Homework 5

18739 Spring 2020

Coding due: Sunday May 10, 21:00 Pacific / Monday May 11 00:00 Eastern
Written due: Friday May 15, 10:20am Pacific/1:20pm Eastern
No late days can be used for the coding portion of the homework.

This homework contains three sections. Full credit is earned by completing two.

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to CMU's Policy on Cheating and Plagiarism (`https://www.cmu.edu/policies/`). Any code included in your submission will be examined under Similarity Check automatically.

Late Day Policy

Your solutions should be uploaded to Gradescope (`https://www.gradescope.com/`) by the deadline. You have 8 late days in total to spend during the whole course but no more than 3 days can be applied to any one homework.

Written Exercises

If a homework contains written exercises, you will need to submit a `.pdf` file. Please indicate your Andrew ID and name on the first page. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: `https://www.overleaf.com`) and Markdown are two recommended languages to generate the clean layout but you are free to use other software. You will need to submit the written part to HomeworkX-PDF on Gradescope, separately from you code submission.

Coding Exercises

Submit the coding portions of homeworks to Gradescope according to the following procedure:

1. Compress your `.py` and other requested files into a zip file called `submission.zip` (No other name is allowed otherwise Gradescope will fail to grade your submission). Do NOT put all files into a folder first and compress the folder. **Create the zip file directly on the top of all required files**. Each homework will specify the `.py` and other files expected to be included in the zip.
2. Submit `submission.zip` to Gradescope. Your code implementation will be auto-graded by Gradescope and you have an infinite number of submissions before the deadline but only the last submission will count towards the grading. Allow the server to take some time (around 2 min) for execution, which means that do not submit until the last moment when everyone is competing for resources.

Contents

Setup

I Membership Inference*[10 points]*

In this part of the assignment, you will be implementing the black-box shadow model membership inference attack of Shokri et al. [3][1] also described in Nasr et al. [2], the survey we read for class. A membership inference is the scenario in which an adversary seeks to determine if a given instance was used for training a given model. We will consider the "black-box" variant of membership inference: the adversary does not have direct access to the model, they only have query access in that they can ask the model to make predictions on chosen instances and observe the outcome. More difficult variants of this scenario only give the adversary the prediction class for their chosen queries but we will work in the easier case where the adversary is given the output probability vector. We will see that despite not having direct access, the attacker can still achieve success. White-box vs. black-box is a common dichotomy in security where it is generally recognized that hiding systems or models (that is, black-box scenarios) from adversaries is typically not an effective form of defense. This is true in a wider sense as exemplified by the concept of "security through obscurity", the usually ineffective approach for securing systems by hiding their internal operation.

White-box Membership Inference Let $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$ be a dataset partitioned into subsets T_0, T_1. Let f be a model trained on T_1 where $\hat{\mathbf{y}} = f(\mathbf{x})$, the model outputs, are probability distributions over a set of classes. An adversary is a procedure, that given an instance x from either T_0 or T_1 outputs either 0 or 1 indicating the guess of which subset the instance comes from (that is, whether it is a training instance). The adversary also has:

- Query access to f's probability vectors. In another words, given any chosen instance x', the attacker can obtain $\hat{y'} = f(x')$. They are not limited in how many queries to make (though typically this is a point of comparison for black-box attacks).
- a shadow subset $\mathcal{S} \subseteq \mathcal{D}$ independent of the training set: $\mathcal{S} \cap \mathcal{T}_1 = \emptyset$ [2].
- Knowledge of the format of the inputs and outputs of the targeted model, including their number and the range of values they can take.

[1] https://arxiv.org/abs/1610.05820

[2] The attack will work better if $\mathcal{S}$ happens to have some overlap with $\mathcal{T}_1$.

- Knowledge of the type and architecture of the machine learning model, as well as the training algorithm[3].

