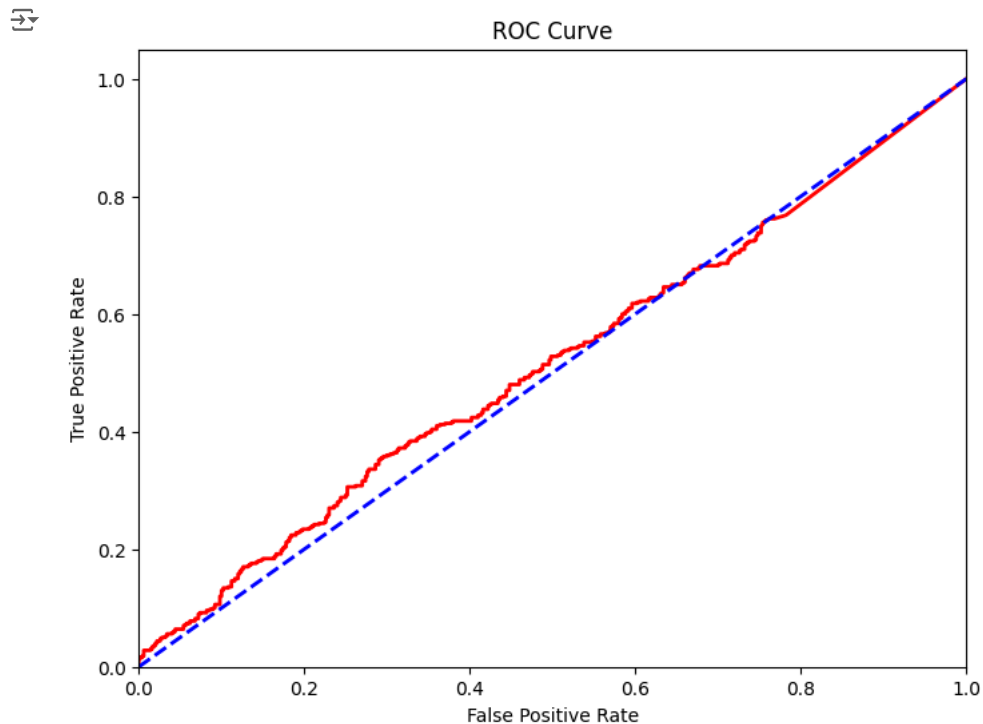
```

Metrics:
Accuracy: 0.51
Error Rate: 0.49
Precision: 0.51

```

Double-click (or enter) to edit

```
1 # TODO: Plot results
2 losses, true_labels = crossentropy_loss_values(model, member_loader, nonmember_loader, device)
3
4 # print(type(losses))
5 # print(type(true_labels))
6
7
8 # ROC curve and ROC area
9 fpr, tpr, thresholds = roc_curve(true_labels, losses, pos_label=0)
10 roc_auc = auc(fpr, tpr)
11
12 # Plot
13 plt.figure(figsize=(8, 6))
14 plt.plot(fpr, tpr, color='red', lw=2)
15 plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
16 plt.xlim([0.0, 1.0])
17 plt.ylim([0.0, 1.05])
18 plt.xlabel('False Positive Rate')
19 plt.ylabel('True Positive Rate')
20 plt.title('ROC Curve')
21 plt.show()
22
23
```



```
1 # TODO: Comment on Observations
2 #The AUC value of 0.51 suggest that the attack is extremely weak.
3 #The model's loss values don't effectively distinguish between members and non-members
4 #thus the attack is unable to accomplisih its objective. This is contrary to the LIRA paper
5 #which had metrics to show that the attack was strong.
```

## ✓ Problem 2

```
1 from transformers import AutoModelForCausalLM, AutoTokenizer, Trainer, TrainingArguments
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 import evaluate

1 # Load the dataset
2 drive.mount('/content/drive')
```

```
3 data = pd.read_csv('/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/Email_data.csv') # Replace with the actual path
4 data.head()
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True)

	Unnamed: 0	file	message
0	427616	shackleton-s/sent/1912.	Message-ID: <21013688.1075844564560.JavaMail.e...
1	108773	farmer-d/logistics/1066.	Message-ID: <22688499.1075854130303.JavaMail.e...
2	355471	parks-j/deleted_items/202.	Message-ID: <27817771.1075841359502.JavaMail.e...
3	457837	stokley-c/chris_stokley/iso/client_rep/41.	Message-ID: <10695160.1075858510449.JavaMail.e...
4	124910	germany-c/all_documents/1174.	Message-ID: <27819143.1075853689038.JavaMail.e...

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True)

