HOMEWORK 4

Hailey Park
hpark353
https://github.com/hparkailey/760ML.git

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.

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.

$$\begin{aligned} E[\mathbb{1}_{\hat{x} \neq x}] &= P(x \neq \hat{x}) \\ &= 1 - P(x = \hat{x}) \\ &= 1 - \frac{\theta_{\hat{x}}}{\sum_{i=1}^{k} \theta_i} \\ &= 1 - \theta_{\hat{x}} \end{aligned}$$

$\theta_{\hat{x}}$ denotes the $argmax_x \theta_x$.
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)

$$\begin{aligned} E[\mathbb{1}_{\hat{x} \neq x}] &= P(x \neq \hat{x}) \\ &= \sum_{i=1}^{k} \sum_{j=1}^{k} P(\hat{x} = 1 \wedge x = j) \text{ where } i \neq j \\ &= \sum_{i=1}^{k} \sum_{j=1}^{k} P(\hat{x} = 1) P(x = j) \text{ where } i \neq j \\ &= \sum_{i=1}^{k} \sum_{j=1}^{k} \theta_i \theta_j \text{ where } i \neq j \end{aligned}$$

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 $\hat{x}$.

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij} P(x = i \wedge \hat{x} = j)$$
$$= \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}\theta_i P(\hat{x} = j)$$

Since we want to minimize this $\mathbb{E}[c_{x\hat{x}}]$, our optimal prediction is $\hat{x} = argmin_j \sum_{i=1}^{k} c_{ij}\theta_i$.

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.)

   The formula for the prior using additive smoothing is as follows:

   $$\hat{P}(y) = \frac{b_y + \frac{1}{2}}{`\sum_{k=1}^{3} b_k + \frac{1}{2} * 3}$$

   b_y denotes the count of language y document across all documents. This yields $P(e) = P(j) = P(s) = 0.3$.

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.

   Refer to Table 1 for the θ_e.

   $$\hat{P}(c_i|y = e) = \frac{b_i + \frac{1}{2}}{`\sum_{k=1}^{27} b_k + \frac{1}{2} * 27}$$

   Here b_i denotes the count of character i across all the English documents.

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

   Refer to table 2 and 3 for θ_j and θ_s.

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 BOW vector for e10.txt is as follows : [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, 498]. This list represents BOW counts for each character in our dictionary from character "a" to " ".

Table 1: θ_e

Chararacter	Conditional Probability
a	0.060169
b	0.011135
c	0.021510
d	0.021973
e	0.105369
f	0.018933
g	0.017479
h	0.047216
i	0.055411
j	0.001421
k	0.003734
l	0.028977
m	0.020519
n	0.057922
o	0.064464
p	0.016752
q	0.000562
r	0.053825
s	0.066182
t	0.080126
u	0.026664
v	0.009285
w	0.015496
x	0.001156
y	0.013844
z	0.000628
	0.179250

Table 2: θ_j

Character	Conditional Probability
a	0.131766
b	0.010867
c	0.005486
d	0.017226
e	0.060205
f	0.003879
g	0.014012
h	0.031762
i	0.097033
j	0.002341
k	0.057409
l	0.001433
m	0.039799
n	0.056711
o	0.091163
p	0.000874
q	0.000105
r	0.042804
s	0.042175
t	0.056990
u	0.070617
v	0.000245
w	0.019742
x	0.000035
y	0.014151
z	0.007722
	0.123449

Table 3: θ_s

Character	Conditional Probability
a	0.104560
b	0.008233
c	0.037526
d	0.039746
e	0.113811
f	0.008603
g	0.007184
h	0.004533
i	0.049860
j	0.006629
k	0.000278
l	0.052943
m	0.025809
n	0.054177
o	0.072492
p	0.024267
q	0.007678
r	0.059295
s	0.065770
t	0.035614
u	0.033702
v	0.005889
w	0.000093
x	0.002498
y	0.007863
z	0.002683
	0.168265

	e	j	s
$\hat{e}$	10	0	0
$\hat{j}$	0	10	0
$\hat{s}$	0	0	10

Table 4: Confusion Matrix

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. Log likelihoods are as follows: 'e': -7841.865447060635, 'j': -8771.433079075032, 's': -8467.282044010557

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 log posterior are as follows: 'e': -7843.069419864961, 'j': -8772.637051879357, 's': -8468.486016814883. Since we predict the most likely class, our predicted class label is "e", English.

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.

   Confusion matrix as shown in table 4.

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. No, changing the order of the characters would not change the classifier prediction, because of the conditional independence assumption of Naive Bayes. This assumption makes the computation of likelihood independent of the order in which the characters are observed.

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.

$$\hat{y} = f(x) = g(W_2\sigma(W_1 x))$$

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)

$$\begin{aligned} \frac{dL}{dW_2} &= \frac{dL}{d\hat{y}} * \frac{d\hat{y}}{dg} * \frac{dg}{dz} * \frac{dz}{dW_2} \\ &= (-y + g(z)) \otimes \sigma(W_1 x) \end{aligned}$$

$$\begin{aligned}\frac{dL}{dW_1} &= \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dg} \cdot \frac{dg}{dz} \cdot \frac{dz}{d\sigma} \cdot \frac{d\sigma}{da} \cdot \frac{da}{dW_1} \\ &= ((-y + g(z)) \cdot W_2) \odot (\sigma(a) \odot (1 - \sigma(a)) \otimes x)\end{aligned}$$

where $z = W_2\sigma(W_1x)$ and $a = W_1x$.

