CSE 512: Homework 4 Due Nov 5

1. **Jensen's inequality. (1 pts)** Jensen's inequality is an application of the convexity law to expectations. Recall that the expectation of a discrete random variable $f(X)$ with pmf $p_X(x)$ is written as

$$\mathbb{E}[f(X)] = \sum_{i=1}^{m} p_i f(x_i)$$

if the number of values X can take is finite, e.g. in $\{x_1, ..., x_m\}$ and $p_i = \mathbf{Pr}(X = x_i)$.

Show that if f is a convex function, then

$$\mathbb{E}[f(X)] \geq f(\mathbb{E}[X]).$$

2. **Entropy, conditional entropy, mutual information, information gain**

I have a very messy sock drawer, containing 10 red socks, 5 blue socks, 4 yellow socks, and 1 black sock.

(a) **(0.5 pts)** Recall the formula for entropy:

$$H(X) = -\sum_{X=x} \mathbf{Pr}(X = x) \log_2(\mathbf{Pr}(X = x)).$$

Define X a random variable which represents the color of a sock, randomly (uniformly) picked. What is the entropy of this sock?

(b) **(0.5 pts)** My mom comes and tells me I must organize my socks better. So, I put all my red socks in the top drawer and the rest in my bottom drawer. Recall the formula for conditional entropy:

$$H(X|Y) = -\sum_{X=x, Y=y} \mathbf{Pr}(X = x, Y = y) \log_2(\mathbf{Pr}(X = x|Y = y)).$$

What is the conditional entropy, where X is the color of a sock randomly picked, and Y is the drawer of which I pick it from? Assume that I pick the top drawer with twice the probability as picking the bottom drawer, but given a drawer, my choice of sock is uniformly distributed.

(c) **(0.5 pts)** The *information gain* (also called *mutual information*) can be defined in terms of the entropy and conditional entropy

$$I(X; Y) = H(X) - H(X|Y).$$

Give the mutual information between X the color of the sock and Y the drawer which it comes from.

3. **Bias-variance tradeoff.** Suppose I want to identify the location of a star, which lives at coordinates defined by $x \in \mathbb{R}$. Every day I go to the telescope, and I receive a new measurement $y_i = x + z_i$, where $z_i \sim \mathcal{N}(0, 1)$ is the noise in each measurement.

After m days, I receive m measurements, $y_1, ..., y_m \in \mathbb{R}$.

(a) Denote x_{MLE} as the maximum likelihood estimate of x.

i. **(0.3 pts)** Compute x_{MLE} in terms of y_i and m.

ii. **(0.4 pts)** Compute the bias and variance of x_{MLE}.

iii. **(0.3 pts)** Describe the behavior of the bias and variance of x_{MLE} as $m \to +\infty$.

(b) A colleague walks in the room and scoffs at my experiment. "I already know where this star is!" the colleague exclaims, and gives me a new set of measurements $\bar{x} \in \mathbb{R}^n$. "You can just cancel your experiment now!" Trouble is, I know the colleague is full of hot air, so while this is valuable information, I'm not willing to take it without any verification. Instead, I estimate x by solving a linear regression problem

$$\underset{x}{\text{minimize}} \quad \frac{1}{m} \sum_{i=1}^{m} (y_i - x)^2 + \rho(x - \bar{x})^2$$

for some $\rho > 0$. Denote x_{MAP} as the solution to this linear regression problem. You should treat $\bar{x}$ as an external constant, which is not random, but may not be equal to x.

i. **(0.3 pts)** Compute x_{MAP} in terms of y_i, m, $\bar{x}$, and ρ.

ii. **(0.4 pts)** Compute the bias and variance of x_{MAP}.

iii. **(0.3 pts)** Describe the behavior of the bias and variance of x_{MAP} as $m \to +\infty$.

