HOMEWORK 4

Nick Boddy
nboddy

GitHub Repo: https://github.com/Nick-Boddy/CS760-HW4-Bayes-NN

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.
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)
Strategy 1: If we select the x that maximizes θ_x, we end up with the largest value in θ, denoted $\theta_{\hat{x}}$.

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= Pr(\hat{x} \neq x) \\ &= 1 - Pr(\hat{x} = x) \\ &= 1 - \theta_{\hat{x}}\end{aligned}$$

Strategy 2: If we generate a prediction $\hat{x} \sim$ multinomial(θ), then the probability that $\hat{x} = x$ is θ_x.

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= Pr(\hat{x} \neq x) \\ &= 1 - Pr(\hat{x} = x) \\ &= 1 - \theta_x\end{aligned}$$

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 $\hat{x}$.

$\mathbb{E}[c_{x\hat{x}}]$ is the expected loss when predicting x using $\hat{x}$.

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

$$= \sum_{i=1}^{k} Pr(x = i) \sum_{j=1}^{k} Pr(\hat{x} = j | x = i) \cdot c_{ij}$$

$$= \sum_{i=1}^{k} Pr(x = i) \mathbb{E}[c_{i\hat{x}}]$$

$$= \sum_{i=1}^{k} \theta_i \cdot \mathbb{E}[c_{i\hat{x}}]$$

$$\hat{x} = \arg\min_{\hat{x}} \sum_{i=1}^{k} \theta_i \cdot \mathbb{E}[c_{i\hat{x}}]$$

$$= \arg\min_{\hat{x}} \sum_{i=1}^{k} \theta_i \cdot c_{i\hat{x}}$$

$$= \arg\min_{\hat{x}} \theta^\top c_{\hat{x}}$$

$$\text{where } c_{\hat{x}} = \begin{bmatrix} c_{1\hat{x}} \\ c_{2\hat{x}} \\ \vdots \\ c_{k\hat{x}} \end{bmatrix}$$

In other words, the optimal $\hat{x}$ is the one in which $\theta^\top c_{\hat{x}}$ is lowest. This makes sense - the expected loss is the weight of the loss multiplied by the probability of it happening.

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.)

$$\hat{p}(y = e) = \frac{N_{y=e} + 0.5}{N_{total} + 0.5(3)} = \frac{10 + 0.5}{30 + 1.5} = \frac{1}{3}$$
$$\hat{p}(y = j) = \frac{N_{y=j} + 0.5}{N_{total} + 0.5(3)} = \frac{10 + 0.5}{30 + 1.5} = \frac{1}{3}$$
$$\hat{p}(y = s) = \frac{N_{y=s} + 0.5}{N_{total} + 0.5(3)} = \frac{10 + 0.5}{30 + 1.5} = \frac{1}{3}$$

Since we have the same number of samples for each language (10 documents), it makes sense that even with additive smoothing they have the same probabilities.

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.

$$\begin{aligned}\theta_{i,e} &:= \hat{p}(c_i \mid y = e) \\ &= \frac{N_{c_i,e} + 0.5}{N_e + 0.5(27)}\end{aligned}$$

Here, $N_{c_i,e}$ is the total count of character c_i in training documents labeled $y = e$. N_e is total count of all characters for that same label. My computed vector for θ_e is:

$$\theta_e = \begin{bmatrix} 0.0601685114819098 \\ 0.011134974392863043 \\ 0.021509995043779945 \\ 0.021972575582355856 \\ 0.1053692383941847 \\ 0.018932760614571286 \\ 0.017478936064761277 \\ 0.047216256401784236 \\ 0.055410540227986124 \\ 0.001420783082768875 \\ 0.0037336857756484387 \\ 0.028977366595076822 \\ 0.020518751032545846 \\ 0.057921691723112505 \\ 0.06446390219725756 \\ 0.01675202378985627 \\ 0.0005617049396993227 \\ 0.053824549810011564 \\ 0.06618205848339666 \\ 0.08012555757475633 \\ 0.026664463902197257 \\ 0.009284652238559392 \\ 0.015496448042293078 \\ 0.001156451346439782 \\ 0.013844374690236246 \\ 0.0006277878737815959 \\ 0.1792499586981662 \end{bmatrix}$$

