HOMEWORK 4

>>Subham Sabud<<
>>9085909589<<

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.

Github repo:(https://github.com/SubhamSabud/hw4_neural_net_and_naive_bayes)

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)
In order to calculate the expected 0-1 loss for each of the two prediction strategies, we need to consider the probability of making an incorrect prediction for each strategy. The expected 0-1 loss is the expected value of the indicator function, which is 1 when the prediction is incorrect and 0 when the prediction is correct.
Let's calculate the expected 0-1 loss for each strategy:
Strategy 1: Maximum Likelihood Prediction

In this strategy, we predict the most probable outcome according to the parameter vector . The expected 0-1 loss can be calculated as follows:
$E[1\hat{x} \neq x] = p(x \neq argmax_x) = 1 - max_x(\theta_x)$

Strategy 2: Mimicking the World
In this strategy, we predict by randomly drawing from the same multinomial distribution that generated the true outcome. The expected 0-1 loss can be calculated as follows:
$E[1\hat{x} \neq x] = \sum_{i=1}^{k} p(x \neq \hat{x}/\hat{x} = i)p(\hat{x} = i)$
$= \sum_{i=1}^{k}(1 - \theta_i)\theta_i = 1 - \sum_{i=1}^{k} \theta_i^2$
Using the properties of the indicator function, we can see that the expected 0-1 loss for Strategy 2 is equal to 1 minus the probability of making the correct prediction:
The difference between the two strategies lies in how x is determined. In Strategy 1, x is the outcome with the highest probability, and in Strategy 2, x is randomly drawn from the same distribution. However, both strategies result in the same expected 0-1 loss, which is the probability of making an incorrect prediction.

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 $\hat{x}$. $E[c_{x\hat{x}}] = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}p(\hat{x}=j, x=i)$
$= \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}p(\hat{x}=j)\theta_i$
$= \sum_{j=1}^{k} p(\hat{x}=j)(\sum_{i=1}^{k} c_{ij}\theta_i) \geq \sum_{j=1}^{k} p(\hat{x}=j) min_w(\sum_{i=1}^{k} c_{iw}\theta_i)$
$= min_w(\sum_{i=1}^{k} c_{iw}\theta_i)$

$\epsilon argmin_w \sum_{i=1}^{k} c_{iw}\theta_i$,
$p(\hat{x}=\hat{w})=1, p(\hat{x}\neq\hat{w})=0$ minimizes the expected loss

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.)

   $\hat{p}(y=c_k) = \frac{count(y=c_k)+\lambda}{N+k\lambda}$

   here λ= smmothing parameter=0.5, N= total sample size and k= number of possible outcomes of y

   $\hat{p}(y=e) = \hat{p}(y=j) = \hat{p}(y=s) = 0.33$

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.

   $\hat{p}(X_i = j/Y = y) = \frac{b_j+\lambda}{\sum b_k + k\lambda}$

   here λ= smmothing parameter=0.5, k= number of possibilities=27

   $\theta_e : \begin{bmatrix} 0.067, 0.0075, 0.017, 0.022, 0.106, 0.016, 0.009, 0.056, 0.05, 0.0005, 0.0009, 0.03, 0.02, 0.054, 0.055, \\ 0.017, 0.0005, 0.045, 0.067, 0.092, 0.025, 0.0083, 0.013, 0.002, 0.0164, 0.0005, 0.194 \end{bmatrix}$

   Total 27 probabilities for A-Z and space ,respectively

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

   $\theta_s : \begin{bmatrix} [0.091, 0.011, 0.026, 0.041, 0.106, 0.008, 0.007, 0.0079, 0.049, 0.0046, 0.002, 0.046, 0.015, \\ 0.05, 0.0910.026, 0.013, 0.067, 0.054, 0.029, 0.0407, 0.0046, 0.00041, 0.0037, 0.0096, 0.0079, 0.175] \end{bmatrix}$

   $\theta_j : \begin{bmatrix} 0.132, 0.0028, 0.00282, 0.016, 0.04, 0.0009, 0.016, 0.025, 0.102, 0.0009, 0.051, 0.00094, 0.036, \\ 0.061, 0.078, 0.00094, 0.000942, 0.023, 0.053, 0.063, 0.081, 0.00094, 0.034, 0.00094, 0.012, 0.0179, 0.129 \end{bmatrix}$

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.

$$x = \begin{bmatrix} 34, 2, 9, 10, 69, 10, 6, 32, 26, 0, 0, 10, 12, 30, \\ 33, 14, 1, 28, 44, 52, 10, 8, 12, 2, 7, 1, 107 \end{bmatrix}$$

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.
The calculated likelihood probabilities are as follows:

$\hat{p}(x/y = e) = exp(-1577)$
$\hat{p}(x/y = j) = exp(-1800)$
$\hat{p}(x/y = s) = exp(-1710)$

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.
we know aposteriori probability is calculated as:
$\hat{p}(y/x) = \hat{p}(x/y) * \hat{p}(y)$
so
$\hat{p}(y = e/x) = exp(-1577) * 0.33$
$\hat{p}(y = j/x) = exp(-1800) * 0.33$
$\hat{p}(y = s/x) = exp(-1710) * 0.33$
predicted class= $argmax(\hat{p}(y = c_k/x))$
$\hat{y} = e$
predicted class is 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.

	pred e	pred j	pred s
act e	10	4	5
act j	0	6	0
act s	0	0	5

Table 1

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.
If we take a test document and arbitrarily shuffle the order of its characters so that the words (and spaces) are scrambled beyond human recognition, this shuffling will not affect your Naive Bayes classifier's prediction on this document. This is because our Naive Bayes classifier is based on a character-based multinomial model, which assumes that the order of the characters does not matter, and only the frequency of each character in the document is relevant. Therefore, shuffling the characters will not change the bag-of-words vector x that represents the document, and hence will not change the likelihood probability $\hat{p}(x|y)$ for each class y.