Let b be a fair coin flip in range $\{0,1\}$ and let x be a sample uniformly from T_b. The probability that the attacker outputs b is a measure of adversary success. We will write $A_{f,S}$ to denote the adversary with the aforementioned access, then:

$$\text{success}\left(A_{f,S}\right) \overset{\text{def}}{=} \Pr_b\left[A_{f,S}(x) = b \mid x \in T_b\right]$$

Shadow Model Attack In the shadow model attack, the attacker uses the shadow dataset S to create a predictor for the question "was x used to train this model?". The process has two main steps: 1) train predictors for known splits of S and collect their predictions on instances in S and out of S into a synthesized dataset for the membership inference task, 2) train a model over the synthesized dataset. We elaborate below.

1. Repeat a number of times the process:
    - Split S into two disjoint subsets S_{in} and S_{out} and train a shadow model g using only S_{in}. We will use this model to generalize output behaviour of models on training instances vs. non-training instances.
    - Synthesize two datasets A_{in} and A_{out}. The features in A_{in} are the ground truth label y and the g-predicted class distribution for each instance $(x,y) \in S_{in}$ while A_{out} has the same but for each instance of S_{out}. The target class in these is an indicator whether the given instances are from S_{in} (indicated by 1) or from S_{out} (indicated by 0).

      We now have $(y,\hat{y},1) \sim A_{in}$ where $\hat{y} = g(x)$ for $(x,y) \in S_{in}$ and $(y,\hat{y},0) \sim A_{out}$ where $\hat{y} = g(x)$ for $(x,y) \in S_{out}$
2. Combine all of the produced A_{in} and A_{out} sets into $(y,\hat{y},b) \sim A$.
3. Train an attack model $m : (y,\hat{y}) \mapsto b$ using A to predict training set membership b.

Now given an instance (x,y), we can use m to predict $b = m(y, g(x))$ telling us whether $(x,y) \in S_{in}$, the training set of the shadow model g. Interestingly, we can also $b = m(y, f(x))$ to determine whether $(x,y) \in T_1$, the training set of the model we are attacking, f!

For the implementation, we are additionally asking you to create not one attack model m but rather a family m^y of models where m^y is specialized to instances of only class y. Thus to make membership guess for instance (x,y), we look up the prediction $b = m^y(y, f(x))$.

Implementation For the attack models, m^y, you can use the following architecture though we encourage you to experiment:

- Let C be the number of classes of the target model (C can be obtained via `shadow_labels.max() + 1`). The input of m has shape `(None, 2C)` ($2C$ because m takes both the predicted distribution over labels and the one-hot true label.
- One hidden layer of shape `(None, 4C)` with a ReLU activation.
- Output is of shape `(None, 1)` with a sigmoid activation.

[3]This is not required in the original paper but is assumed in this homework for simplicity.

- Binary crossentropy for the loss function.

Coding Exercise 1 ~~[4 points] *Implement* `synthesize_attack_data` *in* `hw5_part1.py`*. This corresponds to the first two points of the algorithm above.*~~

Coding Exercise 2 ~~[4 points] *Implement* `build_attack_models` *in* `hw5_part1.py`*. This is the last point of the algorithm above.*~~

Coding Exercise 3 ~~[2 points] *Implement* `evaluate_membership` *in* `hw5_part1.py`*. This method applies the attack models to a dataset to make their membership guesses.*~~

[bonus points]* Find bugs in our solution. Submit them via the Piazza post dedicated to the bugs in this homework. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

The starter code in `hw5_part1.py` includes an invocation of the exercises on a model for CIFAR. You can use it to test your solutions.

Tips:

- The `build_attack_model` function takes the target model, shadow data and labels (S), and the number of shadow models to use for the attack. When splitting the shadow data into S_{in} and S_{out}, you should use the `DataSplit` class (found in `hw5_part1_utils.py`). The constructor for `DataSplit` takes the labels of the dataset you would like to split, and a seed (index from 0 to `num_shadow_models`). The resulting object has two attributes, `in_idx` and `out_idx`, which give the list of indices into the original data that form the "in" and "out" datasets. For example, with a `DataSplit` object, `split`, S_{in} can be obtained via `shadow_data[split.in_idx]`.

