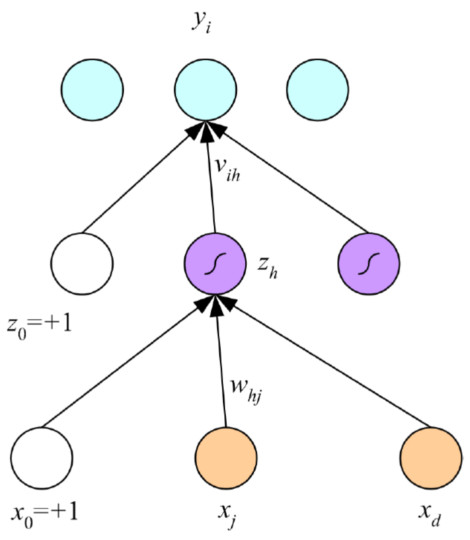
CSCI 5521: Machine Learning Fundamentals (Spring 2021)

Homework 4

(Due Thu, Apr. 22, 11:59 PM central)

1. **(15 points)** Let $\mathcal{X} = \{(\mathbf{x}^1, \mathbf{r}^1), \ldots, (\mathbf{x}^N, \mathbf{r}^N)\}, \mathbf{x}^t \in \mathbb{R}^d, \mathbf{r}^t \in \mathbb{R}^k$ be a set of N training samples. We consider training a multilayer perceptron (below) where the activation functions at each



layer are denoted by $g()$, i.e.,

$$z_h^t = g(a_h^t) = g\left(\sum_{j=1}^{d} w_{h,j} x_j^t + w_0\right) \qquad \text{and} \qquad y_i^t = g(a_i^t) = g\left(\sum_{h=1}^{H} v_{i,h} z_h^t + v_{i0}\right) ,$$

where a_h^t and a_i^t denote the input activation for hidden node h and output node i respectively. Further, let $L(\cdot, \cdot)$ be the loss function (for example the cross entropy loss) which takes in the label r_i^t and predicted label y_i^t for the tth sample, and W and V be the weight matrices, so that the learning focuses on minimizing:

$$E(W, V|\mathcal{X}) = \sum_{t=1}^{N} \sum_{i=1}^{k} L(r_i^t, y_i^t) .$$

(a) **(5 points)** Show that the stochastic gradient descent update for $v_{i,h}$ is of the form $v_{i,h}^{\text{new}} = v_{i,h}^{\text{old}} + \Delta v_{i,h}$ where

$$\Delta v_{i,h} = \eta \Delta_i^t z_h^t , \quad \text{where } \Delta_i^t = g'(a_i^t)\left(-\frac{\partial L(r_i^t, y_i^t)}{\partial y_i^t}\right) .$$

You must show all steps of the derivation for full credit.

(b) **(10 points)** Show that the stochastic gradient descent update for $w_{h,j}$ is of the form $w_{h,j}^{\text{new}} = w_{h,j}^{\text{old}} + \Delta w_{h,j}$ where

$$\Delta w_{h,j} = \eta \Delta_h^t x_j^t \ , \quad \text{where } \Delta_h^t = g'(a_h^t) \left(\sum_{i=1}^{k} \Delta_i^t v_{i,h} \right) \ .$$

You must show all steps of the derivation for full credit.

Programming assignment:

2. **(50 points)** In this problem, we consider binary classification using the `Boston75` dataset. We denote the given dataset as $\{(\mathbf{x}^t, r^t)\}_{t=1}^N$, where $\mathbf{x}^t \in \mathbb{R}^d$ are the feature vectors and r^t are the labels. Your task is to implement the Logistic Regression and Support Vector Machines algorithms, from scratch, using only the provided code and functions specified in the problem description and additional instructions below. You must use gradient descent to train your algorithms. Convergence of gradient descent can be determined by checking the error difference between subsequent iterates $(\mathbf{w}^{t-1}, w_0^{t-1})$ and $(\mathbf{w}^t, w_0^t)$ and making sure the change is below a pre-specified (10^{-6}) threshold. You can also specify `max_iter`, the maximum number of iterations gradient descent can run, and set it to a suitably large value.

