Assignment 2

CMSC 678 — Introduction to Machine Learning

Due Wednesday March 8th, 11:59 PM

Item	**Summary**
Assigned	Monday February 20
Due	Wednesday March 8th
Topic	Loss Functions and Multiclass Classification (I)
Points	115 (+15 Extra Credit)

In this assignment you will become comfortable with the math of loss-based optimization; 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 Discord discussions).

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

	Theory			App.: Classification		
Question	1	2	3	4	5	6
Points	20	20	35	5	35	15 (EC)

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.

What To Turn In Turn in a **PDF** writeup that answers the questions; turn in all requested code **and models** necessary to replicate your results. Turn in models by serializing the learned models and parameters; your programming language likely natively supports this (e.g., Python's `pickle`, or Pytorch's own format). Be sure to include specific instructions on how to build/compile (if necessary) and run your code. Answer the following questions in long-form. Provide any necessary analyses and discussion of your results.

How To Submit Submit the assignment on the submission site under "Assignment 2."

https://www.csee.umbc.edu/courses/graduate/678/spring23/submit.

Be sure to select "`Assignment 2`."

Theory

1. (**20 points**) Consider the following *joint* distribution $p(x, y)$:

$$x = \begin{array}{c|ccc} & \multicolumn{3}{c}{y =} \\ & -1 & 0 & 1 \\ a & 0.3 & 0.02 & 0.03 \\ b & 0.03 & 0.2 & 0.02 \\ c & 0.02 & 0.03 & 0.2 \\ d & 0.05 & 0.05 & 0.05 \end{array}$$

   (a) Compute the marginal distribution $p(x)$.

   (b) Compute the marginal distribution $p(y)$.

   (c) For $y = 0$, compute $p(x|y = 0)$.

   (d) Let x_1, x_2, x_3 be three different (empirical) samples, where $x_1 = a$, $x_2 = c$, and $x_3 = d$. Let $p(y|x_i)$ be the posterior distribution of y given the particular value of x_i, e.g., for x_2, this would be $p(y|x = c)$. Compute

$$\sum_{i=1}^{N} \mathbb{E}_{y \sim p(y|x_i)}[\log p(y, x_i)].$$

2. (**20 points**) For $1 \leq i \leq N$, let $x_i \in \mathbb{R}^2$ be a two-dimensional input, and $y_i \in \{-1, +1\}$ the binary label for x_i.

   (a) Describe, in both words and with a concrete example, when the perceptron algorithm would be guaranteed to converge (find an optimal solution to the training set). For the concrete example, give $N \geq 5$ data points $\{(x_i, y_i)\}$ for which the perceptron algorithm would converge.

   (b) Describe, in words and with a concrete example, when the perceptron algorithm would not be guaranteed to converge. For the concrete example, give $N \geq 5$ data points $\{(x_i, y_i)\}$ for which the perceptron algorithm would not converge.

   (c) Let the dataset $\mathcal{D}$ be

$i =$	x_i	y_i
1	(-1, 1)	1
2	(-1, -1)	-1
3	(0.5, 0.5)	1
4	(1, -1)	1
5	(0.5, -1)	-1

   Run, by hand, the perceptron algorithm on these five data points. Record your computations for each time step (data point examined, activation score for that data point and current weight vector, the predicted value, and the new weights) in the following table.

$t =$	Weights $w^{(t)}$	Data point $x_{(t\%N)+1}$	Activation score	Predicted value $\hat{y}$	New weights $w^{(t+1)}$
0	$(0, 0, \ldots, 0)$				
1					
2					
3					
4					
5					
6					

For $t = 0$, please fix $w^{(t)}$ to have the correct dimensionality. The strange indexing $x_{(t\%N)+1}$ simply says to iterate through the data in the order provided above. Note the second column of each row should equal the last column of the previous row.

3. (**35 points**) First, let $f(x)$ be a K-dimensional feature vector, i.e., $f(x) \in \mathbb{R}^K$. The particular meaning of x doesn't matter for this question: it could be an image, or a text document, or collection of robotic sensor readings. In class, we discussed one method of turning a (binary) linear model into a probabilistic (binary) classifier: given vector weights $w \in \mathbb{R}^K$, we pass our linear scoring model $w^\intercal f(x)$ through the sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$ and use the following decision function:

$$\begin{cases} \text{output "class 1"} & \text{if } \sigma(w^\intercal f(x)) \geq 0.5 \\ \text{output "class 2"} & \text{if } \sigma(w^\intercal f(x)) < 0.5. \end{cases} \qquad \text{[Eq-1]}$$

This question considers the multi-class extension of that model.

Second, assume we have L classes, with corresponding weights θ_l $(1 \leq l \leq L)$ per class. We can group these weights into a matrix $\theta \in \mathbb{R}^{LxK}$. Notice that each θ_l is a K-dimensional vector.

Consider a probabilistic linear classification model $p(\hat{y}|x)$, computed as

$$p(y = \hat{y}|x) = \frac{\exp(\theta_{\hat{y}}^T f(x))}{\sum_{y'} \exp(\theta_{y'}^T f(x))}, \qquad \text{[Eq-2]}$$

where $\hat{y}$ and y' are potential values for the label.

(a) Argue that a binary instantiation (i.e., $L = 2$) of model [Eq-2] provides the same decision function as [Eq-1]. In arguing this, you do not need to prove it in a strict mathematical sense, but you need to make it clear that you understand it. (Imagine how you would explain it to a classmate.)

