COMS 4771 HW2 (Spring 2023)

Due: Sat Mar 04, 2023 at 11:59pm

This homework is to be done **alone**. No late homeworks are allowed. To receive credit, a typesetted copy of the homework pdf must be uploaded to Gradescope by the due date. You must show your work to receive full credit. Discussing possible approaches for solutions for homework questions is encouraged on the course discussion board and with your peers, but you must write your own individual solutions and **not** share your written work/code. You must cite all resources (including online material, books, articles, help taken from/given to specific individuals, etc.) you used to complete your work.

1 Designing socially aware classifiers

Traditional Machine Learning research focuses on simply improving the accuracy. However, the model with the highest accuracy may be discriminatory and thus may have undesirable social impact that unintentionally hurts minority groups[1]. To overcome such undesirable impacts, researchers have put lots of effort in the field called Computational Fairness in recent years.

Two central problems of Computational Fairness are: (1) what is an appropriate definition of fairness that works under different settings of interest? (2) How can we achieve the proposed definitions without sacrificing on prediction accuracy?

In this problem, we will focus on some of the ways we can address the first problem. There are two categories of fairness definitions: individual fairness[2] and group fairness[3]. Most works in the literature focus on the group fairness. Here we will study some of the most popular group fairness definitions and explore them empirically on a real-world dataset.

Generally, group fairness concerns with ensuring that group-level statistics are same across all groups. A group is usually formed with respect to a feature called the **sensitive attribute**. Most common sensitive features include: gender, race, age, religion, income-level, etc. Thus, group fairness ensures that statistics across the sensitive attribute (such as across, say, different age groups) remain the same.

For simplicity, we only consider the setting of binary classification with a single sensitive attribute. Unless stated otherwise, we also consider the sensitive attribute to be binary. (Note that the binary assumption is only for convenience and results can be extended to non-binary cases as well.)

Notations:

Denote $X \in \mathbb{R}^d$, $A \in \{0,1\}$ and $Y \in \{0,1\}$ to be three random variables: non-sensitive features of an instance, the instance's sensitive feature and the target label of the instance respectively, such that $(X, A, Y) \sim \mathcal{D}$. Denote a classifier $f : \mathbb{R}^d \to \{0,1\}$ and denote $\hat{Y} := f(X)$.

[1] see e.g. **Machine Bias** by Angwin et al. for bias in recidivism predication, and **Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification** by Buolamwini and Gebru for bias in face recognition

[2] see e.g. **Fairness Through Awareness** by Dwork et al.

[3] see e.g. **Equality of Opportunity in Supervised Learning** by Hardt et al.

For simplicity, we also use the following abbreviations:

$$\mathbb{P} := \mathbb{P}_{(X,A,Y)\sim D} \qquad \text{and} \qquad \mathbb{P}_a := \mathbb{P}_{(X,a,Y)\sim D}$$

We will explore the following three fairness definitions:
- *Demographic Parity (DP)*

$$\mathbb{P}_0[\hat{Y} = \hat{y}] = \mathbb{P}_1[\hat{Y} = \hat{y}] \qquad \forall \hat{y} \in \{0, 1\}$$

(equal positive rate across the sensitive attribute)

- *Equalized Odds (EO)*

$$\mathbb{P}_0[\hat{Y} = \hat{y} \mid Y = y] = \mathbb{P}_1[\hat{Y} = \hat{y} \mid Y = y] \qquad \forall \hat{y},\ y \in \{0, 1\}$$

(equal true positive- and true negative-rates across the sensitive attribute)

- *Predictive Parity (PP)*

$$\mathbb{P}_0[Y = y \mid \hat{Y} = \hat{y}] = \mathbb{P}_1[Y = y \mid \hat{Y} = \hat{y}] \qquad \forall \hat{y},\ y \in \{0, 1\}$$

(equal positive predictive- and negative predictive-value across the sensitive attribute)

Part 0: The basics.

(i) Why is it not enough to just remove the sensitive attribute A from the dataset to achieve fairness as per the definitions above? Explain with a concrete example.

Part 1: Sometimes, people write the same fairness definition in different ways.

