Homework 4

Yuhan Wang
ywang2558

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 it 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 link: https://github.com/hanhuangv587/CS760hw4

1 Best Prediction

1.1 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.

1. Strategy 1: $\hat{x} \in \arg\max_x \theta_x$, the outcome with the highest probability.

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - P(x = \hat{x}) = 1 - \max_x \theta_x$$

2. 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)

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

1.2 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 of 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 $\hat{x}$.

$$\begin{aligned}
\mathbb{E}[c_{x\hat{x}}] &= \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij} P(x = i, \hat{x} = j) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{k} \theta_i c_{ij} P(\hat{x} = j | x = i) \\
&= \sum_{i=1}^{k}\sum_{j=1}^{k} \theta_i c_{ij} P(\hat{x} = j)
\end{aligned}$$

Let $p_1, \ldots, p_k$ be the probabilities of $P(\hat{x})$. Then the optimization problem is:

$$\begin{aligned}
&\min_{\hat{x}} \sum_{i=1}^{k}\sum_{j=1}^{k} \theta_i c_{ij} p_j \\
&\text{s.t.} \sum_{j=1}^{k} p_j = 1, \\
&p_j \geq 0, j = 1, \ldots, k
\end{aligned}$$

This is a linear programming problem. We can solve it by the simplex method. The optimal solution is:

$$\begin{aligned}
p_j = 1, j = \arg\min_i \sum_{i=1}^{k} \theta_i c_{ij}, \\
p_j = 0, j \neq \arg\min_i \sum_{i=1}^{k} \theta_i c_{ij}.
\end{aligned}$$

2 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 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 the 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!)
In the following questions, you may use the additive smoothing technique to smooth categorical data, in case the estimated probability is zero. Given N data samples $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} = [x_1^{(i)}, \ldots, x_j^{(i)}, \ldots, x_{M_i}^{(i)}]$ is a bag of characters, M_i is the total number of characters in $\boldsymbol{x}^{(i)}$, $x_j^{(i)} \in S, y^{(i)} \in L$ and we have $|S| = K_S, |L| = K_L$. Here S is the set of all character types, and L is the set of all classes of data labels. Then by the additive smoothing with parameter α, we can estimate the conditional probability as

$$P_\alpha(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + \alpha}{(\sum_{b_s \in S} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + K_S \alpha},$$

where $a_s \in S, c_k \in L$. Similarly, we can estimate the prior probability

$$P_\alpha(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + \alpha}{N + K_L \alpha},$$

where $c_k \in L$ and N is the number of training samples.

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 the prior probabilities.

   Since the numbers of files in each language are the same in the training data, $\hat{p}(y = e) = \hat{p}(y = j) = \hat{p}(y = s) = \frac{1}{3}$

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 which is a vector with 27 elements.

   θ_e = [0.0616 0.012 0.0217 0.0219 0.1075 0.0202 0.0145 0.0478 0.0524 0.0006 0.0043 0.031 0.0228 0.056 0.0663 0.0155 0.0008 0.0488 0.0622 0.0868 0.025 0.009 0.0161 0.0011 0.0134 0.0005 0.1802]

3. Print θ_j, θ_s, the class conditional probabilities for Japanese and Spanish.

   θ_j = [1.3168e-01 9.1100e-03 5.7000e-03 1.5940e-02 5.9950e-02 3.2400e-03 1.6140e-02 3.1150e-02 1.0035e-01 2.2900e-03 5.7020e-02 9.2000e-04 4.1190e-02 5.6000e-02 8.9850e-02 9.9000e-04 3.0000e-05 4.3440e-02 4.1530e-02 5.9410e-02 6.9170e-02 1.7000e-04 2.0510e-02 3.0000e-05 1.4090e-02 7.4000e-03 1.2274e-01]
   θ_s = [0.1035 0.0103 0.0365 0.0422 0.1149 0.0072 0.0067 0.005 0.0503 0.0064 0.0002 0.0511 0.0244 0.0557 0.0703 0.0251 0.0081 0.0578 0.0676 0.035 0.0363 0.0059 0.0004 0.0023 0.0059 0.004 0.1671]

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.

   x = [126 16 31 37 166 32 21 57 98 0 7 57 43 100 90 29 0 84 105 118 46 15 24 3 23 1 302]

5. For the x of e10.txt, 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. Also, Store all probabilities here and below in log() internally to avoid underflow. This also means you need to do arithmetic in log space.

$$\hat{p}(x \mid y = e) = e^{-4598.296}$$
$$\hat{p}(x \mid y = j) = e^{-5109.846}$$
$$\hat{p}(x \mid y = s) = e^{-4838.794}$$

6. For the x of e10.txt, use the 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 predicted class label of x 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.