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

$$\theta_j = \begin{bmatrix} 0.1317656102589189 \\ 0.010866906600510151 \\ 0.005485866033054963 \\ 0.01722631818022992 \\ 0.06020475907613823 \\ 0.003878542227191726 \\ 0.014011670568503443 \\ 0.03176211607673224 \\ 0.09703343932352633 \\ 0.0023411020650616725 \\ 0.05740941332681086 \\ 0.001432614696530277 \\ 0.03979873510604843 \\ 0.05671057688947902 \\ 0.09116321324993885 \\ 0.0008735455466648031 \\ 0.00010482546559977637 \\ 0.04280373178657535 \\ 0.0421747789929767 \\ 0.056990111464411755 \\ 0.07061742199238269 \\ 0.0002445927530661449 \\ 0.01974212935462455 \\ 3.4941821866592126e-05 \\ 0.01415143785596981 \\ 0.00772214263251686 \\ 0.12344945665466997 \end{bmatrix} \quad \theta_s = \begin{bmatrix} 0.10456045141993771 \\ 0.008232863618143134 \\ 0.03752582405722919 \\ 0.039745922111559924 \\ 0.1138108599796491 \\ 0.00860287996053159 \\ 0.007184839813758445 \\ 0.0045327001942585795 \\ 0.049859702136844375 \\ 0.006629459467793161 \\ 0.0002775122567913416 \\ 0.052943171656748174 \\ 0.02580863988159477 \\ 0.054176559464709693 \\ 0.07249236841293824 \\ 0.02426690512164287 \\ 0.007677839104560451 \\ 0.05929511886774999 \\ 0.06577040485954797 \\ 0.03561407295488884 \\ 0.03370232185254849 \\ 0.00588942678301625 \\ 9.250408559711388e-05 \\ 0.0024976103111220747 \\ 0.007862847275754679 \\ 0.0026826184823163022 \\ 0.16826493170115014 \end{bmatrix}$$

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.

   e10.txt has the following "bag-of-words" (character counts):

   $$[164, 32, 53, 57, 311, 55, 51, 140, 140, 3, 6, 85, 64, 139, 182, 53, 3, 141, 186, 225, 65, 31, 47, 4, 38, 2, 498]$$

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.

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

   $$\hat{p}(x \mid y = e) = e^{-7841.865447060635}$$
   $$\hat{p}(x \mid y = j) = e^{-8771.433079075032}$$
   $$\hat{p}(x \mid y = s) = e^{-8467.282044010557}$$

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.

$$\hat{p}(y \mid x) = \frac{\hat{p}(x \mid y)\hat{p}(y)}{\hat{p}(x)}$$

However, since we're just finding the y that maximizes this posterior, we can drop $\hat{p}(x)$ as such:

$$\hat{p}(y \mid x) = \hat{p}(x \mid y)\hat{p}(y)$$

$$\hat{p}(y = e \mid x) = \hat{p}(x \mid y = e)\hat{p}(y = e) = \frac{e^{-7841.865447060635}}{3}$$
$$\hat{p}(y = j \mid x) = \hat{p}(x \mid y = j)\hat{p}(y = j) = \frac{e^{-8771.433079075032}}{3}$$
$$\hat{p}(y = s \mid x) = \hat{p}(x \mid y = s)\hat{p}(y = s) = \frac{e^{-8467.282044010557}}{3}$$

$$\hat{y} = \arg\max_y \hat{p}(y \mid x) = e$$

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.

Confusion Matrix:

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

As we can observe from the confusion matrix above, our Naive Bayes classifier has 100% accuracy on our test set. We might infer from this that letter frequency can be a decent predictor of language (at least in regards to distinguishing these three languages).