More specifically, we derive backpropagation updates for each term as follows.

$$\begin{aligned}\frac{dL}{d\hat{y}} &= \frac{-y}{\hat{y}} \\ \frac{d\hat{y}}{dg} &= I_{k\times k} \\ \frac{dg_i(z)}{dz_j} &= \frac{d}{dz_j}\left(\frac{e^{z_i}}{\sum_{i=1}^{k} e^{z_i}}\right) \\ &= \frac{d}{dz}(e^{z_i})\left(\sum_{i=1}^{k} e^{z_i}\right)^{-1} + e^{z_i}\frac{d}{dz_j}\left(\left(\sum_{i=1}^{k} e^{z_i}\right)^{-1}\right) \\ &= g(z_j)(I_{i=j} - g(z_i)) \\ \frac{dz}{dW_2} &= \frac{d}{dW_2}(W_2\sigma(W_1x)) \\ &= \sigma(W_1x)\end{aligned}$$

$$\begin{aligned}\frac{dZ}{d\sigma} &= \frac{d}{d\sigma}W_2\sigma(W_1x) \\ &= W_2\end{aligned}$$

$$\frac{d\sigma}{da} = \sigma(a)(1 - \sigma(a))$$

$$\frac{da}{dW_1} = x \tag{1}$$

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)

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)

   With the hidden dimension of $d1 = 300$, the mean cross entropy loss on test sets across 50 iterations of epochs were 1.68. Learning rate was 0.003 and the optimizer momentum was 0.9.The learning curve is shown in figure 1.

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)

   When the weights were initialized to 0 with d1=300, the mean cross entropy loss on test sets with 50 epochs were 2.18. Learning rate was 0.003 and the optimizer momentum was 0.9. The learning curve is shown as in Figure 2.

   When the weights were randomly initialized between -1 and 1 with the same setting for the rest, the mean cross entropy loss on test sets was 1.92. The learning curve is shown in Figure 3.

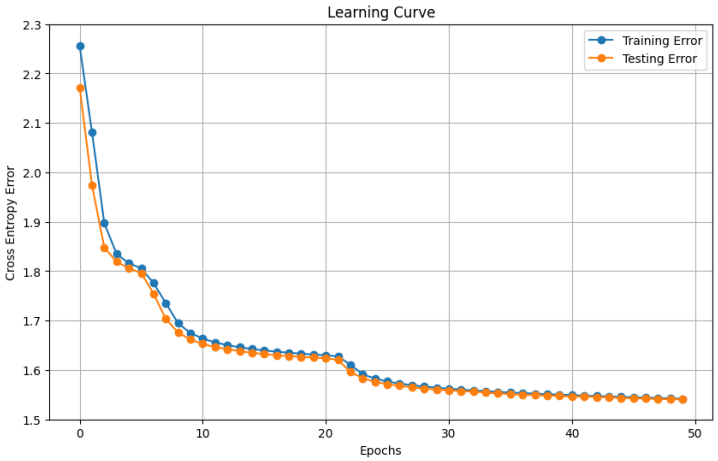

Figure 1: Learning Curve

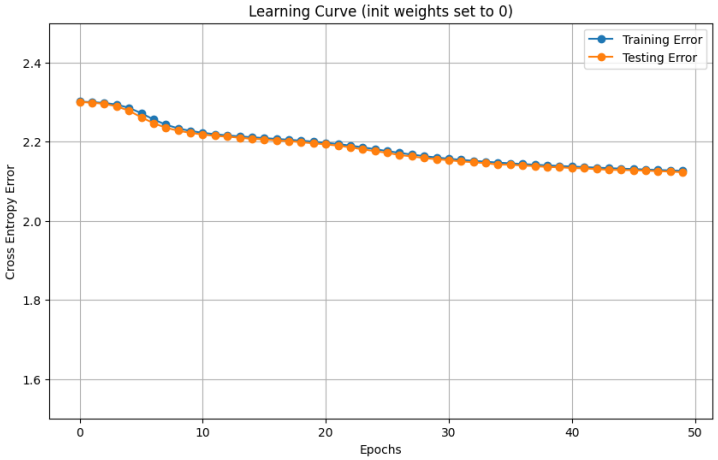

Figure 2: Learning Curve with Weights Set to 0

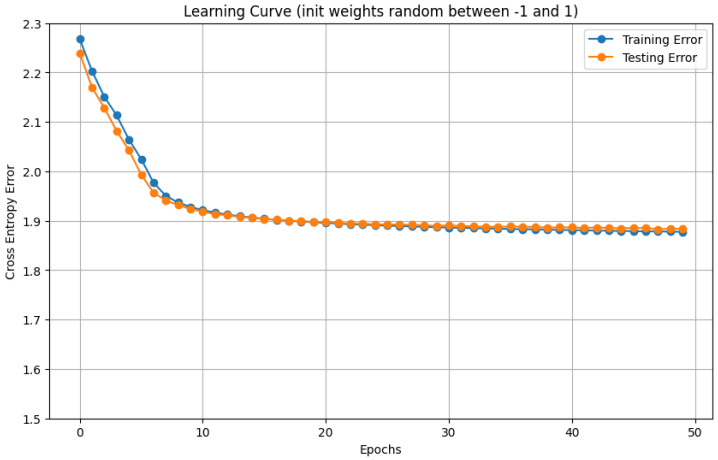

Figure 3: Learning Curve with Random Weights [-1,1]

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)