(ii) Show that the following two definitions for *Demographic Parity* is equivalent under our setting:

$$\mathbb{P}_0[\hat{Y} = 1] = \mathbb{P}_1[\hat{Y} = 1] \iff \mathbb{P}[\hat{Y} = 1] = \mathbb{P}_a[\hat{Y} = 1] \qquad \forall a \in \{0, 1\}$$

(iii) Generalize the result of the above equivalence and state an analogous equivalence relationship of two equality when $A \in \mathbb{N}$, and $\hat{Y} \in \mathbb{R}$.

Part 2: In this part, we will explore the COMPAS dataset. The task is to predict two year recidivism. Download the COMPAS dataset posted on the class discussion board. In this dataset, the target label Y is `two_year_recid` and the sensitive feature A is `race`.

(iv) Develop the following classifiers: (1) MLE based classifier, (2) nearest neighbor classifier, and (3) naïve-bayes classifier, for the given dataset.

For MLE classifier, you can model the class conditional densities by a Multivariate Gaussian distribution. For nearest neighbor classifier, you should consider different values of k and the distance metric (e.g. L_1, L_2, L_∞). For the naïve-bayes classifier, you can model the conditional density for each feature value as count probabilities.

(you may use builtin functions for performing basic linear algebra and probability calculations but you should write the classifiers from scratch.)

You must submit your code to receive full credit.

(v) Which classifier (discussed in previous part) is better for this prediction task? You must justify your answer with appropriate performance graphs demonstrating the superiority of one classifier over the other. Example things to consider: how does the training sample size affects the classification performance.

(vi) To what degree the fairness definitions are satisfied for each of the classifiers you developed? Show your results with appropriate performance graphs.

For each fairness measure, which classifier is the most fair? How would you summarize the difference of these algorithms?

(vii) Choose any one of the three fairness definitions. Describe a real-world scenario where this definition is most reasonable and applicable. What are the potential disadvantage(s) of this fairness definition?

(You are free to reference online and published materials to understand the strengths and weaknesses of each of the fairness definitions. Make sure cite all your resources.)

(viii) [Optional problem, will not be graded] Can an algorithm simultaneously achieve high accuracy and be fair and unbiased on this dataset? Why or why not, and under what fairness definition(s)? Justify your reasoning.

2 Data dependent perceptron mistake bound

In class we have seen and proved the perceptron mistake bound which states that the number of mistakes made by the perceptron algorithm is bounded by $\left(\frac{R}{\gamma}\right)^2$.

(i) Prove that this is tight. That is, give a dataset and an order of updates such that the perceptron algorithm makes exactly $\left(\frac{R}{\gamma}\right)^2$ mistakes.

Interestingly, although you have hence proved that the perceptron mistake bound is tight, this does not mean that it cannot be improved upon. The claimed "tightness" of the bound simply means that there exists a "bad" case which achieves this worst case bound. If we make some extra assumptions, these bad cases might be ruled out and the worst case bound could significantly improve. In ML, it is common to look at how extra assumptions can help improve such bounds [4]. This is what we will do in this problem.

As in class let $S = \{(x_i, y_i)\}_{i=1}^n$ be our (linearly separable) dataset where $x_i \in \mathbb{R}^D$ and $y_i \in \{-1, 1\}$. Also let w^* be the unit vector defining the optimal linear boundary with the optimal margin γ (i.e. $\forall i, y_i(w^* \cdot x_i) \geq \gamma$). Finally, let $R = \max_{x_i \in S} \|x_i\|$. Note that the standard bound tells us that the perceptron algorithm will make at most $\left(\frac{R}{\gamma}\right)^2$ mistakes.

[4] Indeed there is a vast field that comprises of trying to get "data-dependent bounds", i.e. bounds that give better results if you know some nice properties of your data.

Now assume that we are given the extra information that $\max_{x_i \in S} \|(I-P)x_i\| \leq \epsilon < R$ where $P = w^* {w^*}^\mathsf{T}$ and thus $(I-P)$ is the projector onto the orthogonal complement space of w^* [5]. The goal of this problem is to show that when running the perceptron algorithm on S, the number of mistakes is bounded by $\left(\frac{\epsilon}{\gamma}\right)^2 + 1$ (which is arbitrarily better than the standard bound).

Let i_T be the index of the element on which the Tth mistake was made. Let w_T be the weight vector after T mistakes have been made. Note that $w_0 = 0$.

(ii) Show that $\|w_T\|^2 \leq \epsilon^2 T + \sum_{t=1}^{T} \|Px_{i_t}\|^2$.

