HOMEWORK 4

William Powell
908 343 3244

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file to Canvas. Late submissions may not be accepted. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. You can choose any programming language (i.e. python, R, or MATLAB). Please check Piazza for updates about the homework.

Code for this assignment is available at https://github.com/wgraysonp/CS760/tree/main/HW_4.

1 Best Prediction Under 0-1 Loss (10 pts)

Suppose the world generates a single observation $x \sim$ multinomial(θ), where the parameter vector $\theta = (\theta_1, \ldots, \theta_k)$ with $\theta_i \geq 0$ and $\sum_{i=1}^{k} \theta_i = 1$. Note $x \in \{1, \ldots, k\}$. You know θ and want to predict x. Call your prediction $\hat{x}$. What is your expected 0-1 loss:

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}]$$

using the following two prediction strategies respectively? Prove your answer.
Strategy 1: $\hat{x} \in \arg\max_x \theta_x$, the outcome with the highest probability.

With this strategy we have $P(\hat{x} = x) = \theta_{\hat{x}}$. So $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = P(\hat{x} \neq x) = 1 - P(\hat{x} = x) = 1 - \theta_{\hat{x}}$.

Strategy 2: You mimic the world by generating a prediction $\hat{x} \sim$ multinomial(θ). (Hint: your randomness and the world's randomness are independent)

Notice that

$$\begin{aligned} P(\hat{x} = x) &= \sum_{i=1}^{k} P(\hat{x} = i | x = i) P(x = i) \\ &= \sum_{i=1}^{k} P(\hat{x} = i) P(x = i) \\ &= \sum_{i=1}^{k} \theta_i^2 \end{aligned}$$

with the second equality due to the independence of $\hat{x}$ and x. Then

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - P(\hat{x} = x) = 1 - \sum_{i=1}^{k} \theta_i^2.$$

2 Best Prediction Under Different Misclassification Losses (6 pts)

Like in the previous question, the world generates a single observation $x \sim$ multinomial(θ). Let $c_{ij} \geq 0$ denote the loss you incur, if $x = i$ but you predict $\hat{x} = j$, for $i, j \in \{1, \ldots, k\}$. $c_{ii} = 0$ for all i. This is a way to generalize different costs on false positives vs false negatives from binary classification to multi-class classification. You want to minimize your expected loss:

$$\mathbb{E}[c_{x\hat{x}}]$$

Derive your optimal prediction $\hat{x}$.

Let $\mu = (\mu_1, \ldots, \mu_k)$ be an unknown probability vector representing the distribution of our prediction $\hat{x}$. Computing the expected loss we have using independence

$$\mathbb{E}[c_{\hat{x}x}] = \sum_{1 \le i,j \le k} c_{ij} P(\hat{x} = i) P(x = j) = \sum_{i=1}^{k} \mu_i \sum_{j=1}^{k} c_{ij} \theta_j.$$

Clearly this is minimized by setting $\mu_i = 1$ if $i \in \arg\min_{m \in [k]} \sum_{j=1}^{k} c_{mj} \theta_j$ and $\mu_i = 0$ otherwise (here $[k] = \{1, \ldots, k\}$). So the optimal prediction $\hat{x}$ is $\hat{x} \in \arg\min_{m \in [k]} \theta^T c_i$ were $c_i = (c_{i1}, \ldots, c_{ik})$.

3 Language Identification with Naive Bayes (8 pts each)

Implement a character-based Naive Bayes classifier that classifies a document as English, Spanish, or Japanese - all written with the 26 lower case characters and space.
The dataset is languageID.tgz, unpack it. This dataset consists of 60 documents in English, Spanish and Japanese. The correct class label is the first character of the filename: $y \in \{e, j, s\}$. (Note: here each file is a document in corresponding language, and it is regarded as one data.)
We will be using a character-based multinomial Naïve Bayes model. You need to view each document as a bag of characters, including space. We have made sure that there are only 27 different types of printable characters (a to z, and space) – there may be additional control characters such as new-line, please ignore those. Your vocabulary will be these 27 character types. (Note: not word types!)

1. Use files 0.txt to 9.txt in each language as the training data. Estimate the prior probabilities $\hat{p}(y = e)$, $\hat{p}(y = j)$, $\hat{p}(y = s)$ using additive smoothing with parameter $\frac{1}{2}$. Give the formula for additive smoothing with parameter $\frac{1}{2}$ in this case. Print and include in final report the prior probabilities. (Hint: Store all probabilities here and below in $\log()$ internally to avoid underflow. This also means you need to do arithmetic in log-space. But answer questions with probability, not log probability.)

