Assignment 2

CMSC 678 — Introduction to Machine Learning

Due Wednesday February 27th, 2019, 11:59 AM

<table>
<thead>
<tr>
<th>Item</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assigned</td>
<td>Monday February 11th, 2019</td>
</tr>
<tr>
<td>Due</td>
<td>Wednesday February 27th, 2019</td>
</tr>
<tr>
<td>Topic</td>
<td>Loss Functions and Multiclass Classification (I)</td>
</tr>
<tr>
<td>Points</td>
<td>145</td>
</tr>
</tbody>
</table>

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 Piazza discussions).

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

<table>
<thead>
<tr>
<th>Question</th>
<th>Theory</th>
<th>App.: Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>1 15</td>
<td>4 15</td>
</tr>
<tr>
<td></td>
<td>2 35</td>
<td>5 30</td>
</tr>
<tr>
<td></td>
<td>3 20</td>
<td>6 25</td>
</tr>
</tbody>
</table>

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 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., Java’s `Serializable` or Python’s `pickle`). 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:

[https://www.csee.umbc.edu/courses/graduate/678/spring19/submit](https://www.csee.umbc.edu/courses/graduate/678/spring19/submit)

Be sure to select “Assignment 2.”
Task List

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

2. (a) Write out the three feature vectors \( f(x_i, l_1), f(x_i, l_2), f(x_i, l_3) \) for \( x_i = (0.3, -1, 2.5, 0.78) \) and the definition of \( f \) (Eq-4).
   
   (b) Argue that a binary instantiation of model [Eq-3], with feature function defined by [Eq-4] provides the same decision function as [Eq-2].
   
   (c) Show that, for the probabilistic classifier [Eq-3], maximizing the conditional log-likelihood (Eq-5) is the same as minimizing cross-entropy loss (Eq-6).
   
   (d) Given \( N \) data points \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), formulate an ERM objective using cross-entropy loss.
   
   (e) Derive the gradient of this ERM objective with respect to the appropriate variables.

3. (a) Describe, in both words and with a concrete example of at least 5 data points, when the perceptron algorithm would be guaranteed to converge (find an optimal solution to the training set).
   
   (b) Describe, in both words and with a concrete example of at least 5 data points, when the perceptron algorithm would not be guaranteed to converge (find an optimal solution to the training set).
   
   (c) Run the perceptron algorithm by hand on the five provided data points for seven iterations (time steps) total. Record these in the provided tabular format.

4. (a) Explore and prepare the MNIST data training data into internal training int-train and development int-dev sets. Describe the preparation and resulting splits in your report.

5. Implement a “most frequent” baseline classifier. In your report, document the implementation and report how well this baseline does on int-dev.

6. Implement a multiclass perceptron. Train it on int-train and evaluate it on int-dev. Try at least 3 different model configurations. 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. You do not have to turn in these three models.

7. Using the best baseline and perceptron configurations found in the previous two questions, train new models on the entire, original train set. At the end of this training, you should have two models trained on the entire 60,000 training set. Evaluate these models on the original 10,000 image test set. In your report, include and discuss these evaluation results. Turn in both the baseline and perceptron models trained on the train set.

This concludes the “task list.” If this document only has 2 pages (and not 7 pages), please see the fully detailed PDF at [https://www.csee.umbc.edu/courses/graduate/678/spring19/materials/a2.pdf](https://www.csee.umbc.edu/courses/graduate/678/spring19/materials/a2.pdf) While the detailed PDF does not require any additional items to be completed, it specifies and clarifies many of the details.
Full Questions

Theory

1. (15 points) Given an input \( x \), consider a discrete probabilistic model \( p(y \mid x) \), where \( y \in \mathcal{Y} \). You may assume this \( \mathcal{Y} \) has a finite size \( K \). For example, \( p(y \mid x) \) could model the probability of rolling a 1, 2, 3, 4, 5, or 6 (represented by \( y \)) from a weighted six-sided die, given some input \( x \). (The input \( x \) could be continuous or discrete and for this problem is not important.) 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

\[
\begin{align*}
  h(x) &= \arg \max_{y \in \mathcal{Y}} p(y \mid x). \\
\end{align*}
\]

[Eq-1]

Be detailed and justify (using English) each step.

