CS480/680: Introduction to Machine Learning

Homework 3

Due: 11:59 pm, June 30, 2021, submit on CrowdMark .

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: Decision Trees (8 pts)

In this exercise we will implement decision trees for binary classification. Use the provided stub files.

Recall: decision trees are constructed by repeatedly splitting of nodes. We split a node by measuring the loss of splitting with respect to each possible feature and threshold, and split based on the feature and threshold that minimizes this loss. Mathematically:

$$X_L = \{x \ : \ x \in X \wedge x[i] \leq j\},$$

$$X_R = \{x \ : \ x \in X \wedge x[i] > j\},$$

where $x[i]$ is the ith coordinate of point x. The vector of labels y is split into vectors y_L and y_R using the same indices.

The loss of splitting a training dataset into a left and right half is computed as

$$\ell(X, y, i, j) = \frac{|y_L|}{|y|}\ell(y_L) + \frac{|y_R|}{|y|}\ell(y_R)$$

We will consider the following loss functions ℓ, specialized for binary classification.[a] Define $\hat{p}$ for a vector of labels y to be $\frac{|\{y_j=1:y_j \in y\}|}{|y|}$, that is, the fraction of labels which are 1. We have the following three loss functions.

Misclassification error:

$$\min\{\hat{p}, 1-\hat{p}\}$$

Gini coefficient:

$$\hat{p}(1-\hat{p})$$

Entropy:

$$-\hat{p}\log_2(\hat{p}) - (1-\hat{p})\log_2(1-\hat{p})$$

We do not split a node if it is pure (i.e., consists entirely of either 0's or 1's), or if a split would exceed a maximum depth hyperparameter provided to the decision tree (recall that the depth of a single-node tree is 0).

a) (6 pts) Implement and train decision trees on the provided dataset. Create a different plot for each of the three loss functions (misclassification error, Gini index, and entropy). The x-axis of each plot should show the maximum depth of the tree (starting from 0), and the y-axis should indicate the accuracy. Include two trend lines, one for the training accuracy and test accuracy. Observe and comment on how the different loss functions perform, and how train and test accuracy change as a function of the maximum depth.

b) (2 pts) Implement and use Bagging on decision trees. Use the entropy loss, and create an ensemble of 101 decision trees, capping the maximum depth at 3. Repeat 11 times, report the median, minimum, and maximum test classification accuracy. Repeat, but using the random forests method. In particular, at each split, consider only a random $\sqrt{d}$ features when deciding which dimension to split on. Since $\sqrt{d} \approx 3.6$ for the given dataset, choose a random 4 features for each time. Report the classification accuracies as before. Comment on performance between the two, as well as how they perform in comparison to the non-ensemble methods from the previous part.

[a] Note that these are slightly different from what we discussed in class, since we are only focusing on binary classification. Additionally, there was some implicit rescaling which we omit here.

Exercise 2: Adaboost (5 pts)

Recall the update rules of Adaboost:

$$p_i^t = \frac{w_i^t}{\sum_{j=1}^n w_j^t}, \quad i = 1, \ldots, n \tag{1}$$

$$\epsilon_t = \epsilon_t(h_t) = \sum_{i=1}^n p_i^t \cdot [\![h_t(\mathbf{x}_i) \neq y_i]\!] \tag{2}$$

$$\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t} \tag{3}$$

$$w_i^{t+1} = w_i^t \exp(-y_i \beta_t h_t(\mathbf{x}_i)), \quad i = 1, \ldots, n. \tag{4}$$

Here we use i and t to index the training examples and iterations, respectively. $[\![A]\!] = 1$ if the event A holds and 0 otherwise. Here $y_i \in \{\pm 1\}$ and **we also assume** $h_t(\mathbf{x}_i) \in \{\pm 1\}$. In this exercise we offer additional insights about Adaboost.

a) (1 pt) Prove by induction that the updates defined above are equivalent to the following updates where the y's and h's are $\{0, 1\}$ rather than $\{\pm 1\}$:

$$\tilde{\beta}_t = \frac{\epsilon_t}{1-\epsilon_t} \tag{5}$$

$$\tilde{w}_i^{t+1} = \tilde{w}_i^t \tilde{\beta}_t^{1-|\tilde{h}_t(\mathbf{x}_i) - \tilde{y}_i|}, \tag{6}$$

where $\tilde{y}_i = \frac{y_i+1}{2}, \tilde{h}(\mathbf{x}_i) = \frac{h(\mathbf{x}_i)+1}{2} \in \{0, 1\}$.