   For each class label $y_0 \in \{e, j, s\}$ the formula for $\hat{p}(y = y_0)$ with smoothing parameter $1/2$ is given by

   $$\hat{p}(y = y_0) = \frac{1}{N + 3/2} \left(\sum_{i=1}^{N} \mathbb{1}(y_i = y_0) + 1/2 \right) \tag{1}$$

   where $N = 30$ is the number of training samples. In this example we have $\hat{p}(y = y_0) = 0.34$ for each $y_0 \in \{e, j, s\}$.

2. Using the same training data, estimate the class conditional probability (multinomial parameter) for English

   $$\theta_{i,e} := \hat{p}(c_i \mid y = e)$$

   where c_i is the i-th character. That is, $c_1 = a, \ldots, c_{26} = z, c_{27} = space$. Again use additive smoothing with parameter $\frac{1}{2}$. Give the formula for additive smoothing with parameter $\frac{1}{2}$ in this case. Print θ_e and include in final report which is a vector with 27 elements.

   For each $y_0 \in \{e, j, s\}$ let S_{y_0} be the set of all characters c such that c comes from a document with language y_0. Then for each $1 \le i \le 27$ the conditional probability $\hat{p}(c_i | y = y_0)$ is given by

   $$\hat{p}(c_i | y = y_0) = \frac{\sum_{c \in S_{y_0}} \mathbb{1}(c = c_i) + 1/2}{|S_{y_0}| + 27/2}$$

   where $|S_{y_0}|$ is the cardinality of S_{y_0}. The vector θ_e is printed below.

   $$\theta_e = [0.06, 0.011, 0.022, 0.022, 0.105, 0.019, 0.017, 0.047, 0.055, 0.001, 0.004, 0.029, 0.021, 0.058, 0.064, 0.017, 0.001, 0.054, 0.066, 0.08, 0.027, 0.009, 0.015, 0.001, 0.014, 0.001, 0.179]$$

3. Print θ_j, θ_s and include in final report the class conditional probabilities for Japanese and Spanish.

   The vectors θ_j and θ_s are included below.

$$\theta_s = [0.105, 0.008, 0.038, 0.04, 0.114, 0.009, 0.007, 0.005, 0.05, 0.007, 0.0, 0.053, 0.026, 0.054, 0.073, 0.024, 0.008, 0.059, 0.066, 0.036, 0.034, 0.006, 0.0, 0.002, 0.008, 0.003, 0.168]$$

$$\theta_j = [0.131, 0.011, 0.005, 0.017, 0.06, 0.004, 0.014, 0.032, 0.097, 0.002, 0.057, 0.001, 0.04, 0.056, 0.091, 0.001, 0.0, 0.043, 0.042, 0.057, 0.07, 0.0, 0.02, 0.0, 0.014, 0.008, 0.127]$$

4. Treat e10.txt as a test document x. Represent x as a bag-of-words count vector (Hint: the vocabulary has size 27). Print the bag-of-words vector x and include in final report.

   The bag of words vector x is

$$x = [164, 32, 53, 57, 311, 55, 51, 140, 140, 3, 6, 85, 64, 139, 182, 53, 3, 141, 186, 225, 65, 31, 47, 4, 38, 2, 491].$$

5. Compute $\hat{p}(x \mid y)$ for $y = e, j, s$ under the multinomial model assumption, respectively. Use the formula

$$\hat{p}(x \mid y) = \prod_{i=1}^{d} \theta_{i,y}^{x_i}$$

   where $x = (x_1, \ldots, x_d)$. Show the three values: $\hat{p}(x \mid y = e), \hat{p}(x \mid y = j), \hat{p}(x \mid y = s)$. Hint: you may notice that we omitted the multinomial coefficient. This is ok for classification because it is a constant w.r.t. y.

   Using the formula above the values are $\hat{p}(x|y = e) = \exp(-7829.8), \hat{p}(x|y = j) = \exp(-8752.4)$, and $\hat{p}(x|y = s) = \exp(-8454.9)$.

6. Use Bayes rule and your estimated prior and likelihood, compute the posterior $\hat{p}(y \mid x)$. Show the three values: $\hat{p}(y = e \mid x), \hat{p}(y = j \mid x), \hat{p}(y = s \mid x)$. Show the predicted class label of x.

   The estimated posteriors are $\hat{p}(y = e|x) = \exp(-7830.9), \hat{p}(y = j|x) = \exp(-8753.4)$, and $\hat{p}(y = s|x) = \exp(-8455.9)$. Then predicted class label of x is then $y = e$.