2. (35 points) In class, we discussed how to turn a (binary) linear model into a probabilistic (binary) classifier: given vector weights \( w \in \mathbb{R}^K \), we pass our linear scoring model \( w^\top x \) through the sigmoid function \( \sigma(z) = \frac{1}{1 + \exp(-z)} \) and use the following decision function:

\[
\begin{align*}
  &\begin{cases} 
    \text{output “class 1”} & \text{if } \sigma(w^\top x) \geq 0.5 \\
    \text{output “class 2”} & \text{if } \sigma(w^\top x) < 0.5 
  \end{cases} \\
\end{align*}
\]

[Eq-2]

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

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

\[
p(\hat{y} \mid x) = \frac{\exp(\theta^\top f(x, \hat{y}))}{\sum_{y'} \exp(\theta^\top f(x, y'))},
\]

where \( \hat{y}, y' \) are members of the finite set \( \mathcal{Y} \). As in Question 1, \( p(\hat{y} \mid x) \) could model the probability of rolling a 1, 2, 3, 4, 5, or 6 (represented by \( \hat{y} \)) from a weighted six-sided die, given \( x \). Again, the particular meaning of \( x \) doesn’t matter for this question.

Let \( x \) be a \( K \)-dimensional feature vector, i.e., \( x \in \mathbb{R}^K \). Define \( f(x, y) \) to be a \( |\mathcal{Y}| \times K \) feature vector, where every \( K \) values correspond to a different output value of \( \mathcal{Y} \). Formally, let \( f_j(x, y) \) be a \( K \) dimensional feature vector, where \( f_j(x, y) \) is \( x \) if \( y = j \) and a \( K \)-dimensional 0-vector otherwise:

\[
f_j(x, y) = \begin{cases} 
  x & \text{if } y = j \\
  (0, 0, \ldots, 0) & \text{else if } y \neq j;
\end{cases}
\]

then the full flattened feature vector \( f(x, y) \) can be written as

\[
f(x, y) = \text{flatten} \left( (f_1(x, y), f_2(x, y), \ldots, f_{|\mathcal{Y}|}(x, y)) \right) \in \mathbb{R}^{|\mathcal{Y}| \times K}.
\]

(flatten simply removes any substructure, e.g., \( \text{flatten}(((3, 4), (1, 2))) = (3, 4, 1, 2) \).)

(a) If we are dealing with a ternary classification problem (where \( \mathcal{Y} \) has three possible output labels \( l_1, l_2, l_3 \)) and a particular data point \( x_i = (0.3, -1, 2.5, 0.78) \), write out the (flattened) feature vectors \( f(x_i, y = l_1), f(x_i, y = l_2), \) and \( f(x_i, y = l_3) \).

(b) Argue that a binary instantiation of model [Eq-3], with feature function defined by [Eq-4] provides the same decision function as [Eq-2]. 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 friend or classmate.)
According to Question 1, optimizing the MAP estimate is the “right” thing to do (right in the sense of minimizing 0-1 classification loss). Remember that log is monotonic: if \( f(a) < f(b) \), then \( \log f(a) < \log f(b) \). When applied to independent datapoints \((x_i, y_i)\), maximizing Eq-1 is also called maximizing the conditional log-likelihood:

\[
\arg \max_y p(y|x_i) = \arg \max_y \log p(y|x_i)
\]

Given the correct label \( y^*_i \), the above equation will be maximized for \( y = y^*_i \), i.e., \( \log p(y^*_i|x) \) will be the maximum value (across possible \( y \) arguments).

It is sometimes beneficial to interpret a correct label \( y^* \) (such as the above die rolling a 4) in a one-hot format \( \vec{y}^* \). This one-hot vector \( \vec{y}^* \) is a vector the size of \( Y \): each coordinate \( \vec{y}^*[j] \) corresponds to one of the \( j \) items of \( Y \). In a one-hot format, all entries are 0 except for the coordinate corresponding to \( y^* \). For our example of trying to model a correct roll of \( y^* = 4 \), a reasonable one-hot equivalent is \( \vec{y}^* = (0, 0, 0, 1, 0, 0) \). We can use this one-hot format to define the cross-entropy loss:

\[
\ell_{\text{ext}} = - \sum_{j \in Y} y^*[j] \log p(y = j|x_i).
\]

(c) Show that, for the probabilistic classifier [Eq-3], maximizing the conditional log-likelihood ([Eq-5]) is the same as minimizing cross-entropy loss ([Eq-6]).

