Assignment 2

CMSC 678 — Introduction to Machine Learning

Due Friday March 9th, 2018, 11:59 AM

Item	**Summary**
Assigned	Monday February 12th, 2018
Due	Friday March 9th, 2018
Topic	Multiclass Classification: Perceptrons, Softmax, and Neural Networks
Points	225

In this assignment you will do math and implement, experiment with, and compare multiple types of multiclass classifiers.

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

	Theory			Application: Classification					
Question	1	2	3	4	5	6	7	8	9
Points	20	40	25	15	20	15	40	20	30

This handout may be lengthy, but think of it as both a tutorial and assignment. I provide a lot of explanation and re-summarizing of course concepts.

However, because this assignment handout *is* lengthy, I am first providing a **task list**. This task list captures the essence of the questions; it details, without other explanatory text, the tasks you are to do and what your completed assignment should answer. The task list enumerates what you must do, but it does not necessarily specify *how*—that's where the full questions come in.

Following the task list are the **full questions**. The full questions do *not* require you to answer additional questions, but they do provide specific details, hints, and explanations. You **should still read and reference** the full questions.

What To Turn In Turn in a **PDF** writeup that answers the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) and run your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment on the submission site:

https://www.csee.umbc.edu/courses/graduate/678/spring18/submit.

Be sure to select "`Assignment 2`."

Task List

1. Show that the MAP decision function optimizes 0-1 classification loss.

2. Turn in a written summary of maxent modeling. Your summary should consider feature design, the issue of moment matching, regularization, and joint vs. conditional modeling. There is no minimum or maximum page limit, but a 1/2 page summary is reasonable.

3. Prove that the cross-entropy loss (log-likelihood objective) for maxent models is convex.

4. Explore and prepare the MNIST data `train`ing data into internal training `int-train` and development `int-dev` sets.

5. Implement a multiclass perceptron. Train it on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your report, document the internal development progress, i.e., how different model configurations perform on `int-dev`. You may *not* use an existing perceptron implementation.

6. Train a maxent classifier (model ids 0 or 6) on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your report, document the internal development progress, i.e., how different model configurations perform on `int-dev`. You *may* use an existing maxent implementation.

7. Implement a multiclass neural network classifier with at least one hidden layer. Train it on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your report, document the internal development progress, i.e., how different model configurations perform on `int-dev`. Experiment with at least two types of model configurations.

8. Using the best perceptron, maxent, and neural network configurations found in the previous three questions, train new models on the entire, original `train` set. At the end of this training, you should have three models trained on the entire 60,000 `train`ing set. Evaluate these models on the original 10,000 image `test` set. Include these evaluation results, and a discussion of them, in your report.

9. Explore how robust each of your best models is. Provide a holistic analysis of these experiments in your report, and compare these results to those you obtained previously from questions 4-8. You may use the `test` set for these experiments.

Full Questions

Theory

1. (**20 points**) Given an input x, consider a discrete probabilistic model $p(y \mid x)$, where $y \in \mathcal{Y}$. For example, $p(y)$ could model the probabilty of rolling a 1, 2, 3, 4, 5, or 6 (represented by y) from a weighted six-sided die. (The input x could be continuous or discrete.) Show that under 0-1 loss, the MAP (maximum a posteriori) estimate is the optimal decision function. That is, show that the optimal decision function is

$$h(x) = \arg\max_{y \in \mathcal{Y}} p(y \mid x). \qquad \text{[Eq-1]}$$

2. (**40 points**) In long-form prose, discuss the following aspects of maxent modeling:

   - The effects of symmetric and conjoined features (conjunctions of features) [roughly lessons 1-4].
   - How moment matching manifests in maxent models [roughly lessons 3-6].
   - The intended effects of ℓ_2 vs. ℓ_1 vs. no regularization [roughly lessons 8-10].
   - The difference in maximizing joint log-likelihood vs. maximizing conditional log-likelihood [roughly lessons 11-15].
   - The impact of context-only (vs. outcome-only or joint outcome-and-context) features in conditional modeling [roughly lessons 12-16].

   You may use math where appropriate, but do try to come up with succinct written (English) explanations. There is no minimum or maximum page limit, but a 1/2 page summary is reasonable.

   The lessons refer to the online tutorial at `https://goo.gl/SsXQS2`.[1] You are *highly* encouraged to work through the tutorial, but you do not have to. Each lesson has some thought questions posed in the instruction box; you should think about them if you work through the tutorial, but you do **not** have to turn in your responses to them.

3. (**25 points**) Consider a maximum entropy model $p(y) \propto \exp(\theta^T f(y))$, where y is a member of the finite set $\mathcal{Y}$. As in Question 1, $p(y)$ could model the probabilty of rolling a 1, 2, 3, 4, 5, or 6 (represented by y) from a weighted six-sided die. It is sometimes beneficial to interpret a correct label $y^\star$ (such as the above die rolling a 4) in a *one-hot format* $\vec{y^\star}$. This one-hot vector $\vec{y^\star}$ is a vector the size of $\mathcal{Y}$: each coordinate $\vec{y^\star}[j]$ corresponds to one of the j items of $\mathcal{Y}$. In a one-hot format, all entries are 0 except for the coordinate corresponding to $y^\star$. For our example of trying to model a correct roll of $y^\star = 4$, a reasonable one-hot equivalent is $\vec{y^\star} = (0, 0, 0, 1, 0, 0)$.