7. Evaluate the performance of your classifier on the test set (files 10.txt to 19.txt in three languages). Present the performance using a confusion matrix. A confusion matrix summarizes the types of errors your classifier makes, as shown in the table below. The columns are the true language a document is in, and the rows are the classified outcome of that document. The cells are the number of test documents in that situation. For example, the cell with row = English and column = Spanish contains the number of test documents that are really Spanish, but misclassified as English by your classifier.

   The confusion matrix for the classifier on the test set is given below. It can be seen that the model achieved 100% accuracy on the test set.

		True Label		
		e	j	s
Predicted Label	e	10	0	0
	j	0	10	0
	s	0	0	10

8. If you take a test document and arbitrarily shuffle the order of its characters so that the words (and spaces) are scrambled beyond human recognition. How does this shuffling affect your Naive Bayes classifier's

prediction on this document? Explain the key mathematical step in the Naive Bayes model that justifies your answer.

This shuffling has no affect on the classifier's prediction of the document. Naive Bayes assumes that the features are independent conditioned on the class label. If x is a vector of document characters with length d then Naive Bayes assumes

$$p(x|y = y_0) = \prod_{i=1}^{d} p(x_i|y = y_0)$$

for any class label y_0. Since re-ordering the vector x does not change the above quantity, the Naive Bayes classifier will predict the same label if the document is shuffled.

4 Simple Feed-Forward Network (20pts)

In this exercise, you will derive, implement back-propagation for a simple neural network and compare your output with some standard library's output. Consider the following 3-layer neural network.

$$\hat{y} = f(x) = g(W_2\sigma(W_1x))$$

Suppose $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, and $W_2 \in \mathbb{R}^{k \times d_1}$ i.e. $f : \mathbb{R}^d \to \mathbb{R}^k$, Let $\sigma(z) = [\sigma(z_1), ..., \sigma(z_n)]$ for any $z \in \mathbb{R}^n$ where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid (logistic) activation function and $g(z_i) = \frac{exp(z_i)}{\sum_{i=1}^{k} exp(z_i)}$ is the softmax function. Suppose the true pair is (x, y) where $y \in \{0, 1\}^k$ with exactly one of the entries equal to 1, and you are working with the cross-entropy loss function given below,

$$L(x, y) = -\sum_{i=1}^{k} y\log(\hat{y})$$

1. Derive backpropagation updates for the above neural network. (5 pts)

   Let W_{ij}^{ℓ} be the entry in the i-th row and j-th column of matrix ℓ for $\ell = 1, 2$. First we compute $\nabla_{W^2} L$. Let $a = \sigma(W_1x)$, $z = W^2a$, and $\hat{y} = g(z)$. By the chain rule

$$\frac{\partial L}{\partial W_{ij}^2}(x, y) = \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \left(\sum_{\beta=1}^{k} \frac{\partial \hat{y}_\alpha}{\partial z_\beta} \frac{\partial z_\beta}{\partial W_{ij}^2} \right) \tag{2}$$

$$= a_j \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \frac{\partial \hat{y}_\alpha}{\partial z_i} \tag{3}$$

$$= -a_j \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \Big(\hat{y}_\alpha(1 - \hat{y}_\alpha)\mathbb{1}(\alpha = i) + \hat{y}_\alpha\hat{y}_i\mathbb{1}(\alpha \neq i) \Big) \tag{4}$$

$$= -a_j \sum_{\alpha=1}^{k} \mathbb{1}(y = \hat{y}_\alpha) \Big((1 - \hat{y}_\alpha)\mathbb{1}(\alpha = i) + \hat{y}_i\mathbb{1}(\alpha \neq i) \Big). \tag{5}$$

   Here, (3) is from $\frac{\partial z_\beta}{\partial W_{ij}^2} = 0$ if $\beta \neq i$ and $\frac{\partial z_i}{\partial W_{ij}^2} = a_j$. For (4) we have $\frac{\partial \hat{y}_\alpha}{\partial z_i} = -\hat{y}_\alpha(1 - \hat{y}_\alpha)$ if $\alpha = i$ and $\frac{\partial \hat{y}_\alpha}{\partial z_i} = -\hat{y}_\alpha\hat{y}_i$ otherwise. Finally, (5) uses $\frac{\partial L}{\partial \hat{y}_\alpha} = \mathbb{1}(y = y_\alpha) \cdot \frac{1}{\hat{y}_\alpha}$