We showed in class that optimizing the MAP estimate is the "right" thing to do when we want to minimize the empirical posterior 0-1 classification loss. Remember that log is monotonic: if $f(a) < f(b)$, then $\log f(a) < \log f(b)$. As such, this is also called **maximizing the conditional log-likelihood**:

$$\arg\max_y p(y|x_i) = \overbrace{\arg\max_y}^{\text{maximizing}} \underbrace{\log p(y|x_i)}_{\text{the conditional log-likelihood}} \qquad \text{[Eq-3]}$$

Given the correct label $y_i^\star$, the above equation will be maximized for $y = y_i^\star$, i.e., $\log p(y_i^\star|x)$ will be the maximum value (across possible y arguments). Using a one-hot representation of the label,[1] we can define the **cross-entropy loss**, which is the dot product between $\vec{y^\star}$ and the *vector* of log conditional probability values $\log p(y|x_i)$:

$$\ell^{\text{xent}} = -\sum_{j \in \mathcal{Y}} \vec{y^\star}[j] \log p(y = j|x_i). \qquad \text{[Eq-4]}$$

[1]It is sometimes beneficial to interpret a correct label $y^\star$ 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 $y^\star$. As an example, consider the case of rolling a 1, 2, 3, 4, 5, or 6 (represented by y) from a weighted six-sided die (where this die outcome represents our label), given some input x: to represent a correct roll of $y^\star = 4$, a reasonable one-hot equivalent is $\vec{y^\star} = (0, 0, 0, 1, 0, 0)$.

(b) Show that, for the probabilistic classifier [Eq-2], maximizing the conditional log-likelihood ([Eq-3]) is the same as minimizing cross-entropy loss ([Eq-4]).[2]

(c) Given N data points $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, formulate the empirical risk objective for [Eq-2] using cross-entropy loss. You can think of this as filling out the following template,

$$\operatorname*{opt}_{\clubsuit} \underbrace{\frac{1}{N}\sum_{i=1}^{N} \diamondsuit}_{\mathcal{L}(\clubsuit)}, \qquad \text{[Eq-5]}$$

where you must specify

- opt: the optimization goal (e.g., max, min, or something else)
- $\clubsuit$: the variables or values being optimized
- $\diamondsuit$: the function (and its arguments) being optimized.

(d) Derive the gradient $\nabla_{\clubsuit}\mathcal{L}(\clubsuit)$.

Classification: Implementation, Experimentation, and Discussion

The next three questions lead you through building, experimenting with, reporting on the performance of, and analyzing a multiclass perceptron classifier. The **core deliverables** for these questions are:

1. any implementations, scripts, and serialized model files needed to compile and run your code; and
2. a written report discussing
   - your implementations, including any design decisions or difficulties you experienced [from questions 4-5];
   - evidence and analysis of internal development/experimentation on the perceptron model on the *development* set [from question 5]; and
   - (optionally, for extra credit) results and analysis of a full comparison on the *test* set [from 6].

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 6.

Get the data from `https://www.csee.umbc.edu/courses/graduate/678/spring23/materials/mnist_rowmajor.jsonl.gz`. This is a gzipped file, where each line is a JSON object. Each line has three keys: "image" (a row-major representation of each image), "label" (an int, 0-9), and "split" (train or test). Each image x is represented as a 784 ($= 28 \times 28$) one-dimensional array (row-major refers to how each image x should be interpreted in memory).

[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.

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.[3] 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.

4. (**5 points**) Implement a "most frequent" baseline classifier. This baseline identifies the label that was most common in training (`int-train`) and always produces that label during evaluation (regardless of the input).
   (a) In your **report**, document the implementation (this should be fairly straight-forward).
   (b) In your **report**, report how well this baseline does on `int-dev`.
5. (**35 points**) Implement a multiclass perceptron. Train it on `int-train` and evaluate it 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.
   (a) In your **report**, discuss your implementation and convergence criteria (this should also be fairly brief).
   (b) In your **report**, discuss the concrete, quantifiable ways you validated your implementation was correct. One such way could be to run it on the toy data in Question 2, though there are other ways too.
   (c) In your **report**, experiment with at least three different configurations, where you train each configuration on `int-train` and then evaluate on `int-dev`. Document this internal development progress through quantifiable and readable means, i.e., graphs and/or tables showing how different model configurations perform on `int-dev`. A configuration could include learning biases or not, the convergence criteria, or the input featurization, among others.

   Implementation Hint: Training a multiclass perceptron follows the training for a binary perceptron as discussed in class, with the following four 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, if 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. Fourth, a per-class bias replaces the single bias term in the binary perceptron algorithm.
6. (**Extra Credit: 15 points**) Using the best perceptron configuration found in the previous question, train new a perceptron model on the entire, original `train` set; also "re-train" a most-frequent classifier baseline. At the end of this training, you should have two models trained on the entire 60,000 `train`ing set. Evaluate these models on the original 10,000 image `test` set. Do not experiment with

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

$$\mathrm{sign}_\tau(x_i) = \begin{cases} 1 & x_i > \tau \\ 0 & x_i \le \tau. \end{cases} \qquad \text{[Eq-6]}$$

Note that [Eq-6] provides a **binary** representation of the input x.

You do *not* have to do the following but you may find it helps you get a handle on the data: visualize the representations (raw/original, binary, or with arbitrary τ) by creating a heatmap/tile plot. This way you can "look" at your data and verify that the labels line up to their respective images (and make sense!).

different configuration values here (that's what your `int-dev` set was for!). Include these evaluation results in your **report** and discuss the results: were they surprising? How similar were the test results to the best internal development results?