	English	Spanish	Japanese
English	10		
Spanish		10	
Japanese			10

8. Take a test document. 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.

The shuffling of the characters in the document will not affect the prediction of the Naive Bayes classifier. The Naive Bayes classifier is based on the conditional independence of the characters in the document. The shuffling of the characters in the document will not affect the conditional independence of the characters in the document.

3 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.

$$\hat{y} = f(x) = g(W_3\sigma(W_2\sigma(W_1x)))$$

Suppose $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, $W_2 \in \mathbb{R}^{d_2 \times d_1}$, $W_3 \in \mathbb{R}^{k \times d_2}$ 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,

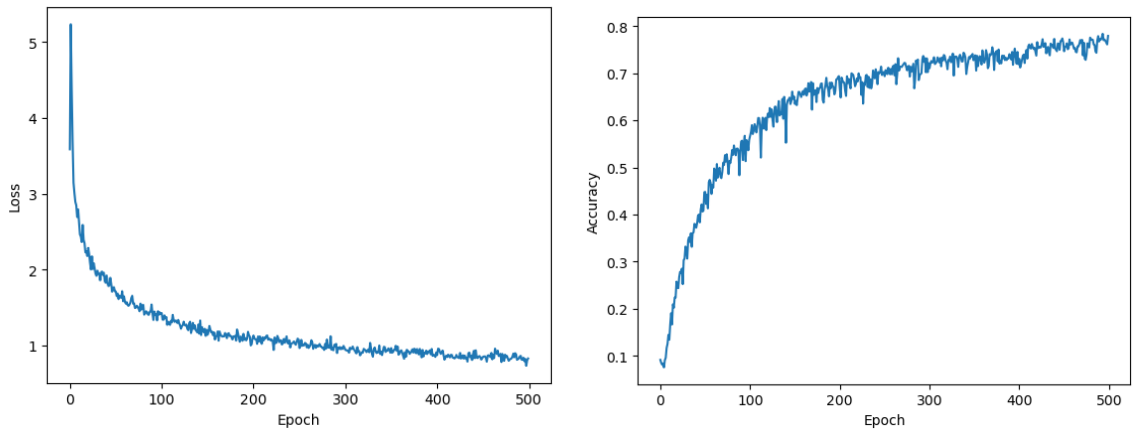
$$L(x, y) = -\sum_{i=1}^{k} y\log(\hat{y})$$

1. Derive backpropagation updates for the above neural network. (5 pts)
   Let $z_1 = W_1x$, $z_2 = W_2\sigma(z_1)$, $z_3 = W_3\sigma(z_2)$, $\hat{y} = g(z_3)$, $h_1 = \sigma(z_1)$ and $h_2 = \sigma(z_2)$.

$$\begin{aligned}
\frac{\partial L}{\partial W_3} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_3} \\
&= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial W_3} \\
&= (\hat{y} - y)\sigma(z_2)^T \\
\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} \\
&= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial \sigma(z_2)} \frac{\partial \sigma(z_2)}{\partial W_2} \\
&= W_3^T(\hat{y} - y)\sigma(z_2)^T(1 - \sigma(z_2))\sigma(z_1)^T \\
\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_1} \\
&= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial \sigma(z_2)} \frac{\partial \sigma(z_2)}{\partial W_1} \\
&= W_2^T W_3^T(\hat{y} - y)\sigma(z_2)^T(1 - \sigma(z_2))\sigma(z_1)^T(1 - \sigma(z_1))x^T \\
&= W_2^T W_3^T(\hat{y} - y)\sigma(z_2)^T(1 - \sigma(z_2))\sigma(z_1)^T(1 - \sigma(z_1))x^T
\end{aligned}$$

