CS480/680: Introduction to Machine Learning

Homework 5

Due: 11:59 pm, April 9, 2021, submit on LEARN and CrowdMark (see Piazza).

Include your name and student number!

Submit your writeup in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!

[Text in square brackets are hints that can be ignored.]

Exercise 1: Quantile and push-forward (10 pts)

In this exercise we compute and simulate the push-forward map T that transforms a source density q into a target density p. Recall that the quantile function of a (univariate) random variable X is defined as the inverse of its cumulative distribution function (cdf) F:

$$\mathsf{F}(x) = \Pr(X \le x), \qquad \mathsf{Q}(u) = \mathsf{F}^{-1}(u), \quad u \in (0,1). \tag{1}$$

We assume F is continuous and strictly increasing so that $\mathsf{Q}^{-1} = \mathsf{F}$. A nice property of the quantile function, relevant to sampling, is that if $U \sim \text{Uniform}(0,1)$, then $\mathsf{Q}(U) \sim \mathsf{F}$.

1. (2 pts) Consider the Gaussian mixture model $\mathsf{p}(x) = \frac{\pi}{\sigma_1}\varphi\left(\frac{x-\mu_1}{\sigma_1}\right) + \frac{1-\pi}{\sigma_2}\varphi\left(\frac{x-\mu_2}{\sigma_2}\right)$, where φ is the *density* of the standard normal distribution (mean 0 and variance 1). Implement the following to create a dataset of $n = 1000$ samples from the GMM p:

   - Sample $U_i \sim \text{Uniform}(0,1)$.
   - If $U_i < \pi$, sample $X_i \sim \mathcal{N}(\mu_1, \sigma_1)$; otherwise sample $X_i \sim \mathcal{N}(\mu_2, \sigma_2)$.

   Plot the histogram of the generated X_i (with $b = 50$ bins) and submit your script as
   `X = GMMsample(gmm, n=1000, b=50), gmm.pi=0.5, gmm.mu=[1,-1], gmm.sigma=[0.5,0.5]`

   [See here or here for how to plot a histogram in `matplotlib` or `pandas` (or `numpy` if you insist).]

2. (2 pts) Compute $U_i = \Phi^{-1}(\mathsf{F}(X_i))$, where F is the *cdf* of the GMM in Ex 1.1 and Φ is the *cdf* of standard normal. Plot the histogram of the generated U_i (with b bins). From your inspection, what distribution should U_i follow (approximately)? Submit your script as `GMMinv(X, gmm, b=50)`.

   [This page may be helpful.]

3. (2 pts) Let $Z \sim \mathcal{N}(0,1)$. We now compute the push-forward map T so that $\mathsf{T}(Z) = X \sim \mathsf{p}$ (the GMM in Ex 1.1). We use the formula:

   $$\mathsf{T}(z) = \mathsf{Q}(\Phi(z)), \tag{2}$$

   where Φ is the *cdf* of the standard normal distribution and $\mathsf{Q} = \mathsf{F}^{-1}$ is the quantile function of X, namely the GMM p in Ex 1.1. Implement the following binary search Algorithm 1 to numerically compute T. Plot the function T with input $z \in [-5,5]$ (increment 0.1). Submit your main script as `BinarySearch(F, u, lb=-100, ub=100, maxiter=100, tol=1e-5)`, where `F` is a function. You may need to write another script to compute and plot T (based on `BinarySearch`).

4. (2 pts) Sample (independently) $Z_i \sim \mathcal{N}(0,1), i = 1, \ldots, n = 1000$ and let $\tilde{X}_i = \mathsf{T}(Z_i)$, computed by your `BinarySearch`. Plot the histogram of the generated $\tilde{X}_i$ (with b bins) and submit your script as `PushForward(Z, gmm)`. Is the histogram similar to the one in Ex 1.1?

5. (2 pts) Now let us compute $\tilde{U}_i = \Phi^{-1}(\mathsf{F}(\tilde{X}_i))$ as in Ex 1.2, with $\tilde{X}_i$'s generated in Ex 1.4. Plot the histogram of the resulting $\tilde{U}_i$ (with b bins). From your inspection what distribution should $\tilde{U}_i$ follow (approximately)? [No need to submit any script, as you can recycle `GMMinv`.]

6. (optional, 0 pt) Parameterize T as a two-layer neural network (with say 50 hidden neurons and `Relu` activation) and use maximum likelihood to estimate T, based on the samples X_i from Ex 1.1. Plot your T with input $z \in [-5,5]$ (increment 0.1) and compare to Ex 1.3.