[Hint: Show that p_i^t remains the same under the two seemingly different updates.]

Ans:

b) (1 pt) Recall in the logistic regression lecture we made the linear assumption on the log odds ratio:

$$\log \frac{p(y=1|\mathbf{X}=\mathbf{x})}{p(y=-1|\mathbf{X}=\mathbf{x})} \approx \mathbf{w}^\top \mathbf{x} + b. \tag{7}$$

Adaboost in effect tries to approximate the log odds ratio using *additive* functions:

$$\log \frac{p(y=1|\mathbf{X}=\mathbf{x})}{p(y=-1|\mathbf{X}=\mathbf{x})} \approx \sum_{t=1}^T \beta_t h_t(\mathbf{x}), \tag{8}$$

where each $h_t(\mathbf{x})$ is a weak classifier. Indeed, fixing $\mathbf{X} = \mathbf{x}$, prove that the minimizer of the following exponential loss

$$\min_{H \in \mathbb{R}} \quad \mathbb{E}[\exp(-yH)|\mathbf{X} = \mathbf{x}] \tag{9}$$

is (proportional to) the log odds ratio. Here the expectation is wrt the conditional distribution $p(y|\mathbf{X} = \mathbf{x})$.

[Any function can be approximated arbitrarily well by additive functions but clearly not by linear functions, thus the power of Adaboost.]

Ans:

c) (1 pt) Suppose h_t is a weak classifier whose error $\epsilon_t > 1/2$, i.e. worse than random guessing! In this case it makes sense to flip h_t to $\bar{h}_t(\mathbf{x}) = -h_t(\mathbf{x})$. Compute the error $\epsilon_t(\bar{h}_t)$ and the resulting $\bar{\beta}_t$. Do we get the same update for w in (4)? Explain.

Ans:

d) (1 pt) Adaboost is a greedy algorithm where we find the weak classifiers sequentially. At iteration t, the classifiers $h_s, s < t$ are already found along with their coefficients β_s. Suppose h_t is given by some oracle, to find the optimal coefficient β_t, we solve an empirical approximation of the exponential loss (9):

$$\min_{\beta \in \mathbb{R}} \quad \frac{1}{n} \sum_{i=1}^{n} \exp(-y_i[H_{t-1}(\mathbf{x}_i) + \beta h_t(\mathbf{x}_i)]), \tag{10}$$

where needless to say, $H_{t-1}(\mathbf{x}) := \sum_{s=1}^{t-1} \beta_s h_s(\mathbf{x})$. While it is possible to solve (10) directly, we gain more insights by defining a distribution over the training examples $(\mathbf{x}_i, y_i), i = 1, \ldots, n$:

$$p_i^t = \frac{\exp(-y_i H_{t-1}(\mathbf{x}_i))}{\sum_{i=1}^{n} \exp(-y_i H_{t-1}(\mathbf{x}_i))}, \tag{11}$$

so that we can rewrite (10) equivalently as:

$$\min_{\beta \in \mathbb{R}} \quad \hat{\mathbb{E}}_t \exp[-y\beta h_t(\mathbf{x})], \quad (\mathbf{x}, y) \sim \mathbf{p}^t. \tag{12}$$

(Here the hat notation is to remind you that this is an empirical expectation specified by $\mathbf{p}^t$ over our training data.) Let ϵ_t be defined as in (2). Prove that the optimal β in (12) is given in (3).

[Hint: We remind again that both y and $h_t(\mathbf{x})$ are $\{\pm 1\}$-valued. Split training examples according to $h_t(\mathbf{x}_i) = y_i$ or not.]

Ans:

e) (1 pt) What is the training error

$$\epsilon_{t+1}(h_t) = \sum_{i=1}^{n} p_i^{t+1} \cdot [\![h_t(\mathbf{x}_i) \neq y_i]\!] \tag{13}$$

of the weak classifier h_t on the next round $t + 1$? Justify your answer. You may assume $0 < \epsilon_t < 1$ so that all quantities are well-defined.