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.

Such a shuffling of a test document would not affect its prediction at all. Our Naive Bayes classifier considers only the counts (i.e. frequency) of each character and doesn't care at all about ordering. As described before, we consider the document to be a "bag of words", in that position is meaningless. We see this mathematically in part 5, where we express the probability of x given y to be the product of the conditional probabilities of every individual character in x. Our formulae don't take at all into consideration locality of the characters.

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)

   Each weight will update according to: $w = w - \alpha \nabla L(w)$ (for a simple SGD).
   Starting with our loss function $L(x, y) = -\sum_{i=1}^{k} y_i \log(\hat{y_i})$, we want to compute the gradient with respect to W_1 and W_2. We'll define the following:

$$z = W_2 a$$

$$a = \sigma(W_1 x)$$

   Calculating the gradient with respect to z_i:

$$\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i}\left[-\sum_{n=1}^{k} y_n \log(\hat{y_n})\right]$$

$$= \frac{\partial}{\partial z_i}\left[-\sum_{n=1}^{k} y_n \log(g(z_n))\right]$$

$$= -\sum_{n=1}^{k} \frac{y_n}{g(z_n)} \frac{\partial}{\partial z_i}\left(g(z_n)\right)$$

$$\frac{\partial}{\partial z_i}\left(g(z_n)\right) = \frac{\partial}{\partial z_i}\left(\frac{e^{z_n}}{\sum_{m=1}^{k} e^{z_m}}\right)$$

   if $n = i$, we derive as follows:

$$= \frac{e^{z_n} * \sum_{m=1}^{k} e^{z_m} - e^{z_n} * e^{z_n}}{\left(\sum_{m=1}^{k} e^{z_m}\right)^2}$$

$$= \frac{e^{z_n}\left(\sum_{m=1}^{k} e^{z_m} - e^{z_n}\right)}{\sum_{m=1}^{k} e^{z_m} \sum_{m=1}^{k} e^{z_m}}$$

$$= g(z_n)(1 - g(z_n))$$

   if $n \neq i$, we derive similarly:

$$= \frac{0 * \sum_{m=1}^{k} e^{z_m} - e^{z_n} * e^{z_i}}{\left(\sum_{m=1}^{k} e^{z_m}\right)^2}$$

$$= \frac{-e^{z_n} e^{z_i}}{\sum_{m=1}^{k} e^{z_m} \sum_{m=1}^{k} e^{z_m}}$$

$$= -g(z_n) g(z_i)$$

$$\frac{\partial L}{\partial z_i} = -\sum_{n=1}^{k} \frac{y_n}{g(z_n)} \frac{\partial}{\partial z_i}\left(g(z_n)\right)$$

$$= -\left[\sum_{n=1, n\neq i}^{k} \frac{y_n}{g(z_n)}(-g(z_n))g(z_i) + \frac{y_i}{g(z_i)} g(z_i)(1 - g(z_i))\right]$$

$$= -\left[-\sum_{n=1, n\neq i}^{k} y_n g(z_i) + y_i(1 - g(z_i))\right]$$

$$= -\left[-g(z_i)\sum_{n=1,n\neq i}^{k} y_n + y_i(1-g(z_i))\right]$$

$$= \left[g(z_i)\left(\sum_{n=1,n\neq i}^{k} y_n\right) - y_i(1-g(z_i))\right]$$

$$= \left[g(z_i)\left(\sum_{n=1,n\neq i}^{k} y_n\right) + y_i g(z_i) - y_i\right]$$

$$= \left[g(z_i)\left(\sum_{n=1}^{k} y_n\right) - y_i\right]$$

Since y is a one-hot vector, its summation equals one:

$$\frac{\partial L}{\partial z_i} = g(z_i) - y_i = \hat{y}_i - y_i$$

$$\frac{\partial L}{\partial z} = \hat{y} - y$$

Note that $\frac{\partial L}{\partial z}$ is a $k \times 1$ vector.
Now, calculating the gradient with respect to W_2:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W_2}$$

$$= (\hat{y} - y)\left(\frac{\partial}{\partial W_2} W_2 a\right)$$

$$= (\hat{y} - y)a^\top$$

Note that $\frac{\partial L}{\partial W_2}$ is a $k \times d_1$ matrix.
Now, to calculate the gradient with respect to W_1:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial W_1}$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial a} = W_2^\top(\hat{y} - y)$$