- The `evaluate_membership` function takes the attack models returned by `build_attack_models`, the target model's predictions on a set of points, and the true labels for the same set of points. Recall that, while the attack model takes both the predicted labels and the one-hot true labels as input, there is also a separate attack model for each class.

Written Exercise 4 [3 points]

What is the relationship between overfitting of the attacked model $f(x)$ and the resulting attack success? Evaluate and report results of both the shadow models attack and the naive baseline attack, one which always guesses that correctly predicted instances were in training, and incorrectly guessed instances were not. As part of this exercise you should analyze the attacks on CIFAR models that do not overfit as much as the one shown in the starter code.

Your analysis of the results should include a measure of overfitting, attacker success rate, and a graphic relating the two. You can find better models that do not overfit as much in Homework 2. You can also analyze models that are reasonable unless they are trained for too many epochs at which point they start to overfit.

Written Exercise 5 [4 points] *For this exercise chose and answer **one** of the options below.*

(Option A) *You may have noticed that shadow models in the homework were of the same architecture as the attacked model $f(x)$. Can an attack be carried out if the attacker does not have the exact architecture of the model being attacked? Investigate this problem by evaluating attacks in which the shadow models deviate* *from the attacked model's architecture. You should include at least 3 types of shadow models differing from the attacked model, arranged on a scale of how similar or different they are from the attacked model. Also include a graphic relating attack success vs. the shadow model closeness/difference.*

(Option B) *Another assumption we made in the homework is that the data used for training shadow models is independent of the data used to train the attacked model $f(x)$. Investigate attack success as a function of the intersection between shadow data and training data. Your should include a graphic that presents a measure of shared data on one axis and the resulting attacker success on the other. Investigate at least 3 levels of overlap outside of no overlap.*

Written Exercise 6 [3 points]

1. *Can the shadow model attack be applied to a both a white-box and black-box scenario? Why or why not?*
2. *How is the success rate of Shadow Model Attack related to the in-training-set/out-of-training set balance of its training data?*
3. *Can regularization help mitigate the risk of being membership inferred. Why or why not?*

II Mitigating Bias with Adversarial Learning *[10 points]*

In this part of the assignment, you will be implementing the GAN-like fair training routine of Zhang et al. [4] described in lecture. You will be implementing two variants of the training procedure:

1. A variant that aims to achieve *demographic parity* which we will identify with the condition

$$\Pr[\hat{Y} = 1 \mid Z = 0] = \Pr[\hat{Y} = 1 \mid Z = 1]$$

   where prediction $\hat{y} = 1$ is positive and $z \in \{0, 1\}$ are two groups (genders, etc.).

2. A variant that aims to achieve *equality of opportunity* for positive ground truth, identified by the condition

$$\Pr[\hat{Y} = 1 \mid Y = 1, Z = 0] = \Pr[\hat{Y} = 1 \mid Y = 1, Z = 1]$$

   where $y = 1$ indicates positive ground truth.

Your solution will fill in the holes of `hw5_part2.py`. The starter code also includes invocations based on the UCI Adult dataset[4]. We split the dataset for you into demographics features matrix X, class y (1 indicates income >= 50k, the outcome we will consider positive), and a group (gender) attribute z (1 indicates male).

The idea of the debiased training procedure is that we can mitigate bias in our classifier via a competition between an *adversary* and the *classifier*. The classifier wants to predict the correct output, while also keeping the adversary from predicting the protected attribute. Meanwhile, the adversary wants to predict the protected attribute. We give the adversary different information depending upon which fairness objective to achieve. For demographic parity, the adversary only gets the classifier prediction while for equality of opportunity, the adversary gets both the correct class and classifier prediction.