```
1 # Email Text
2 data['message'].head(2)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

dtype: object

```
1 # Example Email
2 print(data['message'].iloc[0][:550])
```

Show hidden output

```
1 # TODO: Load and Fine tune model
2
3 # Model
4 pythia_model = AutoModelForCausalLM.from_pretrained(
5     "EleutherAI/pythia-70m-deduped",
6     revision="step3000",
7     cache_dir="./pythia-70m-deduped/step3000",
8 )
9
10 tokenizer = AutoTokenizer.from_pretrained(
11     "EleutherAI/pythia-70m-deduped",
12     revision="step3000",
13     cache_dir="./pythia-70m-deduped/step3000",
14 )
```

```
1 config.json: 100% 567/567 [00:00<00:00, 39.1kB/s]
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

```
1 # Split the data into train (2/3) and test (1/3) sets
2
3
4 train_data, test_data = train_test_split(data, test_size=0.33, random_state=42)
5
6
7 # Tokenize the emails
8 if tokenizer.pad_token is None:
9     tokenizer.pad_token = tokenizer.eos_token
10
11 # def tokenize_function(examples):
12 #     return tokenizer(
13 #         examples["message"],
14 #         truncation=True,
15 #         padding=True,
16 #         max_length=512,
```

```

17 # )
18
19 # # Convert to Hugging Face datasets
20 # train_dataset = Dataset.from_pandas(train_data)
21 # test_dataset = Dataset.from_pandas(test_data)
22
23 # # Tokenize the datasets
24 # train_tokenized = train_dataset.map(tokenize_function, batched=True)
25 # test_tokenized = test_dataset.map(tokenize_function, batched=True)
26
27 # # Training arguments
28 # training_args = TrainingArguments(
29 #     output_dir="./pythia-fine-tuned",
30 #     num_train_epochs=10,
31 #     per_device_train_batch_size=64,
32 #     per_device_eval_batch_size=64,
33 #     warmup_steps=500,
34 #     weight_decay=0.01,
35 # )
36
37 # # Initialize the Trainer
38 # trainer = Trainer(
39 #     model=pythia_model,
40 #     args=training_args,
41 #     train_dataset=train_tokenized,
42 #     eval_dataset=test_tokenized,
43 # )
44
45 # # Print dataset sizes
46 # print(f"Training set size: {len(train_data)}")
47 # print(f"Test set size: {len(test_data)}")
48
49 # # Print an example from the training data
50 # print("\nExample email from training set:")
51 # print(train_data['message'].iloc[0][:550])
52
53 # Save the fine-tuned model
54 # pythia_model.save_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia")
55 # tokenizer.save_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia")

```

```

591 Message-ID: <24048650.1075853975914.JavaMail.e...
664 Message-ID: <8189273.1075840887983.JavaMail.ev...
195 Message-ID: <9300560.1075851573421.JavaMail.ev...
1240 Message-ID: <7654943.1075842838727.JavaMail.ev...
1048 Message-ID: <30379452.1075854430023.JavaMail.e...
...
1130 Message-ID: <21859410.1075852282055.JavaMail.e...
1294 Message-ID: <11613616.1075849659137.JavaMail.e...
860 Message-ID: <19463108.1075852096010.JavaMail.e...
1459 Message-ID: <9982296.1075840531689.JavaMail.ev...
1126 Message-ID: <2462383.1075856344807.JavaMail.ev...
Name: message, Length: 1005, dtype: object

```