The key mathematical step in the Naive Bayes model that justifies this answer is the application of the multinomial distribution to model the probability of a document given a class. The multinomial distribution

is a generalization of the binomial distribution, which models the probability of a sequence of independent trials with two possible outcomes. The multinomial distribution models the probability of a sequence of independent trials with more than two possible outcomes. In this case, the outcomes are the 27 characters in the vocabulary, and the trials are the positions in the document. The probability of a document x given a class y is given by:

$$\hat{p}(x|y) = \frac{n!}{x_1! \dots x_d!} \prod_{i=1}^{d} \theta_{i,y}^{x_i}$$

where n is the length of the document, d is the size of the vocabulary, x_i is the number of times character i appears in the document, and $\theta_{i,y}$ is the conditional probability of character i given class y. This formula shows that only the counts of each character matter, and not their order. Therefore, shuffling the characters will not affect this probability.

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)
   $dZ_2 = A_2 - Y$

   $dW_2 = \frac{1}{m} dZ_2 A_1^T$

   $dZ_1 = W_2^T dZ_2 * \sigma'(Z_1)$

   $dW_1 = \frac{1}{m} dZ_1 X^T$

   $W_2 = W_2 - dW_2$
   $W_1 = W_1 - dW_1$

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)

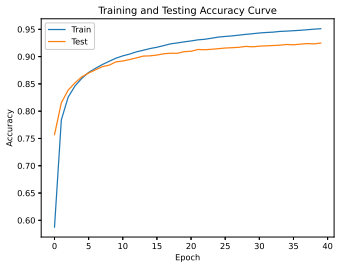

Figure 1: Accuracy curve for Question2

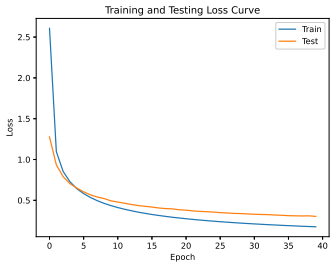

Figure 2: loss curve for Question2

For learning rate=0.01, batch size=16, this neural network model from scratch is achieving testing accuracy=0.9254 after 40 epochs.

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)

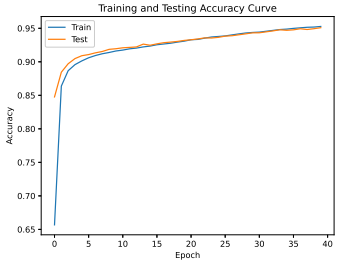

Figure 3: Accuracy curve for Question3

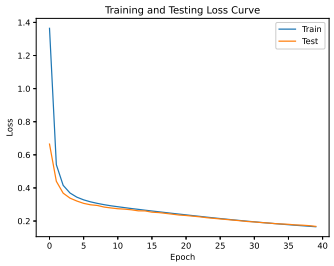

Figure 4: loss curve for Question3

For learning rate=0.01, batch size=16, this pytorch based neural network model is achieving testing accuracy=0.9509 after 40 epochs.

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) **a) all weights initialized to 0**

   a) all weights initialized to 0 For learning rate=0.01, batch size=16, this neural network model made from scratch is achieving testing accuracy=0.4322 after 40 epochs, and it is getting very low training accuracy as well

   b) initialize the weights randomly between -1 and 1 For learning rate=0.01, batch size=16, this pytorch based neural network model is achieving testing accuracy=0.9245 after 40 epochs.

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)

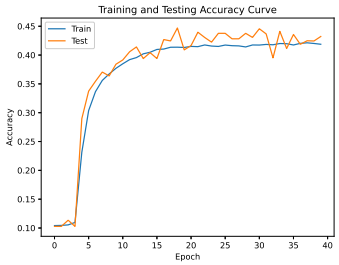

Figure 5: Accuracy curve for Question4.a

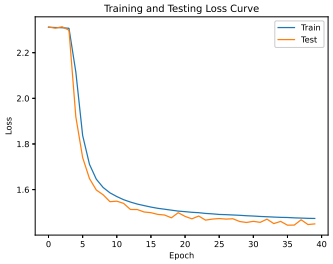

Figure 6: loss curve for Question4.a

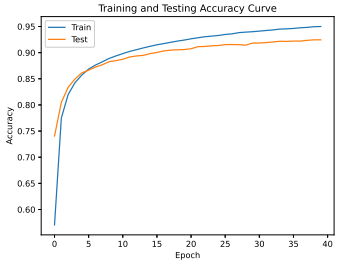

Figure 7: Accuracy curve for Question4.b

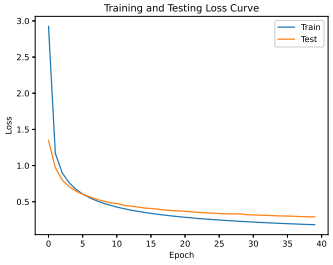

Figure 8: loss curve for Question4.b