[4]http://archive.ics.uci.edu/ml/datasets/Adult

II-1 Demographic Parity *[5 points]*

Coding Exercise 7 ~~[1 points] *Implement the `evaluate_dem_parity` method in `hw5_part2.py`. This method measures the demographic parity of a given model. It should return a tuple with two values: (1) the probability that the prediction for group 0 is 1, and (2) the probability that the prediction for group 1 is 1. Demographic parity is achieved if these two values equal.*~~

Coding Exercise 8 ~~[4 points] *Implement the `train_dem_parity` method of the `AdversarialFairModel` class in `hw5_part2.py`.*~~

[bonus points]* Find bugs in our solution. Submit them via the Piazza post dedicated to the bugs in this homework. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

The `train_dem_parity` method returns nothing but should update `self.classifier` by training it according to the following procedure:

1. Create the adversary and connect it to the classifier's outputs.
2. Create operations for the loss, gradients, and parameter updates of the adversary.
3. Create operations for the loss, modified gradients, and parameter updates of the classifier.
4. For each epoch, train the adversary, then the classifier on all batches (on epoch t, use a learning rate of $1/t$ and an α of $\sqrt{t}$). This will results in reduction of learning rate with epochs and a slowly increasing debiasing strength.

The adversary network should be simply a linear model with a single sigmoid output (as the protected attribute is binary).

II-2 Equality of Opportunity *[5 points]*

Coding Exercise 9 ~~[1 points] *Implement the `evaluate_eq_op` method in `hw5_part2.py`. This computes a measure of equality of opportunity for a given model. We will focus on the positive ground truth (1). It should return a tuple with two values: (1) the probability that the prediction for group 0 is 1, given that the ground truth is 1, and (2) the probability that the prediction for group 1 is 1 given that the ground truth is 1. Equality of opportunity (for positive ground truth) is achieved if these values equal.*~~

Coding Exercise 10 ~~[4 points] *Implement the `train_eq_op` method of the `AdversarialFairModel` in `hw5_part2.py`. The general operation of this method as described in the exercise for demographic parity.*~~

[bonus points]* Find bugs in our solution. Submit them via the Piazza post dedicated to the bugs in this homework. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

You can test your implementation with the last part of the starter code `hw5_part2.py`.

Written Exercise 11 [3 points] *Explain the main difference among demographic parity, equality of odds and equality of opportunity. Describe situations where each of these would be most appropriate. This can involve subjective arguments.*

Written Exercise 12 [3 points] *Notice that the debiasing approach in the work was indirect in that the performance of the adversary was used as a proxy for biased predictions of the classifier.*

Could we have incorporated the disparity in positive outcome probabilities directly into the classifier loss and thus not have to deal with an adversary? Describe how to frame this approach so it can be solved using gradient descent with a single loss. We are NOT asking for an implementation.

What are some reasons for and settings where this direct approach might not be so easy to train?

Written Exercise 13 [4 points] *For this exercise, choose and answer* ***one*** *of the question options. The solution in the starter code can be run in a jupyter notebook to provide additional information during training (see `hw5_part2.ipynb`). You might find that information helpful in this exercise.*

(Option A) *The starter code evaluates the debiasing schemes on group unbalanced data in that there are more males in the dataset than females. What impact does the dataset balance have on the debiasing method and its results? As part of your answer experiment with debiasing balanced datasets and datasets even less balanced than the provided one. Report the results. Pre-processing for balance is already included in the starter code's `main` section.*

(Option B) *The starter code provides the debiasing method a dataset in which the group attribute has been removed from features X. The classifier therefore cannot directly use gender in its prediction. In context of the debiasing methodology of this homework, do you think it is a good idea to remove the group attribute or to keep it? Justify your answer. As part of your answer should present comparisons of results of the methods applied on features X with gender included or not. You would have to preprocess X to include z before you apply the debiased training method.*

III Mitigating Bias in Word Embeddings *[10 points]*

In this part you will implement the word-embedding debiasing technique of Bolukbasi et al. [1]. In the paper, they refer to this technique as "hard-debiasing" or "neutralize and equalize". **Please refer to Section 6 of the paper [1] for more details.**

Before you get started Install extra packages (`json` and `gensim` may be necessary) and download the word2vec word embedding. You will then need to unzip the data and place it in the `data` folder:

```
pip install gensim json

wget https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz

gunzip GoogleNews-vectors-negative300.bin.gz

mv GoogleNews-vectors-negative300.bin data
```

We have provided the class that loads in the embedding in the starter code in `hw5_part3.py`. There are four other json/txt files you need to use in your implementation, which are also loaded for you.