```

1 fine_tuned_model = AutoModelForCausalLM.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia")
2 fine_tuned_tokenizer = AutoTokenizer.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia")
3
4 # Report the perplexity of the original and fine-tuned model.
5
6 perplexity_metric = evaluate.load("perplexity")
7
8 pythia_model.eval()
9 # Calculate perplexity
10 ft_results = perplexity_metric.compute(
11     model_id='EleutherAI/pythia-70m-deduped',
12     predictions=test_data['message'].tolist(),
13     batch_size=8,
14     max_length=512,
15 )
16
17 print(f"Perplexity of the fine-tuned: {ft_results['mean_perplexity']}")
18
19 fine_tuned_model.eval()
20 results = perplexity_metric.compute(
21     model_id='/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia',
22     predictions=test_data['message'].tolist(),
23     batch_size=8,
24     max_length=512,
25 )
26 print(f"Perplexity of the original: {results['mean_perplexity']}")

```

 [Show hidden output](#)

```

1 original_perplexity = results['mean_perplexity']
2 print("The original perplexity", original_perplexity)
3
4 finetuned_perplexity = ft_results['mean_perplexity']
5 print("The fine tuned perplexity", finetuned_perplexity)

```

```

→ The original perplexity 39.115910374034534
   The fine tuned perplexity 29.584752663699064

```

```

1
2 train_random_samples = train_data.sample(n=100, random_state=42)
3 test_random_samples = test_data.sample(n=100, random_state=42)
4
5
6 train_messages = list(train_random_samples['message'])
7 test_messages = list(test_random_samples['message'])
8

```

## The perplexity-based loss attack

```

1 # Perform Two attacks on these samples on the fine-tuning model
2 def get_model_and_tokenizer(model_name):
3     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
4     model = AutoModelForCausalLM.from_pretrained(model_name).to(device)
5     tokenizer = AutoTokenizer.from_pretrained(model_name)
6     return model, tokenizer, device
7
8 def calculate_perplexity(text, model, tokenizer, device):
9     """Calculate perplexity score for a given text sample."""
10    encodings = tokenizer(text, return_tensors="pt")
11    input_ids = encodings.input_ids.to(device)
12
13    with torch.no_grad():
14        outputs = model(input_ids, labels=input_ids)
15        loss = outputs.loss
16
17    return torch.exp(loss).item()
18 def find_optimal_threshold(member_samples: List[str], non_member_samples: List[str], model, tokenizer, device) -> float:
19
20    # Calculate perplexities for both sets
21    member_perplexities = [calculate_perplexity(text, model, tokenizer, device)
22                           for text in member_samples]
23    non_member_perplexities = [calculate_perplexity(text, model, tokenizer, device)
24                              for text in non_member_samples]
25
26    # Try different thresholds
27    all_perplexities = sorted(member_perplexities + non_member_perplexities)
28    best_threshold = None
29    best_accuracy = 0
30
31    for threshold in all_perplexities:
32        member_correct = sum(1 for p in member_perplexities if p < threshold)
33        non_member_correct = sum(1 for p in non_member_perplexities if p >= threshold)
34
35        accuracy = (member_correct + non_member_correct) / (
36            len(member_perplexities) + len(non_member_perplexities))
37
38        if accuracy > best_accuracy:
39            best_accuracy = accuracy
40            best_threshold = threshold
41
42    return best_threshold
43
44 def predict_membership(text: str, threshold: float, model, tokenizer, device) -> Tuple[bool, float]:
45    """Predict whether a text sample is a member of the training set."""
46    perplexity = calculate_perplexity(text, model, tokenizer, device)
47    is_member = perplexity < threshold
48
49    # Calculate confidence based on distance from threshold
50    confidence = abs(perplexity - threshold) / threshold
51    confidence = min(confidence, 1.0)
52
53    return is_member, confidence, perplexity
54
55 def evaluate_attack(test_members: List[str],
56                    test_non_members: List[str],
57                    threshold: float,
58                    model,
59                    tokenizer,
60                    device) -> dict:

```