   (a) **(25 points)** Consider a 2-class logistic regression with one set of parameters $(\mathbf{w}, w_0)$ where $\mathbf{w} \in \mathbb{R}^d$, $w_0 \in \mathbb{R}$. Assuming the two classes are denoted by labels $r^t \in \{0, 1\}$, the posterior probability of class C_1 is given by

   $$P(C_1|\mathbf{x}) = \frac{\exp(\mathbf{w}^\top \mathbf{x} + w_0)}{1 + \exp(\mathbf{w}^\top \mathbf{x} + w_0)} \ ,$$

   and $P(C_0|\mathbf{x}) = 1 - P(1|\mathbf{x})$. Write code for class MyLogisticRegression in file MyLogisticRegression.py with two key functions `MyLogisticRegression.fit(self, X, r)` and `MyLogisticRegression.predict(self, X)` to implement the logistic regression algorithm.

   For the class, the `__init__(self, d, max_iter,` η`)` function takes as input `d` which is the feature dimensionality and will initialize the parameters **w**, `max_iter` which is the maximum number of iterations gradient descent can run, and η which is the learning rate (step size) used in gradient descent. Note, in the provided code, a column of 1s has been appended to the feature matrix X so that w_0 can be absorbed into the vector **w** and we can compute $\sum_j w_j x_j + w_0$ as $\mathbf{w}^\top \mathbf{x}$.

   For `fit(self, X, r)`, the inputs (X, r) are the feature matrix and class labels respectively. The fit function must use gradient descent to estimate the parameters **w**. Note, you may need to clip the sigmoid() values to be in $[\epsilon, 1 - \epsilon]$ for $\epsilon > 0$ to avoid numerical problems (try $\epsilon = 1e - 10$).

   For `predict(self, X)`, the input X is the feature matrix corresponding to the validation set and the output should a list of predictions equal to the number of data points in the validation set.

(b) **(25 points)** Consider a 2-class SVM with parameters $(\mathbf{w}, w_0)$ where $\mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}$. Assuming the two classes are denoted by labels $r^t \in \{-1, 1\}$, we train an SVM by minimizing the following objective function:

$$\frac{1}{N}\sum_{t=1}^{N} \max\{0, 1 - r^t(\mathbf{w}^\top \mathbf{x}^t + w_0)\} + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

where $\lambda \in \mathbb{R}$ is the regularization parameter which specifies the trade-off between fitting to the data and complexity of the model. Write code for class MySVM in file MySVM.py with two key functions `MySVM.fit(self, X, r)` and `MySVM.predict(self, X)` to implement the SVM algorithm.

For the class, the `__init__(self, d, lambda_val, max_iters,` η`)` function takes as input `d` which is the feature dimensionality, `lambda_val` which is the regularization parameter λ, `max_iters` which is the maximum number of iterations gradient descent can run, η which is the learning rate (step size) used in gradient descent, and will initialize the parameters $\mathbf{w}$ to all zeros. Note, in the provided code, a column of 1s has been appended to the feature matrix X so that w_0 can be absorbed into the vector w and we can compute $\sum_j w_j x_j + w_0$ as $\mathbf{w}^\top\mathbf{x}$.

For `fit(self, X, r)`, the inputs (X, r) are the feature matrix and class labels respectively. The fit function must use gradient descent to estimate the parameters $\mathbf{w}$ and w_0.

For `predict(self, X)`, the input X is the feature matrix corresponding to the validation set and the output should a list of predictions equal to the number of data points in the validation set.

We will compare the performance of these two classifiers (note the output from your code and reported in your solution pdf must be in this order):

(i) `MyLogisticRegression` applied to `Boston75`,

(ii) `MySVM` with $\lambda = 0$ applied to `Boston75`,

(iii) `MySVM` with $\lambda = 0.1$ applied to `Boston75`,

(iv) `MySVM` with $\lambda = 1000$ applied to `Boston75`.

Using the code provided in `hw4q2a.py`, `hw4q2b.py`, and `my_cross_val()` with 5-fold cross-validation, report the error rates in each fold as well as the mean and standard deviation of error rates across folds for each of the above 4 settings.