(c) **(0.5 pts)** A natural question to ask is, how to choose ρ? In general, we want ρ to be big when our MLE estimate is not very powerful, either because m is very small or the variance of y_i is very big. On the other hand, if our prior $\bar{x}$ is very close to x, large ρ can also help guide our guess. But while we can't in general know $\|x - \bar{x}\|$, we can try to make some statistical arguments over how ρ should depend on m.

Recall from lecture that

$$\text{MSE} = \mathbb{E}[(x - \hat{x})^2] = B^2 + V.$$

Show that by introducing a dummy variable $\beta = \frac{1}{1+\rho}$, that the MSE can be written as a convex function of β. Use this to find the ρ that minimizes the MSE, as a function of m and an error term $\Delta = \bar{x} - x$.

4. **Decision trees (coding). (3 pts)** In real life, you would use one of many highly optimized packages to program decision trees, but for the sake of understanding, here we will build a tiny decision tree on a simplified version of a multiclass classification problem, using our greedy method.

- Download `covtype.zip`, which is a remote sensing classification problem (more details here `https://archive.ics.uci.edu/ml/datasets/covertype`). Inside there are two raw files: `covtype.info` and `covtype.data`. Skim `covtype.info` to understand the basic task.
- The file `covtype_reduced.mat` has all the data already loaded and separated into a train and test set, and is subsampled severely to reduce computational complexity. (Like I said, this is a simple implementation. Go ahead and use scikit-learn for a "full" version.)
- The task is now to construct a decision tree classifier, that uses information gain to pick the best next split. Open the iPython notebook `covtype.ipynb`, and follow the instructions there.
- Fill in the box to return functions that computes entropy and conditional entropy for a sequence of labels. Return the values given by the test problem provided in the notebook. Remember to include special cases for when one is required to take the log of 0. (We assume that $0 \log_2(0) = 0$.)
- Fill in the box to return a function that, at each step, given X, y, and a set of data sample indices to consider, returns the best feature to split and the best split value, as well as the indices split into two sets. (Follow the template in the iPython notebook). Again, return the solution to the test problem given. Don't forget to handle the special case if the split results in an empty set–something special should happen, so the algorithm knows to reject this split.
- The rest of the implementation is up to you. You can continue to use my iPython notebook, or you can implement your own decision tree using 1) your own implementation of trees and nodes, or 2) something you find online. What you *cannot* do is to use an online implementation of a decision tree. (You can steal the tree, but not the decision part.)
- **My first step.** You can run the "first step" box to start to debug your tree. If you are using my implementation, if you have correctly filled in the holes and added the steps, you should get a result that looks similar to this. (The actual implementation of the split function can cause some variation.)

```
printing tree...
    root 0 2.0 leaf nodes  100 number of samples
current train err: 0.47
current test err: 0.5
printing tree...
    root 0 2.0 split 0, val 2930.62
    0 1 2.0 leaf nodes  41 number of samples
    0 2 1.0 leaf nodes  59 number of samples
one step train err: 0.41
one step test err: 0.42
```

- Report your train and test misclassification rate for 25 steps of training. Describe your resulting tree. Report how many nodes there are in your resulting tree, how many are leaves. List, for the leaves, how many samples ended up on each leaf, and report also their "purity", e.g. # values most frequent / total size of set.
- Did your tree overfit? What are some hints that this may have happened?

5. **Adaboost** A popular and computationally cheap boosting method is adaboost, described in Algorithm 1. In particular, it is a greedy coordinate-wise method that minimizes the empirical exponential loss, e.g. given a predictor $h(x) = y$, we find h which minimizes

$$f(h) = \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i h(x_i)).$$

In this problem, we will implement Adaboost and analyze its greedy structure.