```

61 """Evaluate attack performance on test sets."""
62 results = {'true_positives': 0, 'false_positives': 0,
63           'true_negatives': 0, 'false_negatives': 0}
64
65 # Test member samples
66 for text in test_members:
67     is_member, _, _ = predict_membership(text, threshold, model, tokenizer, device)
68     if is_member:
69         results['true_positives'] += 1
70     else:
71         results['false_negatives'] += 1
72
73 # Test non-member samples
74 for text in test_non_members:
75     is_member, _, _ = predict_membership(text, threshold, model, tokenizer, device)
76     if is_member:
77         results['false_positives'] += 1
78     else:
79         results['true_negatives'] += 1
80
81 # Calculate metrics
82 total = sum(results.values())
83 accuracy = (results['true_positives'] + results['true_negatives']) / total
84
85 precision = (results['true_positives'] /
86             (results['true_positives'] + results['false_positives'])
87             if (results['true_positives'] + results['false_positives']) > 0 else 0)
88
89 recall = (results['true_positives'] /
90          (results['true_positives'] + results['false_negatives'])
91          if (results['true_positives'] + results['false_negatives']) > 0 else 0)
92
93 f1 = (2 * (precision * recall) / (precision + recall)
94      if (precision + recall) > 0 else 0)
95
96 return {
97     'accuracy': accuracy,
98     'precision': precision,
99     'recall': recall,
100    'f1_score': f1,
101    'raw_results': results
102 }


```

Double-click (or enter) to edit

```

1 pretrained_path = "EleutherAI/pythia-70m-deduped"
2 finetuned_path = "/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia"
3
4 finetuned_model, tokenizer, device = get_model_and_tokenizer(finetuned_path)
5
6 test_members = train_messages
7 test_non_members = test_messages
8
9 #Threshold
10 threshold = find_optimal_threshold(test_members, test_non_members, finetuned_model, tokenizer, device)
11 print(f"T: {threshold}")
12
13 #ASR
14 metrics = evaluate_attack(test_members, test_non_members, threshold, finetuned_model, tokenizer, device)
15 print(f"ASR: {metrics}")

```

 T: 53.587947845458984  
ASR: {'accuracy': 0.525, 'precision': 0.5144508670520231, 'recall': 0.89, 'f1\_score': 0.6520146520146519, 'raw\_results':

### Normalized perplexity attack

```

1 def get_model_and_tokenizer(pretrained_name, finetuned_path):
2
3     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
4     pretrained_model = AutoModelForCausalLM.from_pretrained(pretrained_path).to(device)
5     finetuned_model = AutoModelForCausalLM.from_pretrained(finnetuned_path).to(device)
6     finetuned_tokenizer = AutoTokenizer.from_pretrained(pretrained_path)
7
8     return pretrained_model, finetuned_model, tokenizer, device
9
10 def calculate_perplexity(text, model, tokenizer, device):
11     encodings = tokenizer(text, return_tensors="pt")
12     input_ids = encodings.input_ids.to(device)
13

```

```

14     with torch.no_grad():
15         outputs = model(input_ids, labels=input_ids)
16         loss = outputs.loss
17
18     return torch.exp(loss).item()
19
20 def calculate_normalized_perplexity(text, pretrained_model, finetuned_model, tokenizer, device):
21     pretrained_perplexity = calculate_perplexity(text, pretrained_model, tokenizer, device)
22     finetuned_perplexity = calculate_perplexity(text, finetuned_model, tokenizer, device)
23
24     # Avoid division by zero
25     if pretrained_perplexity == 0:
26         pretrained_perplexity = 1e-10
27
28     normalized_perplexity = finetuned_perplexity / pretrained_perplexity
29     return normalized_perplexity
30
31 def find_optimal_threshold(member_samples: List[str], non_member_samples: List[str], pretrained_model, finetuned_model, token
32     # Calculate normalized perplexities
33     member_perplexities = [
34         calculate_normalized_perplexity(
35             text, pretrained_model, finetuned_model, tokenizer, device
36         ) for text in member_samples
37     ]
38
39     non_member_perplexities = [
40         calculate_normalized_perplexity(
41             text, pretrained_model, finetuned_model, tokenizer, device
42         ) for text in non_member_samples
43     ]
44
45     # Find optimal threshold
46     all_perplexities = sorted(member_perplexities + non_member_perplexities)
47     best_threshold = None
48     best_accuracy = 0
49
50     for threshold in all_perplexities:
51         member_correct = sum(1 for p in member_perplexities if p < threshold)
52         non_member_correct = sum(1 for p in non_member_perplexities if p >= threshold)
53
54         accuracy = (member_correct + non_member_correct) / (
55             len(member_perplexities) + len(non_member_perplexities))
56
57         if accuracy > best_accuracy:
58             best_accuracy = accuracy
59             best_threshold = threshold
60
61     return best_threshold
62
63 def predict_membership(text, threshold, pretrained_model, finetuned_model, tokenizer, device):
64     norm_perplexity = calculate_normalized_perplexity(
65         text, pretrained_model, finetuned_model, tokenizer, device
66     )
67
68     is_member = norm_perplexity < threshold
69
70     # Calculate confidence based on distance from threshold
71     confidence = abs(norm_perplexity - threshold) / threshold
72     confidence = min(confidence, 1.0)
73
74     return is_member, confidence, norm_perplexity
75
76 def evaluate_attack(test_members: List[str],
77                     test_non_members: List[str],
78                     threshold: float,
79                     pretrained_model,
80                     finetuned_model,
81                     tokenizer,
82                     device) -> dict:
83
84     results = {
85         'true_positives': 0,
86         'false_positives': 0,
87         'true_negatives': 0,
88         'false_negatives': 0,
89         'member_perplexities': [],
90         'non_member_perplexities': []
91     }
92
93     # Test member samples
94     for text in test_members:
95         is_member, _, norm_perplexity = predict_membership(

```