   We can use this one-hot format to help define the **cross-entropy loss**:

$$\ell^{\text{xent}} = -\sum_{j \in \mathcal{Y}} \vec{y^\star}[j] \log p(y = j). \qquad \text{[Eq-2]}$$

   Show that, given N correct labels $Y^\star = \{y_1^\star, y_2^\star, \ldots, y_N^\star\}$, the cross-entropy loss function on $Y^\star$, i.e., the empirical risk, is convex. You may use, without proof, the fact that the (finite) sum of convex functions is convex. *Hint*: consider some of the parts of Assignment 1, question 1.[2]

[1] This is a shortened URL for `https://www.csee.umbc.edu/courses/graduate/678/spring18/loglin-tutorial`.

[2] You may be wondering what are the predicted items in cross-entropy loss. The cross-entropy loss measures the correctness of probabilities corresponding to particular classes, against hard, "gold standard" (correct) judgments. Therefore, the predicted items are actually probabilities, from which you can easily get a single discrete prediction. You do not have to answer, but you should think about, when [Eq-2] would be minimized.

Classification Project: Implementation, Experimentation, and Discussion

The next six questions lead you through building, experimenting with, reporting on the performance of, and analyzing a suite of multiclass classifiers: multiclass perceptron, a maxent model, and a neural network classifier (non-linear multilayer perceptron). The **core deliverables** for these questions are:

(a) any implementations, scripts, and serialized model files needed to compile and run your code; and

(b) a written report discussing

- an analysis of your particular development split [from question 4];
- your implementations, including any design decisions or difficulties you experienced [from questions 5-7];
- evidence and analysis of internal development/experimentation on each of the models on the *development* set [from questions 5-7]; and
- results and analysis of a full comparison on the *test* set [from 8-9].

Each step that this assignment leads you through is its own "question." You may answer these questions individually, or all at once in your report.

The data we'll be using is the MNIST digit dataset: in total, there are 70,000 "images" of handwritten digits (each between 0-9). The task is to predict the digit $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ from a 28x28 input grayscale image x. 60,000 of these are allocated for training and 10,000 are allocated for testing. Do **not** use the 10,000 testing portion until question 8.

4. (**15 points**) You have two tasks in this question. First, get acquainted with the data: make sure that you can read it properly and you are sufficiently familiar with the storage format so you can easily use these files in later questions.

   Second, you are to split the 60,000 training set into an internal training set (`int-train`) and an internal development set (`int-dev`). How you split (e.g., randomly, stratified sampling, taking the first M), and the resulting sizes of `int-train` and `int-dev` are up to you. In your **report**, compare the `int-train`, `int-dev`, and full `train` sets. Discuss how you split these.

   How you choose to do this is up to you: one way is to produce a histogram of the number of digits per split. Another way is to compute the entropy of the empirical label distribution for each of the splits. There are other ways as well.

   The dataset is available from multiple locations, in multiple formats. All of the data in these formats is the same, but don't assume the order within each set is the same. Which you use is more a question of what is easiest for you.

for Python As a gzipped row-major Pickle file, on GL at

`/afs/umbc.edu/users/f/e/ferraro/pub/678-s18/mnist-data/mnist_rowmajor.pkl.gz.`

This maps the keys `'images_train', 'labels_train', 'images_test', 'labels_test'` to numpy arrays.

for Matlab or Python As column-major mat files, on GL at
`/afs/umbc.edu/users/f/e/ferraro/pub/678-s18/mnist-data/mnist_colmajor.mat`.
This maps the keys `'images_train'`, `'labels_train'`, `'images_test'`, `'labels_test'` to arrays/vectors.

for any language as four compressed IDX files from the "original" source, at `http://yann.lecun.com/exdb/mnist/`.[3] These are binary files. If you want to use this version, be aware of first the type to read (e.g., 32 bit int vs. unsigned byte), and that ints are in a high endian form!

Each image x is represented as a 784 ($= 28 \times 28$) one-dimensional array; the row-major/column-major distinction refers to how each image x should be interpreted in memory.

Notice that each component of x is a float ($0 \leq x_i \leq 1$). If you want to visualize the images, you can apply the sign function (with $\tau = 0$) and print its output in a 28x28 grid

$$\text{sign}_\tau(x_i) = \begin{cases} 1 & x_i > \tau \\ 0 & x_i \leq \tau. \end{cases} \qquad \text{[Eq-3]}$$

Note that [Eq-3] provides a **binary** representation of the input x.

5. (**20 points**) Implement a multiclass perceptron. Train it on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your **report**, document the internal development progress, i.e., how different model configurations perform on `int-dev`.

   For this question, you may use computation accelerators (e.g., blas, numpy, MATLAB) but you may not use any existing perceptron implementation.