Algorithm 1: Discrete Adaboost (source: https://en.wikipedia.org/wiki/AdaBoost)

```
**Data:** Samples: $x_1, ..., x_m$, training labels $y_i \in \{-1, 1\}$, weak learners $\mathcal{H}$.
**Result:** Classifier $H(x) = \sum_{t=1}^{T} \alpha_t h^{(t)}(x)$
Initial weights $w_i^{(0)} = \frac{1}{m}$ for $i = 1, ..., m$;
**for** $t = 1, ..., T$ **do**
    Choose $h^{(t)}(x)$ which minimizes the weighted sum error for misclassified points
        $h^{(t)}(x) = \underset{h\in\mathcal{H}}{\text{argmin}} \sum_{\substack{i=1 \\ h(x_i)\neq y_i}}^{m} w_i^{(t-1)}$
    Update
        $\alpha^{(t)} = \frac{1}{2}\log\left(\frac{1-\epsilon^{(t)}}{\epsilon^{(t)}}\right), \qquad \epsilon^{(t)} := \sum_{h^{(t)}(x_i)\neq y_i} w_i^{(t-1)} \qquad (\star)$
    Update weights
        $\hat{w}_i^{(t)} = w_i^{(t-1)} \exp(-y_i \alpha^{(t)} h^{(t)}(x_i)), \qquad w_i^{(t+1)} = \frac{\hat{w}_i^{(t+1)}}{\sum_j \hat{w}_j^{(t+1)}}$
**end**
```

(a) **Greedy behavior.** We will now show that the update for α indeed minimizes the empirical risk over the already-trained classifiers, using the exponential loss function. That is, at some time t, show that

$$\alpha^{(t)} = \underset{\alpha^{(t)}}{\text{argmin}} \sum_{i=1}^{m} \mathcal{L}\left(H^{(t)}(x_i); y_i\right), \quad \mathcal{L}(\hat{y}; y) = \exp(-y\hat{y}), \qquad (\Delta)$$

where $H^{(t)}(x) = \sum_{t'=1}^{t} \alpha^{(t')} h^{(t')}(x)$ the current aggregated predictor. Note that y, $\hat{y}$, and $h^{(t)}(x)$ all take binary values in $\{-1, 1\}$. We will do this in several steps.

i. **(0.3 pts)** Define

$$v_i^{(t)} = \exp\left(-y_i \left(\sum_{t'=1}^{t} \alpha^{(t')} h^{(t')}(x_i)\right)\right).$$

Show that the sequence of $w_i^{(t)}$ and $H^{(t)}(x)$ as generated in the algorithm can be written succinctly as

$$w_i^{(t+1)} = \frac{v_i^{(t)}}{\sum_j v_j^{(t)}}.$$

ii. **(0.4 pts)** Noting that $v_i \geq 0$ for all i, define

$$g(\alpha) := \sum_{i=1}^{m} v_i \exp(-y_i \alpha^{(t)} h^{(t)}(x_i)).$$

Show that $g(\alpha)$ is convex, and is minimized when $\alpha = \alpha^{(t)}$ as defined in ($\star$). Hint: observe that $h^{(t)}(x_i) \neq y_i \iff y_i h^{(t)}(x_i) = -1$.

iii. **(0.3 pts)** Use these two parts to argue that the $\alpha^{(t)}$ picked at each iteration is the one that greedily minimizes the loss at each iteration, e.g. satisfies (Δ). That is, show that $g(\alpha)$ is exactly the function where

$$g(\alpha^{(t)}) = \sum_{i=1}^{m} \mathcal{L}(H^{(t)}(x_i); y_i).$$

(b) **Coding. (1 pts)** Open the mnist_adaboost_release directory and in the iPython notebook, download the data. We are again going to do 4/9 handwriting disambiguation. Only minimal preprocessing was used in this dataset. You may now use sklearn's tree implementation.

- Write a function that computes the train and misclassification rates , as well as the train exponential loss value, using just this decision stump (tree of depth = 1). Over the given dataset, these values should be:

$$10.98\% \text{ train misclassification}, 15.27\% \text{ test misclasification}, 0.6259675480492947 \text{ exponential loss}$$

- **Deep trees.** Using the tree library from sklearn, train trees with depth 1 to 100, and plot the train and test misclassification rate, as well as the train exponential loss, as a function of depth. Report also the smallest train and test misclassification rate, and smallest train exponential loss, over the sweep.
- **Boosted decision stumps.** Now build a decision stump, e.g. a tree with depth = 1. Initialize weights as $w_i = 1/m$ for all $i = 1, ..., m$, and fit the decision tree over the *weighted* misclassification error, using the code snippet

```
clf = clf.fit(Xtrain, ytrain, sample_weight = w).
```