You will have to submit (a) **code** and (b) **summary of results**:

(a) **Code**: You will have to submit code for `MyLogisticRegression` and `MySVM`. We provide template code for the following classes and functions.

```
class MyLogisticRegression:
  def __init__(self, d, max_iters, η):
  def fit(self, X, r):
  def predict(self, X):
```

```
class MySVM:
  def __init__(self, d, lambda_val, max_iters, η):
  def fit(self, X, r):
  def predict(self, X):
```

Your classes **should not** inherit any base class in `sklearn`. Again, the three functions you must implement in each class are `__init__`, `fit`, and `predict`.

The wrapper scripts `hw4q2a.py` and `hw4q2b.py` are used to prepare the dataset, initialize class objects, make calls to `my_cross_val`(`method`,X,r,k) to generate the error rate results (here k is the number of folds), and print the results. **You must use this wrapper scripts or no credit will be given.**

(b) **Summary of results**: To generate the results, run the hw4q2a.py and hw4q2b.py scripts. For each method, report the validation set error rates for each of the $k = 5$ folds, the mean error rate over the k folds, and the standard deviation of the error rates over the k folds. Make a table to present the results for each method (4 tables in total). Each column of the table represents a fold, and add two columns at the end to show the overall mean error rate and standard deviation over the k folds (see below for an example).

Error rates for MyLogisticRegression on `Boston75`						
Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean	SD
#	#	#	#	#	#	#

3. **(35 points)** In this problem, we consider multiclass classification using the `MNIST` dataset where each sample is an image of a hand-written digit and has a corresponding label indicating the value of the digit written $(0, 1, \ldots, 9)$. We will be using PyTorch[1] to implement the multilayer perceptron algorithm.

   To use PyTorch, you must install both PyTorch and Torchvision. Follow the installation instruction at `https://pytorch.org/get-started/locally/` and `https://pypi.org/project/torchvision/`. We recommend using Anaconda but you can also use the following commands to install PyTorch and Tortchvision in your environment: `pip install pytorch` and `pip install torchvision`. The implementation must be for the CPU version only (no GPUs or MPI parallel programming is required for this assignment).

   Write code for class MyMLP in file MyMLP.py with three key functions `MyMLP.fit(self, dataloader, criterion, optimizer)`, `MyMLP.predict(self, dataloader, criterion)`, and `MyMLP.forward(self, x)` to implement the MLP algorithm.

   For the class, the `__init__(self, input_size, hidden_size, output_size, max_epochs, learning_rate)` function takes as input `input_size` which is the number of pixels in an image (set to 28*28), `hidden_size` which is the number of hidden nodes, `output_size` which is the number of output notes (i.e., number of classes in the dataset), `max_epochs` is the maximum number of epochs to run during training (i.e., number of passes through the dataset), and `learning_rate` which is the gradient descent learning rate (i.e., step size η).

[1] `https://pytorch.org/`

For `fit(self, dataloader, criterion, optimizer)`, the input `dataloader` contains both the feature matrix and class labels, `criterion` is the cross entropy loss, and `optimizer` is the stochastic gradient descent (SGD) algorithm used to minimize the error (see below for more details on `criterion` and `optimizer`). The fit function must use SGD to estimate the parameters $\mathbf{w}$ and w_0. You must also train your algorithm for `max_epochs` epochs where an epoch is a single pass through all the training data.

You must use `cross_entropy` loss and SGD as the optimizer using the following:

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate).
```

For `predict(self, dataloader, criterion)`, the input `dataloader` includes the feature matrix and class labels corresponding to the validation set, and `criterion` is as described above. The output should be the corresponding loss and accuracy evaluated on the test set.

For `forward(self, x)`, the input `x` are images from `dataloader`. The forward function implements the forward pass of the MLP and should pass the data through the network and return the output of the network.

The multilayer (fully connected) perceptron must have the following architecture:

- Input: 1-channel input, size 28x28
- Fully connected linear layer 1: input with bias; output - 128 nodes
- ReLU activation function
- Fully connected linear layer 2: input - 128 nodes; output - 10 nodes.

You will likely need to use the following functions (among others):

- `nn.Linear()`,
- `nn.ReLU()`,
- `loss = criterion()`,
- `optimizer.zero_grad()`,
- `loss.backward()`,
- `optimizer.step()`,

see the provided code for additional details.

We will compare the performance of the MLP with multiclass logistic regression and multiclass SVM from scikit-learn (note the output from your code and reported in your solution pdf must be in this order):

(i) `MyMLP` with `max_epochs = 10`,

(ii) `sklearn.linear_model.LogisticRegression` with `penalty='l2'`, `solver='lbfgs'`, `max_iter=500`, and `multi_class='multinomial'`,

(iii) `sklearn.svm.SVC` with `gamma='scale', C=10, max_iter=500`.

Note, Logistic Regression and SVM may take a while to run. We recommend decreasing the `max_iter` parameter while testing but you must report the final results with `max_iter = 500`.

Using the code provided for `hw4q3.py`, report the classification accuracy on the test set for each of the three settings/algorithms above. Note, you need to add your code to call the sklearn algorithms in `hw4q3.py`.

You will have to submit (a) **code** and (b) **summary of results**:

(a) **Code**: You will have to submit code for `MyMLP`. We provide template code for the following classes and functions.

```
class MyMLP:
  __init__(self, input_size, hidden_size, output_size, max_epochs, learning_rate):
  fit(self, dataloader, criterion, optimizer):
  predict(self, dataloader, criterion):
  forward(self, x):
```

Your classes **should not** inherit any base class in `sklearn` (other than LogisticRegression and SVC). Again, the four functions you must implement in the class are `__init__`, `fit`, `predict`, and `forward`.

The wrapper script `hw4q3.py` is used to prepare the dataset, initialize class objects, make calls to each method's `fit()` and `prediction()` methods to generate the test set accuracy results, and print the results. **You must use this wrapper script or no credit will be given.**

(b) **Summary of results**: To generate the results, run the hw4q3.py script. For each method, report the test set classification accuracy Also, include in your report the figure hw4q3.png which is generated after running hw4q3.py which shows the training loss loss as the number of epochs increases.

Additional instructions: Code can only be written in Python 3.6+; no other programming languages will be accepted. One should be able to execute all programs from the Python command prompt or terminal. Please specify instructions on how to run your program in the README file.

Each function must take the inputs in the order specified in the problem and display the textual output via the terminal and plots/figures should be included in the report.

For each part, you can submit additional files/functions as needed. In your code, you can only use machine learning libraries such as those available from scikit-learn as specified in the problem description. You may use libraries for basic matrix computations and plotting such as numpy, pandas, and matplotlib. Put comments in your code so that one can follow the key parts and steps in your code.

Your code must be runnable on a CSE lab machine (e.g., csel-kh1260-01.cselabs.umn.edu). One option is to SSH into a machine. Learn about SSH at these links: `https://cseit.umn.edu/knowledge-help/learn-about-ssh`, `https://cseit.umn.edu/knowledge-help/choose-ssh-tool`, and `https://cseit.umn.edu/knowledge-help/remote-linux-applications-over-ssh`.

Instructions

Follow the rules strictly. If we cannot run your code, you will not get any credit.

- **Things to submit**
  1. hw4.pdf: A document which contains the solution to Problems 1, 2, and 3 including the summary of results for Problems 2 and 3. This document must be in PDF format (no word, photo, etc. is accepted). If you submit a scanned copy of a hand-written document, make sure the copy is clearly readable, otherwise no credit may be given. **Do not include the PDF file within the ZIP file**. Rather, the PDF document should be submitted as a separate document.
  2. Python code for Problem 2 (must include the required `MyLogisticRegression.py` and `MySVM.py` files).
  3. Python code for Problem 3 (must include the required `MyMLP.py` and your edited `hw4q3.py` file).
  4. README.txt: README file that contains your name, student ID, email, instructions on how to run your code, any assumptions you are making, and any other necessary details.
  5. Any other files, except the data, which are necessary for your code (such as package dependencies like a requirements.txt or yml file).

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to list in the README.txt which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online,
- Look up things/post on sites like Quora, StackExchange, etc.