   Here, configuration types include:

   - the bias;
   - if you implement a standard multiclass perceptron, a voted perceptron, or an averaged perceptron;
   - the convergence criteria;
   - the input feature representation (e.g., do you use x directly, or use a binary repres, or a thresholded identity or binary representation, e.g., $\tau > 0$).

   Training a multiclass perceptron follows the training for a binary perceptron as discussed in class, with the following three changes: first, rather than there being a single weight vector $\mathbf{w}$, there is now a weight vector $\mathbf{w}_j$ per class j. Second, a single prediction is obtained from the maximum of a vector of predictions $\vec{y}$, using each of the class-specific weight vectors. That is, $\vec{y}[j] = \mathbf{w}_j^\intercal x$ and the predicted value $y = \arg\max_j \vec{y}[j]$. Third, each a prediction is incorrect, the correct weights $\mathbf{w}_{y^\star}$ are increased by x, and the mispredicted weights $\mathbf{w}_y$ are decreased by x.

6. (**15 points**) Experiment with maxent classifiers. This assignment is plenty difficult already, so you do *not* have to implement your own maxent classifier. You can use any existing maxent implementation; just tell us which one it is.

   A popular implementation is `liblinear`.[4] It is on GL at

   `/afs/umbc.edu/users/f/e/ferraro/pub/678-s18/liblinear-2.20`.

[3] They are also on GL at `/afs/umbc.edu/users/f/e/ferraro/pub/678-s18/mnist-data/*idx*.gz`. However, you will need to reference the format documentation at the above URL.

[4] `https://www.csie.ntu.edu.tw/~cjlin/liblinear`

Within this directory there are compiled binaries in `bin/train` and `bin/predict`; they have been compiled for GL. There are also Python and MATLAB modules of `liblinear`.

Train a maxent classifier (model ids 0 or 6) on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your **report**, document the internal development progress, i.e., how different model configurations perform on `int-dev`.

Here, configuration types include:

- the regularization used;
- the bias;
- the input feature representation.

7. (**40 points**) Implement a multiclass neural network classifier with at least one hidden layer. Train it on `int-train` and evaluate it on `int-dev`. Try to find the best model configuration possible. In your **report**, document the internal development progress, i.e., how different model configurations perform on `int-dev`. Experiment with at least two types of model configurations.

   For this question, you may use computation accelerators (e.g., blas, numpy, MATLAB) and existing gradient descent and learning rate implementations. You may **not** use any existing neural network, gradient computation, or backpropagation implementations. That is, *you* must derive and implement the gradient yourself but you may use an existing implementation to actually perform the gradient descent (including setting the learning rate).

   Configuration types include, but aren't limited to:

   - the number of and size of the hidden layers;
   - activation functions for each layer (e.g., sigmoid, tanh, ReLU);
   - bias for each layer;
   - loss function (e.g., cross-entropy, hinge loss, squared expectation loss). You are not restricted to these loss functions. Using o to represent the output of the last layer, cross-entropy loss, also called log loss, can be defined as in [Eq-2]:

     $$-\vec{y^\star}^\intercal \log(\text{softmax}(o)). \qquad \text{[Eq-4]}$$

     The squared expectation loss can be defined as

     $$\|\vec{y^\star} - \text{softmax}(o)\|_2^2. \qquad \text{[Eq-5]}$$

     Multiclass hinge loss can be defined as

     $$\max\left\{0, 1 + \max_{y \neq y^\star}\left(\vec{o}[y] - \vec{o}[y^\star]\right)\right\}. \qquad \text{[Eq-6]}$$

   - the input feature representation;
   - any regularization of the loss function (either directly, or through a mechanism like dropout).

8. (**20 points**) Using the best perceptron, maxent, and neural network configurations found in the previous three questions, train new models on the entire, original `train` set. At the end of this training, you should have three models trained on the entire 60,000 `train`ing set. Evaluate these models on the original 10,000 image `test` set. Include these evaluation results in your **report** and discuss the results.

9. (**30 points**) Explore how robust each of your best models is. Provide a holistic analysis of these experiments in your **report**, and compare these results to those you obtained previously from questions 4-8. You may use the `test` set for these experiments.

   How can you examine the "robustness" of your models? There are a number of ways, but three common ones include

   - adding random noise to your data (at training time, testing time, or both);
   - using only a percentage of the `train` set;
   - the order of the `train` set.

   In the first way, you could add a little bit of random noise (e.g., $\epsilon \sim \mathrm{Normal}(0, 0.01)$) to each input pixel. You could also add this random noise randomly (so first flip a coin as whether a pixel will be noised, and then if so, add the noise).

   In the second way, you can train on increasing percentages of the training data. This lets you evaluate your models on the full test set when they have only seen 1% (600 images), or 10% (6000 images), or 25% (15,000 images)—or some other percentage—of the training images.

   In the third way, you can change the order of the training examples. This allows you to examine how the models perform when they're shown training examples in random order, in sorted order (all 0s first, then all 1s, then all 2s, etc.), or in some stratified order (e.g., a 0, then a 1, then a 2, etc.).