```

96         text, threshold, pretrained_model, finetuned_model, tokenizer, device
97     )
98     results['member_perplexities'].append(norm_perplexity)
99     if is_member:
100         results['true_positives'] += 1
101     else:
102         results['false_negatives'] += 1
103
104 # Test non-member samples
105 for text in test_non_members:
106     is_member, _, norm_perplexity = predict_membership(
107         text, threshold, pretrained_model, finetuned_model, tokenizer, device
108     )
109     results['non_member_perplexities'].append(norm_perplexity)
110     if is_member:
111         results['false_positives'] += 1
112     else:
113         results['true_negatives'] += 1
114
115 # Calculate metrics
116 total = len(test_members) + len(test_non_members)
117 accuracy = (results['true_positives'] + results['true_negatives']) / total
118
119 precision = (results['true_positives'] /
120              (results['true_positives'] + results['false_positives'])
121              if (results['true_positives'] + results['false_positives']) > 0 else 0)
122
123 recall = (results['true_positives'] /
124           (results['true_positives'] + results['false_negatives'])
125           if (results['true_positives'] + results['false_negatives']) > 0 else 0)
126
127 f1 = (2 * (precision * recall) / (precision + recall)
128       if (precision + recall) > 0 else 0)
129
130 return {
131     'accuracy': accuracy,
132     'precision': precision,
133     'recall': recall,
134     'f1_score': f1,
135     'raw_results': results
136 }

```

```

1 # Load models and tokenizer
2 pretrained_name = "EleutherAI/pythia-70m-deduped" # base model
3 finetuned_name = "/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia"
4 pretrained_model, finetuned_model, tokenizer, device = load_models_and_tokenizer(
5     pretrained_name, finetuned_name
6 )
7
8 #Emails Inputs
9 test_members = train_messages
10 test_non_members = test_messages
11
12
13 # Threshold
14 threshold = find_optimal_threshold(test_members, test_non_members, finetuned_model, tokenizer, device)
15 print(f"T: {threshold}")
16
17
18 # Evaluate attack performance
19 metrics = evaluate_attack(test_members, test_non_members, threshold, finetuned_model, tokenizer, device)
20 print(f"ASR: {metrics}")
21

```

```

➦ /usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenizat
warnings.warn(
T: 53.587947845458984
ASR: {'accuracy': 0.525, 'precision': 0.5144508670520231, 'recall': 0.89, 'f1_score': 0.6520146520146519, 'raw_results':

```

```

1 # TODO: Plot results
2 def plot_attack_analysis(member_scores, nonmember_scores, attack_name=""):
3
4     #labels
5     y_true = np.concatenate([np.ones(len(member_scores)), np.zeros(len(nonmember_scores))])
6     y_scores = np.concatenate([member_scores, nonmember_scores])
7
8     #histograms
9     plt.figure(figsize=(12, 5))
10
11     plt.subplot(1, 2, 1)
12     plt.hist(member_scores, bins=20, alpha=0.5, label='Members', density=True, color='seagreen')
13     plt.hist(nonmember_scores, bins=20, alpha=0.5, label='Non-members', density=True, color='purple')

