CS 273P: Machine Learning and Data Mining

Homework 4

Due date: **Monday, February 27, 2023**

Instructor: Xiaohui Xie

You have to submit 1) `a single PDF file` that contains everything to Gradescope, and associated each page of the PDF to each problem, 2) `a single zip file` containing source codes to Canvas.

Points: This homework adds up to a total of **125 points**, as follows:

Problem 1: Setting up the data	10 points
Problem 2: Decision Trees	20 points
Problem 3: MNIST	90 points
Statement of Collaboration	5 points

Be sure to re-download and replace the `mltools` package; it contains a number of new classifiers for you.

1 Setting up the data (10 points)

The following is the snippet of code to load the datasets, and split it into train and validation data:

```
# Data Loading
X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)
```

1. Print the minimum, maximum, mean, and the variance of all of the features. `5 points`
2. Split the dataset, and rescale each into training and validation, as:

```
Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
Xt, Yt = Xtr[:5000], Ytr[:5000]   # subsample for efficiency (you can go higher)
XtS, params  = ml.rescale(Xt)    # Normalize the features
XvS, _  = ml.rescale(Xva, params) # Normalize the features
```

   Print the min, maximum, mean, and the variance of the rescaled features. `5 points`

2 Decision Trees (20 points)

For this problem, you will be analyzing the hyper-parameters for the decision tree implementation. There are three hyper-parameters in this implementation that become relevant to its performance; `maxDepth`, `minParent`, and `minLeaf`, where the latter two specify the minimum number of data points necessary to split a node and form a node, respectively.

```
learner = ml.dtree.treeClassify(Xt, Yt, maxDepth=15)
```

1. Keeping `minParent=2` and `minLeaf=1`, vary `maxDepth` to a range of your choosing, and plot the training and validation AUC. `5 points`

   Hint: use learner.auc().

2. Plot the number of nodes in the tree as `maxDepth` is varied (using `learner.sz`). Plot another line in this plot by increasing either `minParent` or `minLeaf` (choose either, and by how much). `5 points`

3. Set `maxDepth` to a fixed value, and plot the training and validation performance of the other two hyper-parameters in an appropriate range, using the same 2D plot we used for nearest-neighbors. Show the plots, and recommend a choice for `minParent` and `minLeaf` based on these results. `10 points`

```
minParents = range(2,10,1) # Or something else
minLeaves = range(1,10,1) # Or something else
tr_auc = np.zeros((len(minParents),len(minLeaves)))
va_auc = np.zeros((len(minParents),len(minLeaves)))
for i,p in enumerate(minParents):
  for j,l in enumerate(minLeaves):
    tr_auc[i][j] = ... # train learner using k and a
    va_auc[i][j] = ...
# Now plot it
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(mat, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+minParents)
ax.set_yticklabels(['']+minLeaves)
plt.show()
```

3 MNIST Classification (90 points)

In this problem, we are going to solve a classification task via two deep learning tools, a) multilayer perceptron, b) a deep (though not very deep) convolutional neural network. We will use the PyTorch framework to build, train and test the networks. Particularly, we will target the MNIST classification problem. The MNIST is a dataset containing small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

3.1 Implement the helper functions.

Before we train our model, we need to implement/complete some helper functions. Fill in the corresponding blanks in the provided notebook.

1. Visualization of the MNIST dataset. Visualize the first image and its label from the training dataset. It should be an image of handwritten '5'. `5 points`

2. Create dataloaders for training and test sets. `5 points`

3. Implement the function for training one epoch, `train()`. `10 points`

4. Implement the function for validating the model, `validate()`. `5 points`

5. Implement the function for calculating the prediction accuracy, `get_accuracy()`. `5 points`

Once we have these helper functions ready, we can define our models and later train these models.

3.2 Implement our models.

3.2.1 LeNet

We will use the famous LeNet as our deep CNN model. The structure of LeNet is shown in Fig. 1, and its detailed architecture is given in Fig. 2. Please complete class definition `class LeNet5`. `20 points`

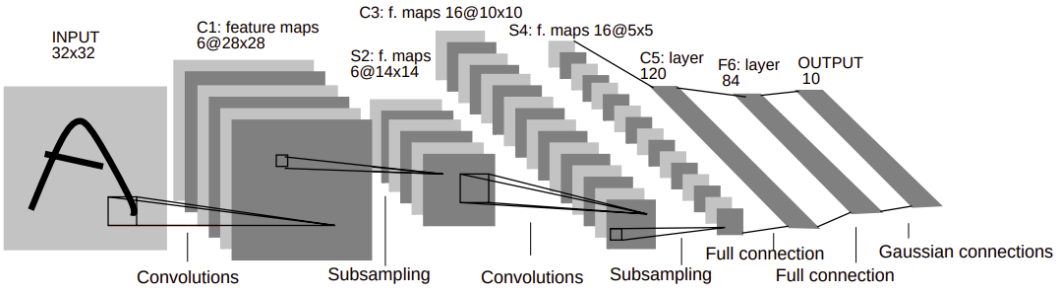

Figure 1: Architecture of LeNet.

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

Figure 2: Detailed Architecture of LeNet.

3.2.2 MLP

Complete the codes inside `class MLP`. The construction function takes an argument `layers`. `layers[0]` is the input dimension, `layers[-1]` is the number of classes, while `layers[1:-1]` are the numbers of nodes of hidden layers. Each hidden layer should be followed by a non-linear activation function. In this homework, we use tanh. `10 points`

3.3 Train and evaluate our models.

1. Train the LeNet, by calling `training_loop()`. Plot and report the performance. `5 points`
2. Train the MLP with hidder layers of [256, 64, 16]. Plot and report its performance. `5 points`

3.4 Comparison of these two models.

1. What is the number of trainable parameters of LeNet? `5 points`
2. What is the number of trainable parameters of MLP? `5 points`
3. Which model has better performance in terms of prediction accuracy on the test data? Give a reason why this model works better than the other. `10 points`

Statement of Collaboration (5 points)

It is **mandatory** to include a `Statement of Collaboration` in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using ed) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content `before` they start working on it.

However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to ed, etc.). Especially `after` you have started working on the assignment, try to restrict the discussion to ed as much as possible, so that there is no doubt as to the extent of your collaboration.