Hint: Start by showing that $\|w_T\|^2 \leq \sum_{t=1}^{T} \|x_{i_t}\|^2$. Also, it may be helpful for both (ii) and (iii) to review the properties of projection matrices (but make sure you prove any facts you use).

(iii) Show that $\left(w_T \cdot w^*\right)^2 \geq T(T-1)\gamma^2 + \sum_{t=1}^{T} \|Px_{i_t}\|^2$.

Hint: start by showing that $w_T \cdot w^* = \sum_{t=1}^{T} y_{i_t} x_{i_t} \cdot w^*$.

(iv) Use parts (ii) and (iii) to show that $T \leq \left(\frac{\epsilon}{\gamma}\right)^2 + 1$. Notice (for yourself, no writing necessary) that you successfully proved the tighter bound!

3 Constrained optimization

Compute the distance from the hyperplane $g(x) = w \cdot x + w_0 = 0$ to a point x_a by using constraint optimization techniques, that is, by minimizing the squared distance $\|x - x_a\|^2$ subject to the constraint $g(x) = 0$.

4 Decision Trees, Ensembling and Double Descent

In class, you have studied that decision trees can be simple but powerful classifiers that separate the training data by making a sequence of binary decisions along the optimal features to split the data. In their most basic implementation (without early stopping or pruning), decision trees continue this procedure until every data point in the dataset has a leaf node associated with it. In this problem, we will choose a method to characterize the complexity of decision trees and test the limits of single decision trees on a challenging data-set, before moving onto examining a more powerful algorithm and testing its limits, too.

(i) Download the FashionMNIST dataset[6] provided to you as **train.npy**, **trainlabels.npy**, **test.npy** and **testlabels.npy**. These will be the train and test splits of the dataset you should use for all the following parts. Please ensure that you do not use any other split or download method as you will be evaluated against this split. Once downloaded, visualize a few data-points and their associated labels. Briefly comment on why this is a harder dataset to work with than classical MNIST, and why a simple classifier such as nearest neighbors is likely to do poorly.

[5] Geometrically the data resides in a high dimensional oval (ellipsoid) where the longest axis is in the direction of w^* and has radius R, and the other axes have radius ϵ. While this assumption seems quite construed (if we know that the longest axis is in the direction of w^* it seems that we could trivially find w^*), similar situations can be quite common when doing metric learning (which would approximately stretch the dataset in the direction of w^* while compressing in the orthogonal directions, hence resulting in a similarly oval shaped dataset).

[6] FashionMNIST dataset - https://github.com/zalandoresearch/fashion-mnist

There are many ways to measure the number of parameters in classifiers. In this problem, we will be using the number of leaves in the trained decision tree as a measure of how complex it is. Trivially, this is related to other methods of capturing decision tree complexity, such as its maximum depth and its total number of decisions/nodes.

(ii) As a first step, train a series of decision trees on the training split of FashionMNIST, with a varying limit on the maximum number of permitted leaf nodes. Once trained, evaluate the performance of your classifiers on both the train and test splits, plotting the **0-1 loss** of the train/test curves against the maximum permitted number of leaf nodes (log scale horizontal axis). You are **permitted** to use an open source implementation of a decision tree classifier (such as sklearn's DecisionTreeClassifier) as long as you are able to control the maximum number of leaves. What is the minimum loss you can achieve and what do you observe on the plot?

For your analysis in the rest of the question, recall the definitions of **bias** and **variance** as sources of error. Consider your classifier $\hat{f}(x; D)$ where D is the dataset used to train it. High bias classifiers are ones where in expectation over D, the learnt classifier predicts far away from the true classifier. On the other hand, a high variance classifier is one where the estimate of the function itself will be be strongly sensitive to the split D of the training data used. High bias and high variance classifiers are typically associated with models considered too simple (underfit) or too complex (overfit) respectively, and study of the bias-variance tradeoff helps find the correct model class for good train-test generalization.

(iii) Inspect your decision tree classifiers for the maximum number of leaves they actually used. Did they always use the full capacity of leaves you permitted? (**hint:** If your answer is yes, go back to step 2 and try a tree with higher complexity, aka, one with a greater number of maximum leaf nodes than what you have already tried) What is the maximum number of leaves that a trained classifier ends up using?