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)
   test error = 0.2628
   optimizer = SGD
   batch size = 64
   learning rate = 0.004

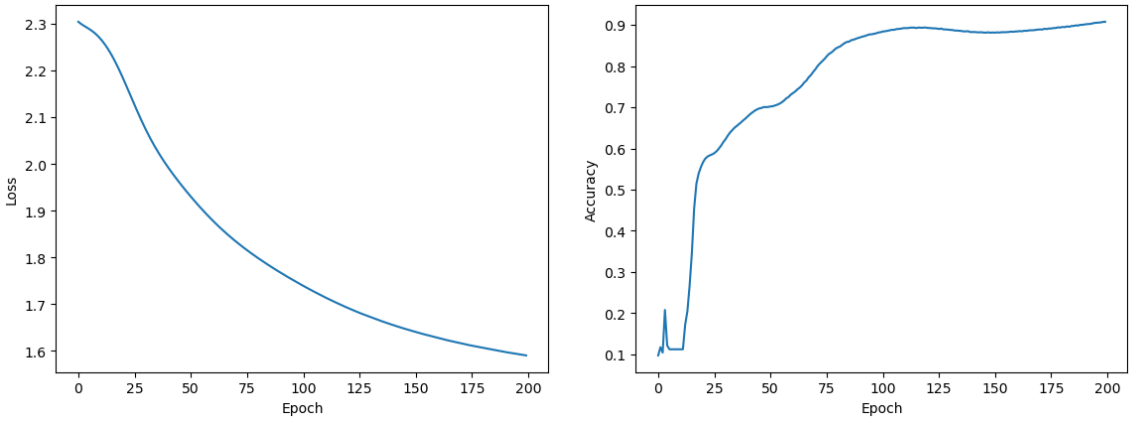
epochs = 500



(i) epoch vs loss (ii) epoch vs accuracy

Fig. 1: Learning curve for hand implementation, initialize the weights randomly between -1 and 1

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)
   test error = 0.1168
   optimizer = Adam
   epochs = 200



(i) epoch vs loss (ii) epoch vs accuracy

Fig. 2: Learning curve for PyTorch implementation

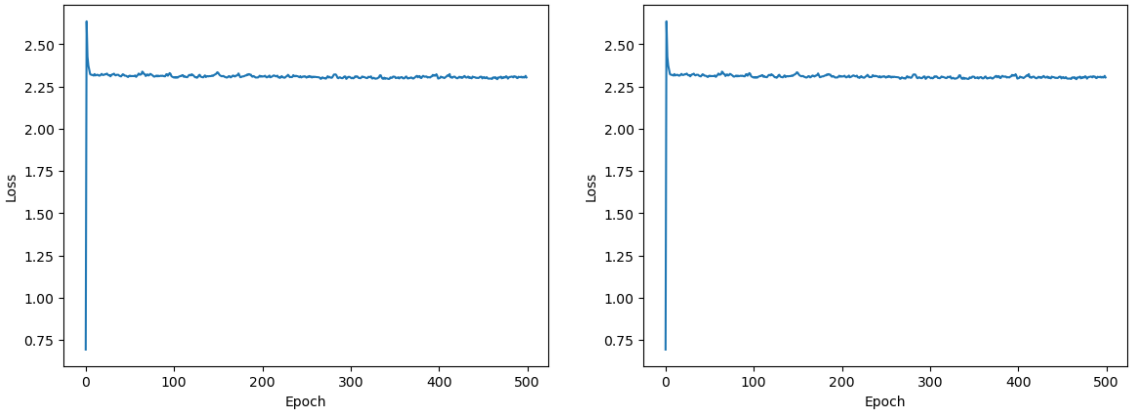
4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly

between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)

b) is shown in q3.2 with learning rate = 0.004, batch size = 64, epochs = 500

a) is shown below with learning rate = 0.004, batch size = 64, epochs = 500

test error = 0.9042



(i) epoch vs loss

(ii) epoch vs accuracy

Fig. 3: Learning curve for all weights initialized to 0

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 $d_1 = 300$, $d_2 = 200$, $d_3 = 100$. 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)