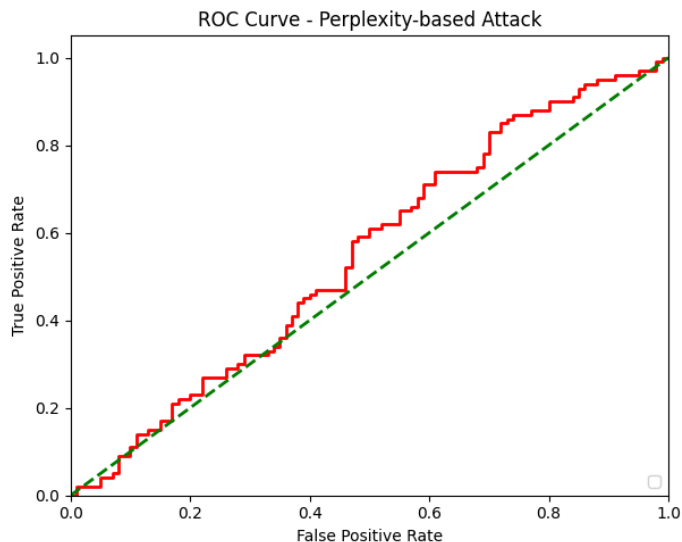
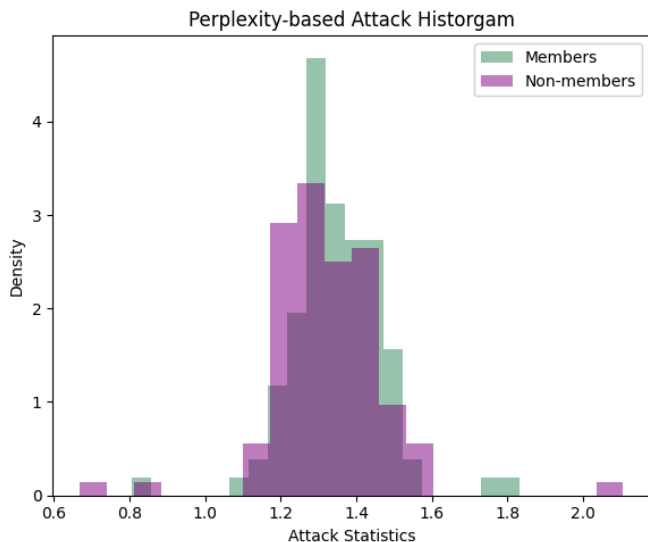
```