- `definitional_pairs.json`: Definitional pairs used to find the gender dimension.
- `gender_specific_full.json`: All words that you should **not** debias.
- `equalize_pairs.json`: Word pairs to equalize so that they are equal-distant to the debiased gender-neutral words.
- `questions-word.txt`: An evaluation dataset to test the performance of word embeddings. Each line contains an analogy, in some lines it has subcategory information with colon that you can ignore.

III-1 Debiasing word embeddings

Coding Exercise 14 ~~[3 points] *Complete the method `identify_gender_subspace` that extracts the gender direction (1 dimension). This is done by performing PCA on the gender definitional words. You can use `np.linalg.svd` for PCA. No other packages (such as sklearn) are allowed in this part.*~~

Coding Exercise 15 ~~[3 points] *Complete the method `neutralize` that project all gender-neutral words (the complement of gender-specific words) away from the gender axis.*~~

Coding Exercise 16 ~~[4 points] *Complete the method `equalize` that makes sure both words within each equalized pair are equal-distant to the gender-neutral words.*~~

[bonus points]* Find bugs in our solution. Submit them via the Piazza post dedicated to the bugs in this homework. Bonus point award depends on severity. Only the first mention of each bug will be awarded points.

To evaluate the bias and utility of the original and debiased embedding, you can use `compute_analogy`, which computes the fourth word given three words in an analogy. It is done by finding a word (different from the given three words) that is closest (in terms of inner product) to the fourth vertex in the parallelogram where other given words occupy three vertices. The end of `hw5_part3.py` also includes an invocation of the requested methods you can test your solution with.

Written Exercise 17 [3 points] *In the `main` section of `hw3_part3.py`, we have provided evaluation code on two tasks, before and after debiasing. Please run the script and answer the following questions:*

1. *How many of the appropriate analogies stay the same before and after debiasing?*
2. *Rate the (potentially) biased analogies as either problematic or OK before and after debiasing. You can make your choice subjectively. Does debiasing help?*
3. *How does the evaluation accuracy change on the `questions-words.txt`?*

Written Exercise 18 [3 points] *Why do we need the equalize step? What would be the consequence of just neutralizing with out equalizing? Run the debiasing procedure without equalizing and describe the results.*

Written Exercise 19 [4 points] *To evaluate the debiasing effect on gender-profession bias quantitatively, one metric we can use is to compute the coordinates of profession words projected onto gender direction. We provided the file `profession_words.json` for this purpose. Compute the average bias of these professions by averaging the absolute values of the coordinates for all profession words. (Some of these professions words may not exist in the 300,000 curtailed vocabulary, you can skip them and only keep the ones in the vocabulary).*

- *How does the average bias change before and after debiasing?*
- *What are some of the profession words with large bias even after debiasing? What semantic features do they have in common? What is one way to address the remaining bias in these words?*

Submission

As part of the submission, you should include all of the code and notebooks you have used to carry out the various experiments. Make sure you annotate in your code the exercise being addressed and make a note in each write-up where to find the code addressing it.

References

[1] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Advances in neural information processing systems*, pages 4349–4357, 2016. URL `https://arxiv.org/abs/1607.06520`.

[2] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910*, 2018. URL `https://arxiv.org/pdf/1812.00910`.

[3] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017. URL `https://arxiv.org/abs/1610.05820`.

[4] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. *CoRR*, abs/1801.07593, 2018. URL `http://arxiv.org/abs/1801.07593`.