Clearly there is a limit to the training of the decision tree beyond which the loss starts going up again. As the complexity of the decision tree increases, its variance does too: it achieves low training (empirical) risk but high test (true) risk. A trivial way to mitigate this would be to keep restricting the maximum number of leaves, but this would bring us back to the high bias zone. Instead, we will turn to ensembling: a trick that allows us to reduce variance without resorting to simpler, high bias classifiers. The idea behind ensembling is to train a range of independent weaker models and combine them towards the final goal of making a stronger model: if our problem is classification, for instance, we may have them vote on the final answer.

The canonical first ensembling method associated with decision trees is the **Random Forest** algorithm. The algorithm trains a series of (shallower) decision trees on random subsets of the original set, before having them vote on the final answer. Intuitively, this allows each tree to be smaller and requires a larger number of trees to agree on an answer before providing a result, hence keep model bias low while reducing variance. Additionally, random forest also allows individual decision trees to only access a random subset of the features in the training data.

(iv) Why do you think it is important for individual estimators in the random forest to have access to only a subset of all features towards reducing variance?

(v) With the random forest model, we now have two hyperparameters to control: the number of estimators and the maximum permitted leaves in each estimator, making the **total parameter count** the **product** of the two. In the ensuing sections, you are allowed to use an open source implementation of the random forest classifier (such as sklearn's RandomForestClassifier) as long as you can control the number of estimators used and maximum number of leaves in each decision tree trained.

   (a) First, make a plot measuring the train and test 0-1 loss of a random forest classifier with a fixed number of estimators (default works just fine) but with varying number of maximum allowed tree leaves for individual estimators. You should plot the train and test error on the same axis against a log scale of the total number of parameters on the horizontal axis. In this case, you are making individual classifiers more powerful but keeping the size of the forest the same. What do you observe- does an overfit seem possible?

   (b) Second, make a plot measuring the train and test 0-1 loss of a random forest classifier with a fixed maximum number of leaves but varying number of estimators. You should plot the train and test error on the same axis against a log scale of the total number of parameters on the horizontal axis. Ensure that the maximum number of leaves permitted is small compared to your answer in part (iii) to have shallower trees. In this case, you are making the whole forest larger without allowing any individual tree to fit the data perfectly, aka without any individual tree achieving zero empirical risk. How does your best loss compare to the best loss achieved with a single decision tree? What about for a similar number of total parameters? With a sufficiently large number of estimators chosen, you should still see variance increasing, albeit with an overall lower test loss curve.

(vi) Now we will generate a final plot. Here we will vary both the number of estimators and the number of maximum leaves allowed, albeit in a structured manner. First while allowing only a single estimator (effectively reducing the random forest to a decision tree) increase the maximum leaves permitted until your answer in part (iii), aka the number of leaves needed for the single tree to overfit- we will call this **Phase 1**. Now, keeping the maximum permitted leaves the same, keep doubling the number of estimators allowed. We will call this **Phase 2**. Train a random forest for all these combinations and make a plot of the train and test 0-1 loss versus the **total number of parameters** on a log scale. Note that **Phase 1** and **Phase 2** is clearly separable on the horizontal axis. Please note the following details as you perform the experiment to make it clear why this experiment is different from the ones you have previously performed. What surprising result do you observe from the loss curve at the end of phase 1?

   - In this experiment, we follow a very structured way of increasing the number of parameters in the model: we first increase the maximum number of permitted leaves of a **single** tree until we see no further change in the operation of the algorithm. Only then do we increase the model complexity by growing the size of the forest. In (v), on the contrary, you experimented with growing the size of the forest without having any single strong tree and growing the size of the trees while keeping forest size constant.
   - Observe what point on the horizontal axis (number of parameters) corresponds to the 'surprising' result on the test loss. Is there a relationship between the order of the number of parameters and the number of data points in the training set?

   The phenomenon, called **double descent** (in seeming contrast to the traditional U-shaped curve test set curve observed in classical ML models) is a recent discovery and has been seen as a way to reconcile the ‘classical’ U-shaped test loss curve and the ‘modern regime’ monotonically decreasing test loss curve usually found in neural networks. You can read more about the theoretical reasoning (as well as some more experiments relating parameters and dataset size concretely) behind the phenomenon in this paper: https://arxiv.org/abs/1812.11118.

Important Note: Remember to include all the plots as required by the questions in your report, organized by the questions that require them, with details clearly being given about any hyperparameters you chose to use. Make sure to submit all code in accordance with the coding instructions already provided. Also, list all your dependencies in a file called **requirements.txt** and submit this inside your code zip.