```

14 plt.xlabel('Attack Statistics')
15 plt.ylabel('Density')
16 plt.title(f'{attack_name} Histogram')
17 plt.legend()
18
19 # ROC curve and metrics
20 fpr, tpr, thresholds = roc_curve(y_true, y_scores)
21 roc_auc = auc(fpr, tpr)
22
23 # Find TPR at specific FPR thresholds
24 def get_tpr_at_fpr(target_fpr):
25     idx = np.searchsorted(fpr, target_fpr)
26     return tpr[idx]
27
28 tpr_01_fpr = get_tpr_at_fpr(0.001)
29 tpr_1_fpr = get_tpr_at_fpr(0.01)
30 tpr_10_fpr = get_tpr_at_fpr(0.1)
31
32 # Plot ROC curve
33 plt.subplot(1, 2, 2)
34 plt.plot(fpr, tpr, color='red', lw=2)
35 plt.plot([0, 1], [0, 1], color='green', lw=2, linestyle='--')
36
37
38 plt.xlim([0.0, 1.0])
39 plt.ylim([0.0, 1.05])
40 plt.xlabel('False Positive Rate')
41 plt.ylabel('True Positive Rate')
42 plt.title(f'ROC Curve - {attack_name}')
43 plt.legend(loc="bottom right")
44
45 plt.tight_layout()
46 plt.show()
47
48 #Print
49 print(f"\nMetrics Summary for {attack_name}:")
50 print(f"AUC Score: {roc_auc:.3f}")
51 print(f"TPR at 0.1% FPR: {tpr_01_fpr:.3f}")
52 print(f"TPR at 1% FPR: {tpr_1_fpr:.3f}")
53 print(f"TPR at 10% FPR: {tpr_10_fpr:.3f}")
54
55 # Plots
56
57 member_perplexity_list = []
58 for each in train_messages:
59     result = calculate_normalized_perplexity(each, pretrained_model, finetuned_model, tokenizer, device)
60     member_perplexity_list.append(result)
61 member_perplexities = np.array(member_perplexity_list)
62
63
64 non_member_perplexity_list = []
65 for each in test_messages:
66     result = calculate_normalized_perplexity(each, pretrained_model, finetuned_model, tokenizer, device)
67     non_member_perplexity_list.append(result)
68 non_member_perplexities = np.array(non_member_perplexity_list)
69
70
71 plot_attack_analysis(member_perplexities, non_member_perplexities, "Perplexity-based Attack")
72
73
74 plot_attack_analysis(member_perplexities, non_member_perplexities, "Normalized Perplexity Attack")

```

WARNING:matplotlib.legend:No artists with labels found to put in legend. Note that artists whose label start with an un



WARNING:matplotlib.legend:No artists with labels found to put in legend. Note that artists whose label start with an un

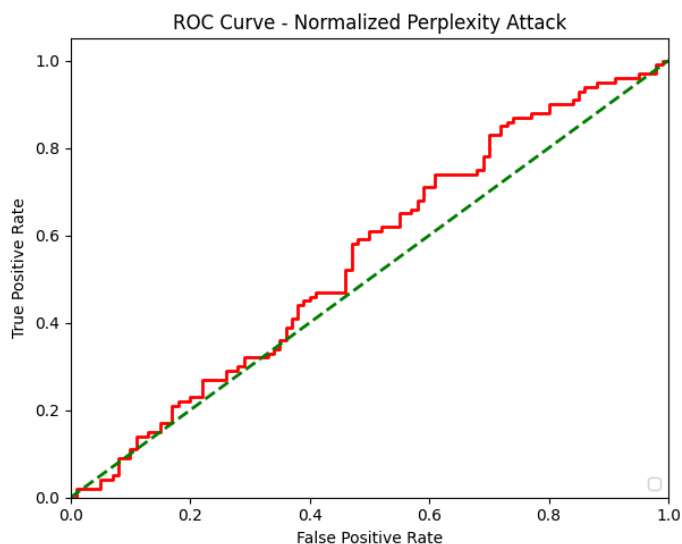
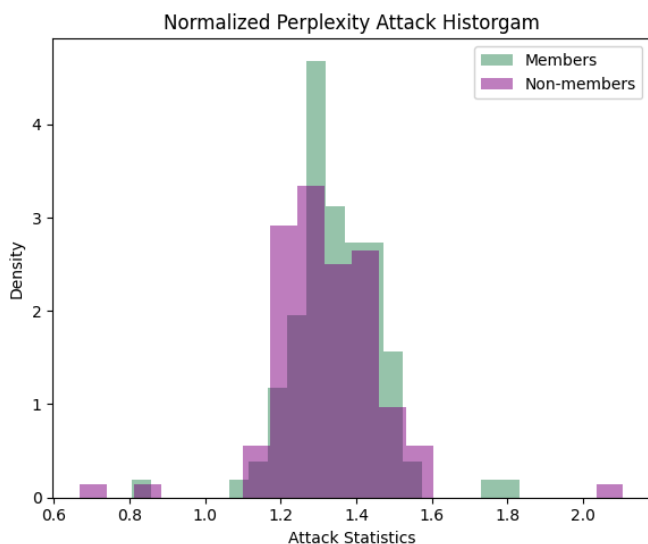
Metrics Summary for Perplexity-based Attack:

AUC Score: 0.550

TPR at 0.1% FPR: 0.000

TPR at 1% FPR: 0.000

TPR at 10% FPR: 0.090



Metrics Summary for Normalized Perplexity Attack:

AUC Score: 0.550

TPR at 0.1% FPR: 0.000

TPR at 1% FPR: 0.000

TPR at 10% FPR: 0.090

1 # TODO: Comment on Observations

2

3 # The normalized based attack is a strong attack that reduces  
4 # the impact of corpus token variations for our model. However, the  
5 # results show that there was no difference in the strength of the  
6 # attack between the perplexity and the normalized attack.

## ✓ Problem 3

1 import random

2