Algorithm 1: Binary search for solving a monotonic nonlinear equation $\mathsf{F}(x) = u$.

Input: $u \in (0, 1), \texttt{lb} \leq \texttt{ub}, \texttt{maxiter}, \texttt{tol}$
Output: x such that $|\mathsf{F}(x) - u| \leq \texttt{tol}$

```
while F(lb) > u do
    ub ← lb
    lb ← lb/2
while F(ub) < u do
    lb ← ub
    ub ← ub * 2
for i = 1, ..., maxiter do
    x ← (lb+ub)/2
    t ← F(x)
    if t > u then
        ub ← x
    else
        lb ← x
    if |t − u| ≤ tol then
        break
```

Exercise 2: Robustness and Lasso (5 pts)

Consider the linear regression problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w} - \mathbf{y}\|_2, \tag{3}$$

where $X \in \mathbb{R}^{n\times d}$, $\mathbf{y} \in \mathbb{R}^n$, n is the number of training examples and d is the number of features. For simplicity we omitted the bias. Now suppose we perturb each *feature* independently, and we are interested in solving the robust linear regression problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \max_{\forall j, \|\mathbf{z}_j\|_2 \leq \lambda} \|(X + Z)\mathbf{w} - \mathbf{y}\|_2, \tag{4}$$

where the perturbation matrix $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_d] \in \mathbb{R}^{n\times d}$. Prove that robust linear regression is exactly equivalent to (square-root) Lasso (note the absence of the square on the ℓ_2 norm):

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|X\mathbf{w} - \mathbf{y}\|_2 + \lambda\|\mathbf{w}\|_1, \tag{5}$$

where recall that $\|\mathbf{w}\|_1 = \sum_j |w_j|$.

[Hint: Start from (4), apply the Cauchy-Schwarz inequality $\|\mathbf{w}\|_2 = \max_{\|\mathbf{u}\|_2\leq 1} \mathbf{w}^\top\mathbf{u}$ a few times and a few other tricks, in order to convert it into (5).]

Exercise 3: Adversarial examples (5 pts)

In this exercise we experiment with adversarial examples.

1. (1 pts) Train a neural network on MNIST. (The dataset can be downloaded with torchvision API for example.) You can build your own model or use an existing model architecture. Report the architecture and the test accuracy (should be $> 90\%$).

   [You are suggested to save this trained model for later purpose.]

   [You can use any model architecture (e.g. CNN, MLP), but the test accuracy should be at least 90%.]

2. (2 pts) Use the Fast Gradient Sign Method (FGSM) to generate adversarial images for all the test images (1 adversarial example per image) with L_∞ perturbation budget $\epsilon = 0.2$ against your previous model, and report the model accuracy on this adversarially attacked set of test images. [The resulting value

should be low.]

Display some of your adversarial images (e.g. randomly sample 5) with their labels, and what do you observe?

Repeat the above with $\varepsilon = 0.1$ and $\varepsilon = 0.5$. What do you observe?

[Let x denote the original image, and $\tilde{x}$ the perturbed image. L_∞ perturbation with budget ϵ means $\|x - \tilde{x}\|_\infty \leq \epsilon$]

3. (2 pts) Do adversarial training: Initialize a new model with same architecture as in exercise 3.1. During the training process, instead of using the original training data, generate adversarial examples (with an L_∞ perturbation budget ϵ you prefer) with respect to the parameters at the current iteration and train on them instead. You can use either FGSM or a projected gradient descent (PGD) based strategy to generate these adversarial images.

   After you adversarially trained the new model, repeat the instructions in exercise 3.2 with budgets $\epsilon = 0.1$, 0.2, and 0.5 against your new model. Compare the results with those from exercise 3.2, what do you observe by performing adversarial training?

   [For adversarial training, feel free to generate and use more than 1 adversarial example per training image.]

4. (Bonus: Up to 3 points) Come up with a more robust model: that is, one which achieves a non-trivially higher accuracy when evaluated against the attacks from exercise 3.2. Do you believe that this is *truly* more robust? That is, can you come up with a better way of attacking your new method that causes the accuracy to drop lower again?

   [The truth is that you are unlikely to come up with a truly and strongly robust model: in the research community, attacks are "winning" against the defenses. This is because of an asymmetry: for an attack to be good, it only needs to defeat one defense. But for a defense to be good, it must defeat all attacks.]