Note that $\frac{\partial L}{\partial a}$ is a $d_1 \times 1$ vector.

$$\frac{\partial a}{\partial W_1} = \frac{\partial}{\partial W_1}\sigma(W_1 x) = \sigma(W_1 x)(1 - \sigma(W_1 x))x$$

$$\frac{\partial L}{\partial W_1} = W_2^\top(\hat{y} - y)\sigma(W_1 x)(1 - \sigma(W_1 x))x$$

Note that $\frac{\partial L}{\partial W_1}$ is a $d_1 \times d$ matrix.
So, finally, our calculated weight updates for mini-batch SGD will be:

$$W_2 = W_2 - \frac{\alpha}{b}\sum_{n=1}^{b}(\hat{y}^{(n)} - y^{(n)})\sigma(W_1 x^{(n)})^\top$$

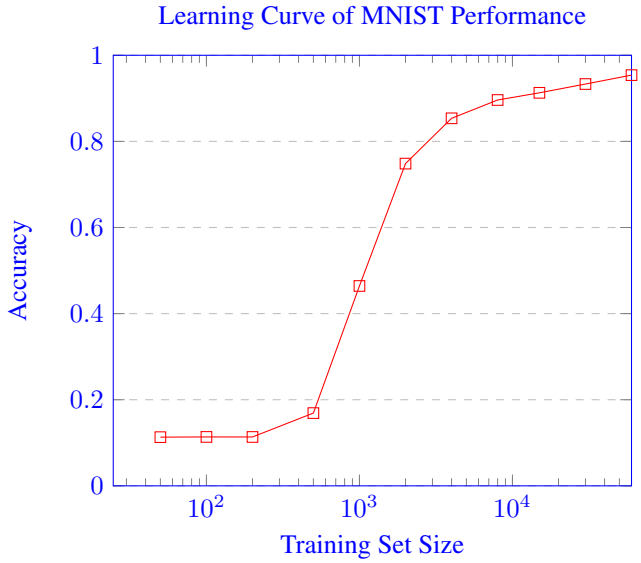
$$W_1 = W_1 - \frac{\alpha}{b}\sum_{n=1}^{b}\left(W_2^\top(\hat{y}^{(n)} - y^{(n)})\sigma(W_1 x^{(n)})(1 - \sigma(W_1 x^{(n)}))x^{(n)}\right)$$

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)

   For my implementation, I achieved an accuracy of 0.9539 using the full 60000 train set. My hidden layer had 100 neurons, my learning rate was 0.01, my batch size was 10, and I did 10 epochs (going through the entire training set 10 times). It took about 5 minutes to run.
   I tested the accuracy for other train set sizes and plotted the learning curve below.

   Learning Curve

train set size	accuracy
60000	0.9539
30000	0.9332
15000	0.9128
8000	0.8961
4000	0.8536
2000	0.7484
1000	0.4642
500	0.169
200	0.1135
100	0.1136
50	0.1131



   As observed in the learning curve, as we approach a training set size of 0, we approach an accuracy of 0.1 (which would approximately be the accuracy of random guessing).

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)

   I didn't get around to doing this, sorry. I would anticipate the implementation to be a lot more straightforward when using a library, and with its built-in optimizers available, it would likely outperform my personal implementation.

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)

   I didn't get around to doing this, sorry. I'd anticipate that random weight initialization would, to some significant degree, outperform weights initialized to zero because it's less likely to get stuck in local minima.

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)