Now computing $\nabla_{W^1} L$ we have

$$\frac{\partial L}{\partial W^1_{ij}} = \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \left(\sum_{\beta=1}^{k} \frac{\partial \hat{y}_\alpha}{\partial z_\beta} \left(\sum_{\gamma=1}^{d_1} \frac{\partial z_\beta}{\partial a_\gamma} \frac{\partial a_\gamma}{\partial W^2_{ij}} \right) \right) \tag{6}$$

$$= a_i(1-a_i)x_j \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \left(\sum_{\beta=1}^{k} \frac{\partial \hat{y}_\alpha}{\partial z_\beta} \frac{\partial z_\beta}{\partial a_i} \right) \tag{7}$$

$$= a_i(1-a_i)x_j \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \left(\sum_{\beta=1}^{k} \frac{\partial \hat{y}_\alpha}{\partial z_\beta} W^2_{\beta i} \right) \tag{8}$$

$$= -a_i(1-a_i)x_j \sum_{\alpha=1}^{k} \frac{\partial L}{\partial \hat{y}_\alpha} \left(\sum_{\beta=1}^{k} \Big(\hat{y}_\alpha(1-\hat{y}_\alpha)\mathbb{1}(\beta=\alpha) + \hat{y}_\alpha \hat{y}_\beta \mathbb{1}(\beta \neq \alpha) \Big) W^2_{\beta i} \right) \tag{9}$$

$$= -a_i(1-a_i)x_j \sum_{\alpha=1}^{k} \mathbb{1}(y=\hat{y}_\alpha) \left(\sum_{\beta=1}^{k} \Big((1-\hat{y}_\alpha)\mathbb{1}(\beta=\alpha) + \hat{y}_\beta \mathbb{1}(\beta \neq \alpha) \Big) W^2_{\beta i} \right) \tag{10}$$

Here (7) uses $\frac{\partial a_\gamma}{\partial W^1_{ij}} = 0$ if $\gamma \neq i$ and $\frac{\partial a_i}{\partial W^1_{ij}} = \frac{\partial}{\partial W^2_{ij}} \sigma(W^1_i x) = \sigma(W^1_i x)(1-\sigma(W^1_i x))x_j$ where W^1_i is the i-th row of W^1. Line (8) uses $\frac{\partial z_\beta}{\partial a_i} = \frac{\partial}{\partial a_i} W^2 a = W^2_{\beta i}$. Lines (9) and (10) use the same differentiation rules as (4) and (5) respectively.

2. Implement it in NumPy or PyTorch using basic linear algebra operations. (e.g. You are not allowed to use auto-grad, built-in optimizer, model, etc. in this step. You can use library functions for data loading, processing, etc.). Evaluate your implementation on MNIST dataset, report test errors and learning curve. (10 pts)

Training and test curves are shown below in Figure 1. Training was run for 50 epochs where one epoch is one full pass over the training data set. Both training error and test error were recored at the end of each epoch. The model was trained using SGD with learning rate $\alpha = 0.1$ and batch size 128 and weights were initialized uniformly at random in $[-1, 1]$. The final test error was 0.05 for final accuracy of 95%.

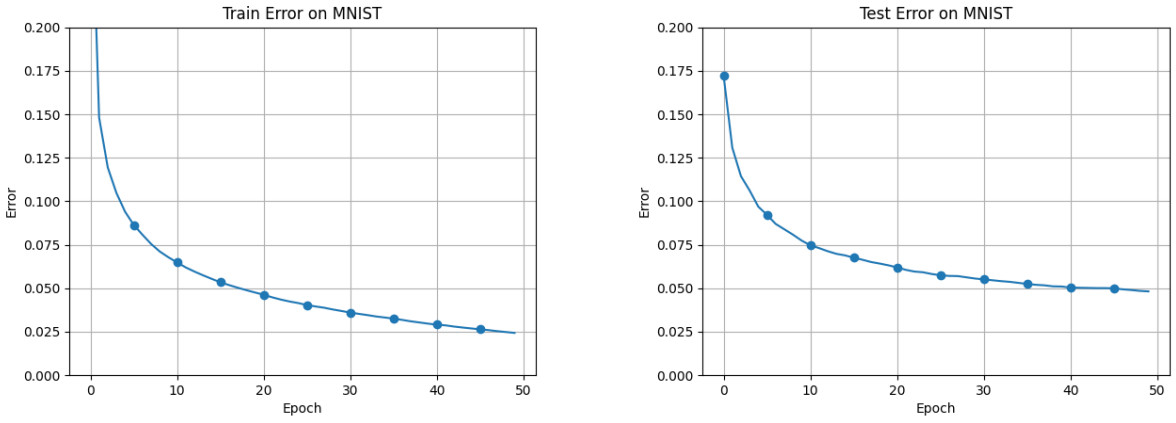

Figure 1: Training and test error on MNIST for 3-layer network implemented from scratch.

3. Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)

Training and test curves for the model implemented with torch are shown below in Figure 2. The same batch size, learning rate, and initialization as in the previous question were used. The results are quite similar.

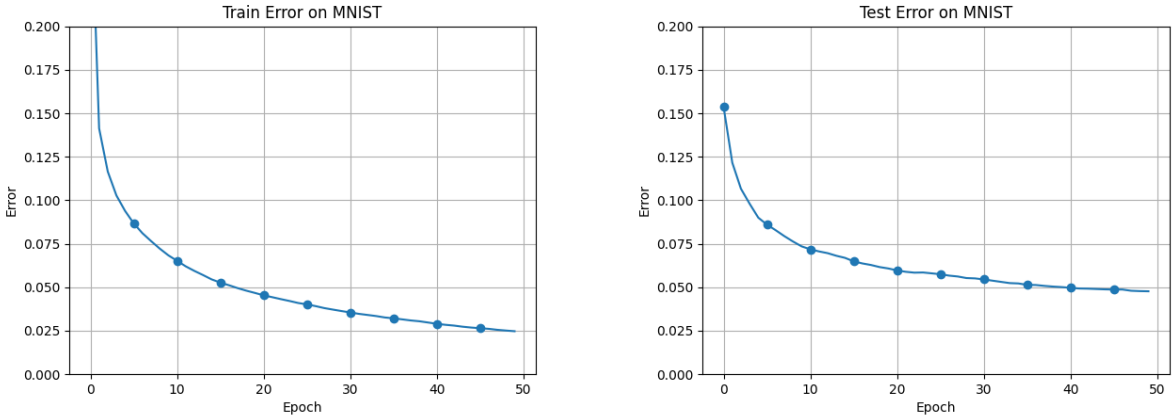

Figure 2: Training and test error on MNIST for 3-layer network implemented using PyTorch.

4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)

   Figure 3 compares the training and test error for the network implemented from scratch using two different initialization strategies: uniform in $[-1, 1]$ and zero. Random initialization performs better, achieving a final test error of 0.05 versus 0.075 for zero initialization. In this experiment all hyperparamter values are the same as in the previous two questions.

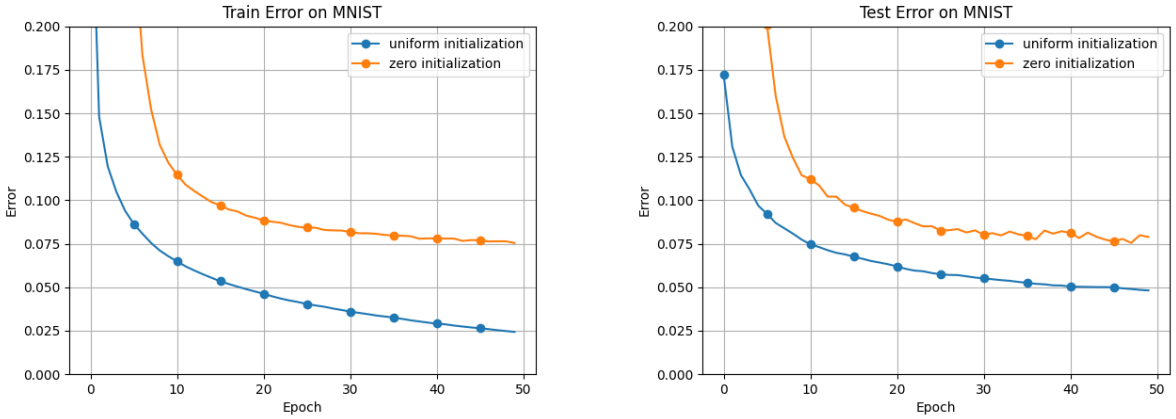

Figure 3: Comparison of weight initialization strategies.

You should play with different hyperparameters like learning rate, batch size, etc. for your own learning. You only need to report results for any particular setting of hyperparameters. You should mention the values of those along with the results. Use $d_1 = 300$, $d_2 = 200$. For optimization use SGD (Stochastic gradient descent) without momentum, with some batch size say 32, 64, etc. MNIST can be obtained from here (https://pytorch.org/vision/stable/datasets.html)