```

3 fine_tuned_model = AutoModelForCausalLM.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_model")
4 fine_tuned_tokenizer = AutoTokenizer.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_tokenizer")

1
2 fine_tuned_model = AutoModelForCausalLM.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_model")
3
4 vocab_size = fine_tuned_tokenizer.vocab_size
5 random_token = random.randint(0, vocab_size-1)
6
7
8 input_ids = torch.tensor([[random_token]]).to(fine_tuned_model.device)
9
10 # Generate text
11 with torch.no_grad():
12     outputs = fine_tuned_model.generate(
13         input_ids,
14         max_length=1000,
15         do_sample=True,
16         temperature=1.0,
17         pad_token_id=tokenizer.pad_token_id,
18         bos_token_id=tokenizer.bos_token_id,
19         eos_token_id=tokenizer.eos_token_id
20     )
21
22 # Generated
23
24 generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
25 print(generated_text)

```

 [Show hidden output](#)

```

1 test_members = train_messages
2 test_non_members = test_messages
3
4 def generate_email(model, tokenizer, max_length=1000):
5
6     vocab_size = tokenizer.vocab_size
7     random_token = random.randint(0, vocab_size-1)
8
9     # Generate text
10    input_ids = torch.tensor([[random_token]]).to(model.device)
11    with torch.no_grad():
12        outputs = model.generate(
13            input_ids,
14            max_length=max_length,
15            do_sample=True,
16            temperature=1.0,
17            pad_token_id=tokenizer.pad_token_id,
18            bos_token_id=tokenizer.bos_token_id,
19            eos_token_id=tokenizer.eos_token_id
20        )
21
22    return tokenizer.decode(outputs[0], skip_special_tokens=True)
23
24 def bleu_score(reference, candidate):
25     reference_tokens = nltk.word_tokenize(reference.lower())
26     candidate_tokens = nltk.word_tokenize(candidate.lower())
27     references = [reference_tokens]
28     smoothing = SmoothingFunction().method1
29
30 def plot_bleu_scores(model, tokenizer, in_set_emails, out_set_emails, num_samples=10):
31
32     in_set_emails = test_members[:num_samples]
33     out_set_emails = test_non_members[:num_samples]
34
35     bleu_scores_in = []
36     bleu_scores_out = []
37
38     for i, ref_email in enumerate(in_set_emails):
39         generated = generate_email(model, tokenizer)
40         bleu = bleu_score(ref_email, generated)
41         bleu_scores_in.append(bleu)
42         print(f"In-set email {i+1} BLEU score: {bleu:.4f}")
43
44     for i, ref_email in enumerate(out_set_emails):
45         generated = generate_email(model, tokenizer)
46         bleu = bleu_score(ref_email, generated)
47         bleu_scores_out.append(bleu)
48         print(f"Out-of-set email {i+1} BLEU score: {bleu:.4f}")
49
50     # Create histogram
51     plt.figure(figsize=(10, 5))

```

```

51 plt.figure(figsize=(10, 6))
52
53 # Histograms
54 plt.hist(bleu_scores_in, alpha=0.5, label='In Training Set', color='seagreen', bins=20)
55 plt.hist(bleu_scores_out, alpha=0.5, label='Out of Training Set', color='red', bins=20)
56
57 plt.xlabel('BLEU Score')
58 plt.ylabel('Frequency')
59 plt.title('BLEU Scores for Generated Emails')
60 plt.legend()
61 plt.grid(True, alpha=0.3)
62
63 plt.tight_layout()
64 plt.show()
65
66 return bleu_scores_in, bleu_scores_out
67
68 # model and tokenizer
69 fine_tuned_model = AutoModelForCausalLM.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_
70 tokenizer = AutoTokenizer.from_pretrained("/content/drive/MyDrive/CS6983_GenAI_Folder/Homework_3/fine_tuned_pythia")
71
72 in_set_emails = test_members
73 out_set_emails = test_non_members
74
75 bleu_scores_in, bleu_scores_out = plot_bleu_scores(
76     fine_tuned_model,
77     tokenizer,
78     in_set_emails,
79     out_set_emails
80 )

```

 [Show hidden output](#)

Citation: <https://thepythoncode.com/article/bleu-score-in-python>