(d) Given \( N \) data points \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \), formulate the objective using cross-entropy loss.

You can think of this as filling out the following template,

\[
\text{opt } \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{y_i \sim \square} [\Diamond],
\]

where you must specify

- \text{opt: the optimization goal (e.g., max, min, or something else)}
- \text{\( \bullet \): the variables or values being optimized}
- \text{\( \square \): the distribution underlying the expectation (i.e., governing the random variable \( y_i \))}
- \text{\( \Diamond \): the function (of a random variable) being optimized.}

(e) Find the gradient \( \nabla_{\bullet} \mathcal{L}(\bullet) \).

3. (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 \( \{(x_1, y_1)\} \) 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 \( \{(x_1, y_1)\} \) for which the perceptron algorithm would not converge.

---

1 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.
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.

<table>
<thead>
<tr>
<th>( t )</th>
<th>Weights ( w^{(t)} )</th>
<th>Data point ( x(t%N)+1 )</th>
<th>Activation score</th>
<th>Predicted value ( \hat{y} )</th>
<th>New weights ( w^{(t+1)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>((0, 0, \ldots, 0))</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

### Classification Project: Implementation, Experimentation, and Discussion

The next four 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
   - an analysis of your particular development split [from question 4];
   - your implementations, including any design decisions or difficulties you experienced [from questions 5-6];
   - evidence and analysis of internal development/experimentation on the perceptron model on the development set [from question 6]; and
   - results and analysis of a full comparison on the test set [from 7].

Though 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 7.

4. (15 points) You have two tasks in this question (but we can only grade the second). 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.

(a) In your report, quantitatively compare the int-train, int-dev, and full train sets. Use tables, graphs (e.g., histograms) to compare and summarize the data splits. 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 (2.7) As a gzipped row-major Pickle file, on GL at 
This maps the keys ‘images_train’, ‘labels_train’, ‘images_test’, ‘labels_test’ to numpy arrays. This Pickle file was created with Python 2.7; please see the note on Piazza (titled “Reading the Pickle file for A2”) for how to use this under Python 3.

for Matlab or Python As column-major mat files, on GL at 
/afs/umbc.edu/users/f/e/ferraro/pub/678-s19/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/](http://yann.lecun.com/exdb/mnist/). 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

$$\text{sign}_\tau(x_i) = \begin{cases} 1 & x_i > \tau \\ 0 & x_i \leq \tau. \end{cases}$$

[Eq-8]

Note that [Eq-8] 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 $\tau$) 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!).

5. (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.

6. (30 points) Implement a multiclass perceptron; this could be a standard perceptron, voted perceptron, or averaged perceptron (your choice). Train it on int-train and evaluate it on int-dev. Try to find the best model configuration possible.

(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 3, though there are other ways too.
(c) In your report, experiment with at least 3 different configurations and document the internal development progress through quantifiable and readable means, i.e., graphs and/or tables showing how different model configurations perform on int-dev. Provide your own analysis and summarization of this.

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 biases;
- 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 representation, or a thresholded identity or binary representation, e.g., $\tau > 0$).

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 $w$, there is now a weight vector $w_j$ per class $j$. Second, a single prediction is obtained from the maximum of a vector of predictions $\tilde{y}$, using each of the class-specific weight vectors. That is, $\tilde{y}[j] = w_j^T x$ and the predicted value $y = \arg \max_j \tilde{y}[j]$. Third, each a prediction is incorrect, the correct weights $w_{y^*}$ are increased by $x$, and the mispredicted weights $w_y$ are decreased by $x$. Fourth, a per-class bias replaces the single bias term in the binary perceptron algorithm.

7. (25 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 training set. Evaluate these models on the original 10,000 image test set. Do not experiment with 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?