Implement the Adaboost method, as shown in algorithm 1. Plot the training exponential loss, and train and test misclassification rate. Report also the smallest train and test misclassification rate, and smallest train exponential loss, over the sweep. Comment on how the boosted decision stumps performed compared to the deep decision tree.
- Plot also $\epsilon^{(t)}$ and $\alpha^{(t)}$ as a function of t. For what values of $\epsilon^{(t)}$ is $\alpha^{(t)}$ really large and positive? really large and negative? close to 0? Interpret this mechanism; what is it saying about how boosting uses classifiers, in terms of their weighted performance?

Challenge!

1. **Medians.** We will show that the solution to the scalar optimization problem

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^{m} |z_i - x|$$

is in fact achieved when x^* is the median of z_i. We can do this using *subdifferentials*, which are a generalization of gradients. In particular, for a function that is differentiable almost everywhere, you can find the subdifferential by taking the convex hull of all the derivatives approaching that point; e.g.

$$\partial f(x) = \text{conv}\left(\left\{\lim_{\epsilon\to 0} \nabla f(x+\epsilon z) : \text{ for all z } \right\}\right).$$

For reference, the *convex hull* of $\mathcal{S}$ is defined as the set that contains all convex combinations of elements in $\mathcal{S}$:

$$\text{conv}(\mathcal{S}) := \{\theta x + (1-\theta) y : x \in \mathcal{S}, y \in \mathcal{S}\}.$$

   (a) Show that the subdifferential of the absolute function $f(x) = |x|$ is

$$\partial f(x) = \begin{cases} \{1\}, & \text{if } x > 0 \\ \{-1\}, & \text{if } x < 0 \\ [-1,1], & \text{if } x = 0. \end{cases}$$

   (b) For a minimization of a nonsmooth convex function $g(x)$,

$$x^* = \underset{x}{\text{argmin}}\; g(x) \qquad \Longleftrightarrow \qquad 0 \in \partial g(x^*).$$

   Write down the subdifferential of $f(x) = \sum_{i=1}^{m} |z_i - x|$. Assume that each z_i are distinct (no 2 values are the same).

   (c) Use these pieces to argue that x^* is the median of z_i, under the assumption that each z_i are distinct and m is an odd number.

2. Let's revisit the messy sock problem. We are using this problem to arrive at a well-known result in information theory. Therefore, although you are free to google around, submit here full justifications for each answer for full credit. (You cannot use the result to justify the result.)

   (a) Suppose that my mom gives me some money to buy 10 socks, and I decide only to buy red and black socks. (Stonybrook colors!) Define the random variable X as the color of the sock I randomly pull out of my shopping bag. How many socks of each color should I buy to maximize the entropy of X?

   As a reminder, the entropy of a random variable which can take 2 values is

$$H(X) = -\mathbf{Pr}(X = \text{red}) \log_2(\mathbf{Pr}(X = \text{red})) - \mathbf{Pr}(X = \text{black}) \log_2 \mathbf{Pr}(X = \text{black}).$$

   (b) Show more generally that, for any $0 \le c \le 1$, the constrained optimization problem

$$\begin{array}{ll} \underset{p}{\text{maximize}} & f(p) := -p\log_2(p) - (c-p)\log_2(c-p) \\ \text{subject to} & 0 \le p \le c \end{array}$$

   reaches its optimum value uniquely at $p = c/2$.

   (c) Now for my birthday my mom takes me to Mall of America, and I have access to socks of 100 different colors. I have enough money to buy 10,000 socks. Again, define X as the random variable taking the value of the color of sock I randomly pull out of my final purchase of 10,000 socks. What color socks should I buy in order to maximize the entropy of X? Use the above pieces to rigorously prove your answer.