[Hint: This exercise should be simple, given what you have done in d). Split training examples according to $h_t(\mathbf{x}_i) = y_i$ or not.]

Ans:

Exercise 3: CNN Implementation (8 pts)

Note: Please mention your Python version (and maybe the version of all other packages). For this exercise, you may submit a Python notebook.

In this exercise you are going to run some experiments involving CNNs. You need Python, matplotlib, and either PyTorch, TensorFlow, or JAX (for the adventurous). Be sure to install all necessary dependencies. You can find detailed instructions and tutorials for each of these libraries on the respective websites.

For all experiments, running on CPU is sufficient. You do not need to run the code on GPUs. You may find Colab useful for running things more efficiently (for free), but note that heavy usage may result in them taking away your GPU (unless you pay). Before start, we suggest you review what we learned about each layer in CNN, and read the documentation and tutorial for your framework of choice (either PyTorch, TensorFlow, or JAX).

a) Train a VGG11 net on the MNIST dataset. VGG11 was an earlier version of VGG16 and can be found as model A in Table 1 of this paper, whose Section 2.1 also gives you all the details about each layer. The goal is to get the loss as close to 0 loss as possible. Note that our input dimension is different from the VGG paper. You need to resize each image in MNIST from its original size 28×28 to 32×32 [make sure you understand why this is].

For your convenience, we list the details of the VGG11 architecture here. The convolutional layers are denoted as `Conv(number of input channels, number of output channels, kernel size, stride, padding)`; the batch normalization layers are denoted as `BatchNorm(number of channels)`;

the max-pooling layers are denoted as `MaxPool(kernel size, stride)`; the fully-connected layers are denoted as `FC(number of input features, number of output features)`; the drop out layers are denoted as `Dropout(dropout ratio)`:

```
- Conv(001, 064, 3, 1, 1) - BatchNorm(064) - ReLU - MaxPool(2, 2)
- Conv(064, 128, 3, 1, 1) - BatchNorm(128) - ReLU - MaxPool(2, 2)
- Conv(128, 256, 3, 1, 1) - BatchNorm(256) - ReLU
- Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU - MaxPool(2, 2)
- Conv(256, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU
- Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
- FC(0512, 4096) - ReLU - Dropout(0.5)
- FC(4096, 4096) - ReLU - Dropout(0.5)
- FC(4096, 10)
```

You should use the cross-entropy loss at the end.

[This experiment will take up to 1 hour on a CPU, so please be cautious of your time. If this running time is not bearable, you may cut the training set to 1/10, so only have ∼600 images per class instead of the regular ∼6000.]

b) Once you've done the above, the next goal is to inspect the training process. Create the following plots:

(i) (1 pt) test accuracy vs the number of epochs (say $3 \sim 5$)

(ii) (1 pt) training accuracy vs the number of epochs

(iii) (1 pt) test loss vs the number of epochs

(iv) (1 pt) training loss vs the number of epochs

[If running more than 1 epoch is computationally infeasible, simply run 1 epoch and try to record the accuracy/loss after every few minibatches.]

Ans:

c) Then, it is time to inspect the generalization properties of your final model. Flip and blur the test set images using any Python library of your choice, and complete the following:

(i) (1 pt) test accuracy vs type of flip. Try the following two types of flipping: flip each image from left to right, and from top to bottom. Report the test accuracy after each flip. What is the effect? Please explain the effect in one sentence.

As one resource for those working in PyTorch, you can read this doc to learn how to build a complex transformation pipeline. We suggest the following command for performing flipping:

```
torchvision.transforms.RandomHorizontalFlip(p=1)
torchvision.transforms.RandomVerticalFlip(p=1)
```

Ans:

(ii) (1 pt) test accuracy vs Gaussian noise. Try adding standard Gaussian noise to each test image with variance 0.01, 0.1, 1 and report the test accuracies. What is the effect? Please explain the effect in one sentence.

Again for those working in PyTorch, one way of approaching this is to apply a user-defined lambda as a new transform t which adds Gaussian noise with variance say 0.01:

```
t = torchvision.transforms.Lambda(lambda x : x + 0.1*torch.randn_like(x))
```

Ans:

d) (2 pts) Lastly, let us verify the effect of regularization. Retrain your model with data augmentation and test again as in part 3. Report the test accuracies and explain what kind of data augmentation you use in retraining.

Ans: