CS 475 Machine Learning: Homework 3
Non-linear Methods

Due: Monday April 13, 2018, 11:59pm

110 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Programming (60 points)

1.1 Boosting (30 Points)

You will implement the AdaBoost algorithm for binary classification. AdaBoost takes a weak learner and boosts it into a strong learner. The weak learner you will be using will be a decision stump: a linear classifier **in one dimension of the data**. This is essentially a decision tree that can only take a single feature (hence a stump). For the weak learner you can choose any cutoff in one dimension (more details below).

The hypothesis set H is the set of all one dimensional linear classifiers:

$$H = \{h_{j,c} : j \text{ is the feature index and } c \text{ is the cutoff}\}$$

where

$$h_{j,c}(\mathbf{x}_i) = \begin{cases} \arg\max\limits_{\hat{y}} \sum\limits_{\forall k:\mathbf{x}_{kj}>c} [y_k = \hat{y}] & \text{if } \mathbf{x}_{ij} > c \\ \arg\max\limits_{\hat{y}} \sum\limits_{\forall k:\mathbf{x}_{kj}\leq c} [y_k = \hat{y}] & \text{otherwise} \end{cases}$$

h_t will be used to describe the optimal $h_{j,c}$ at iteration t.

The AdaBoost algorithm is as follows:

1. Input: $(y_1, \mathbf{x}_1), ..., (y_n, \mathbf{x}_n)$ where $\mathbf{x}_i \in \mathbb{R}^m$ and $y_i \in \{-1, 1\}$
2. Initialize: $D_1(i) = \frac{1}{n}$
   where $D_t(i)$ is the weight of instance $(y_i, \mathbf{x}_i)$ at iteration t
3. For each iteration $t \in \{1, 2, ..., T\}$ do:
   (a) $h_t = \arg\min_{h' \in H} \epsilon_t(h')$
   where $\epsilon_t(h) \equiv \sum_{i=1}^n D_t(i)[h(\mathbf{x}_i) \neq y_i]$
   (b) $\alpha_t = \frac{1}{2}\log\frac{1-\epsilon_t(h_t)}{\epsilon_t(h_t)}$
   (c) $D_{t+1}(i) = \frac{1}{Z_{t+1}} D_t(i)\exp(-\alpha_t y_i h_t(\mathbf{x}_i))$
   where $Z_{t+1} = \sum_{i=1}^n D_t(i)\exp(-\alpha_t y_i h_t(\mathbf{x}_i))$ is the normalizing factor

Note for the log used in calculating α_t, the natural log should be used, which is `math.log()` in Python (do `import math` before using the log function). This algorithm will be selected with the value `adaboost` for the argument `algorithm` on the command line.

1.1.1 Choosing $h_{j,c}$

Remember that when you are finding the best $h \in H$, you only need to check up to $n-1$ values of c per dimension. This is because more values will have no affect on your training set. You are only choosing values of c that partition the examples into non-empty sets.

To this end, you should choose values of c in line with something like the *max-margin principle*, where the observable error does not distinguish values of c. That is, assuming you are choosing c for $h_{j,c}$ (i.e. in dimension j) with examples sorted in ascending order of value in j: choose from values of c that will partition the data into $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$ and $\{\mathbf{x}_{k+1}, ..., \mathbf{x}_n\}$ in dimension j as $c \in \{\frac{1}{2}(\mathbf{x}_{k+1,j} + \mathbf{x}_{k,j}) : k \in \{1, 2, ..., n-1\}\}$. Note that the number of distinct values of c may be less than $n-1$ in cases where there are duplicate values in dimension j for some instances. You will choose one dimension j and one of these (up to) $n-1$ values for c that minimizes the error of $h_{j,c}$:

$$h_t = \arg\min_{h_{j,c}} \epsilon_t(h_{j,c}) = \arg\min_{h_{j,c}} \sum_{i=1}^{n} D_t(i)[h_{j,c}(\mathbf{x}_i) \neq y_i]$$

Observe that you can cache these choices after the first time computing them and use the cached values in each iteration of boosting.

1.1.2 Making Predictions

In AdaBoost, you make your predictions as a weighted vote of the classifiers learned on each iteration:

$$\hat{y} = f(x) = \arg\max_{\hat{y}'} \sum_{t=1}^{T} \alpha_t[h_t(x) = \hat{y}]$$

Keep in mind that each h_t is the $h_{j,c}$ at iteration t.

1.1.3 Boosting Iterations

The number of boosting iterations to run will be given as a command line flag. Add this command line option by adding the following code block to the `get_args` function in `classify.py`.

```
parser.add_argument("--num-boosting-iterations", type=int, help="The number of boosting iterations to run.",
default=10)
```

The default number of iterations is 10.

1.1.4 Stopping Criteria

In some cases, your error ϵ_t will reach 0. In this case, α_t will become infinite. Therefore, we will place a stopping criteria on boosting. When you encounter an ϵ_t that is close to 0, where close to 0 is measured as less than 0.000001, stop boosting and do not use the current hypothesis (since we cannot set α_t for this hypothesis). You should stop even if you have not performed every boosting iteration requested by `--num-boosting-iterations`.

1.1.5 Deliverables

You need to implement AdaBoost. Your predictor will be selected by passing the string `adaboost` as the argument for the `algorithm` parameter.

1.1.6 How Your Code Will Be Called

You may use the following commands to test your algorithm.

```
python classify.py --mode train --algorithm adaboost --model-file speech.adaboost.model \
--data speech.train --num-boosting-iterations 10
```

To run the trained model on development data:

```
python classify.py --mode test --model-file speech.adaboost.model --data speech.dev \
--predictions-file speech.dev.predictions
```

1.2 Data Sets

Because the complexity of AdaBoost depends heavily on the number of features in the data, we will not test your algorithms on the NLP data set.

1.3 Multi-layer Perceptron on MNIST (30 Points)

You will implement a simple multi-layer perceptron in pytorch. Because of computational constraints, please restrict the number of hidden layers to 3 or less and the number of neurons in each layer to 500 or less. We suggest you use ReLU activations and a batch size somewhere around 30 to 200.

For pytorch installation, please refer to `http://pytorch.org/`.

For basics and help, please refer to the pytorch tutorial recitation slides (a similar coding example for perceptron is given) on piazza as well as `http://pytorch.org/tutorials/`.

The purpose of this question is to serve as a warm-up for future asignments that will involve pytorch.

1.3.1 Importing and iterating over data

The MNIST dataset can be easily imported using the function `torchvision.datasets.MNIST` (see documentation: `http://pytorch.org/docs/master/torchvision/datasets.html#mnist`). For iterating over the data during training, you will need to input the dataset into a dataloader using `torch.utils.data.DataLoader` (see documentation: `http://pytorch.org/docs/master/data.html`).

Note that when drawing samples from the data loader you will need to convert the drawn samples into a torch Variable in order to make use of training and inference.

1.3.2 Deliverables

Code for this question must be saved in a file named `MLP_MNIST.py`. To assess correctness of implementation we will be directly instantiating a model from your model class. Thus, submit a second code file `MLP_MNIST_test.py` that is identical to your `MLP_MNIST.py` except that it does **not** include training of your model. (If your model starts training then you will run into a timeout error on gradescope.)

Please save your trained model as `model.pkl` in the root directory of your submission. Your trained model will need to achieve a test accuracy of 90% or greater for full credit.

Use the following code for saving:

```
torch.save(my_net.state_dict(), 'model.pkl')
```

where `my_net` is your trained model instantiation.

Make sure your model initialization is done in the following manner:

```
my_net = Net()
```

i.e. Your model class must be named `Net` and it must not require any inputted parameters for initialization. Additionally, inference must be called in the following manner:

```
output = my_net.forward(x)
```

i.e. The method for inference must be named `forward` and take only data input `x` as a parameter. If you deviate from these specifications your code will likely run into errors on gradescope.

2 Analytical (50 points)

1) Deep Neural Networks (12 points)

(a) Consider a 2-layer neural network, with M input nodes, Z nodes in the hidden layer and K nodes in the output layer. The network is fully connected, i.e. every node in the $n-1$th layer is connected to every node in the nth layer. However, for your application of interest, you suspect that only some of the nodes in the input are relevant. How would you modify the objective function to reflect this belief?

(b) Consider a N layer neural network. We could (a) train the entire network at once using back-propagation or (b) pre-train each layer individually, and then tune the final network with back-propagation. Will (a) and (b) converge to the same solution? Why would we favor strategy (a) vs. strategy (b)?

(c) Consider a $N \geq 2$ layer neural network with a single node in the output layer. We wish to train this network for binary classification. Rather than use a cross entropy objective, we want to take a max-margin approach and ensure a margin of $\gamma = 1$. Describe the structure of the last layer of the network, including the final activation function, and the training objective function that implements a max-margin neural network. What are the benefits of this network compared to one trained with cross entropy? Will a max-margin trained neural network learn the same decision boundary as an SVM?

2) Adaboost (12 points) There is one good example at $x = 0$ and two negative examples at $x = \pm 1$. There are three weak classifiers are

$$\begin{aligned} h_1(x) &= 1 \cdot \mathbf{1}(x > 1/2) - 1 \cdot \mathbf{1}(x \leq 1/2), \\ h_2(x) &= 1 \cdot \mathbf{1}(x > -1/2) - 1 \cdot \mathbf{1}(x \leq -1/2) \\ h_3(x) &= 1. \end{aligned}$$

Show that this data can be classified correctly by a strong classifier which uses only three weak classifiers. Calculate the first two iterations of AdaBoost for this problem. Are they sufficient to classify the data correctly?

3) Ensemble Methods (12 points) Consider the following binary classification Boosting algorithm.

1. Given $\{\mathbf{x}_i, y_i\}_{i=1}^N$, number of iterations T, weak learner f.
2. Initialize $\mathcal{D}_0$ to be a uniform distribution over examples.
3. For each iteration $t = 1 \dots T$:
   (a) Train a weak learner f on the data given $\mathcal{D}_t$ to produce hypothesis h_t.
   (b) Compute the error of h_t as $\epsilon_t = P_{\mathcal{D}_t}[h_t(\mathbf{x}_i) \neq y_i]$
   (c) Compute $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
   (d) Either update $\mathcal{D}$ as:
   $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t + (T-t)/T) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t + (T-t)/T) & \text{otherwise} \end{cases}$
   or update $\mathcal{D}$ as:
   $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t - (T-t)/T) & \text{if } h_t(\mathbf{x}_i) = y_i \\ \exp(\alpha_t + (T-t)/T) & \text{otherwise} \end{cases}$
4. Output final hypothesis $H(\mathbf{x}) = \text{sign}\left\{\sum_{t=1}^T \alpha_t h_t(\mathbf{x})\right\}$

Z_t is a normalization constant so that $\mathcal{D}$ is a valid probability distribution.

Describe the difference between this algorithm and the AdaBoost algorithm we learned about in class. Consider the two proposed updates in (d). What is the difference in the behavior of these two updates? Do either of them offer any advantages compared to the Adaboost algorithm learned in class? How does changing the algorithm's user provided parameter affect the behavior?

4. Overfitting in Clustering (14 points) Given the data set $x_1, ..., x_n$, we want cluster the data using the K-means algorithm. The K-means algorithm aims to partition the n observations into k sets $(k < n)$ $S = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares

$$\underset{S=\{S_1,\dots,S_k\}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{x_i \in S_j} \|x_j - \mu_j\|_2^2 \tag{1}$$

where μ_j is the mean of points in S_j.

(a) Let γ_k denote the optimal value of the objective function, prove γ_k is non-increasing in k.

(b) Suppose we modified the objective function as follows:

$$\underset{S=\{S_1,\dots,S_k\}}{\operatorname{argmin}} \sum_{j=1}^{k} \sum_{x_i \in S_j} \max(\|x_j - \mu_j\|_2^2, \tau) \tag{2}$$

where τ is some (given) constant and γ'_k is the optimal value of this new objective function. Compare the values of γ_k and γ'_k $(<, \leq, =, \geq, >)$ and prove this relation.

(c) K-medoids is an algorithm similar to K-means. Both K-means and K-medoids attempt to minimize the squared error but unlike K-means, K-medoids chooses a provided example as a cluster center (medoids) rather than the mean of a subset of the examples. For a given data set $\mathbf{X}$, compare the optimal clusterings produced by K-means and K-medoids $(<, \leq, =, \geq, >)$ and prove this relation.

(d) Suppose you wanted to select k (the number of clusters) to minimize the objective function. Should you work with objective 1 or 2? If 2, how does your choice of τ effect your choice of k?

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `code.zip`. **You must submit source code (.py files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you. We will include the libraries specific in `requirements.txt` but nothing else.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (code.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: https://piazza.com/class/it1vketjjo71l1.