HOMEWORK 4

>>Yuanjun Ge<<
>>9082698615<<

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.
$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{x=1}^{k} P(x \neq \hat{x}) = 1 - P(x = \hat{x})$
And when we use this strategy, we can get $P(x = \hat{x}) = \theta_{\max}$.
So,

$$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \theta_{\max}$$

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)
When we apply the second strategy, since the the randomness is independent. We have $P(\hat{x} = x) = \sum_{i=1}^{k} P(x = x_i) * P(x = x_i)$ and $P(x = x_i) = \theta_i$.
So,

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

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 $\hat{x}$.
$\mathbf{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} P(x = x_i) * P(\hat{x} = x_j) * c_{ij}$
let $P(x = x_i) = \theta_i$, and $P(x = x_j) = \theta_j$
Thus,

$$\mathbf{E}[c_{x\hat{x}}] = \sum_{i=1}^{k} \sum_{j=1}^{k} \theta_i \theta_j c_{ij}$$

To minimize the expected loss, we need to choose $\hat{x} = j$ such that $(\sum_{i=1}^{k} \theta_i c_{ij})$ is minimum. Therefore, $\hat{x} \in \arg\min_j \sum_{i=1}^{k} \theta_i c_{ij}$.

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.)

   $$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha d}$$

   .
   Since we have 30 data points, so $N = 30$, for each language $x_i = 10$, and $\alpha = 0.5$, $d = 3$ (three dimensional multinomial distribution).

   $$\hat{p}(y = e) = \hat{p}(y = j) = \hat{p}(y = s) = \frac{10 + \frac{1}{2}}{30 + 3 \cdot \frac{1}{2}} = \frac{1}{3}$$

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 and

include in final report which is a vector with 27 elements.

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

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

$$\theta_j = \begin{bmatrix} 0.1317656102589189 \\ 0.010866906600510151 \\ 0.005485866033054963 \\ 0.01722631818022992 \\ 0.06020475907613823 \\ 0.003878542227191726 \\ 0.014011670568503443 \\ 0.03176211607673224 \\ 0.09703343932352633 \\ 0.0023411020650616725 \\ 0.05740941332681086 \\ 0.001432614696530277 \\ 0.03979873510604843 \\ 0.05671057688947902 \\ 0.09116321324993885 \\ 0.0008735455466648031 \\ 0.00010482546559977637 \\ 0.04280373178657535 \\ 0.0421747789929767 \\ 0.056990111464411755 \\ 0.07061742199238269 \\ 0.0002445927530661449 \\ 0.01974212935462455 \\ 3.4941821866592126e-05 \\ 0.01415143785596981 \\ 0.00772214263251686 \\ 0.12344945665466997 \end{bmatrix} \quad \theta_s = \begin{bmatrix} 0.10456045141993771 \\ 0.008232863618143134 \\ 0.03752582405722919 \\ 0.039745922111559924 \\ 0.1138108599796491 \\ 0.00860287996053159 \\ 0.0071844839813758445 \\ 0.0045327001942585795 \\ 0.049859702136844375 \\ 0.006629459467793161 \\ 0.0002775122567913416 \\ 0.052943171656748174 \\ 0.02580863988159477 \\ 0.054176559464709693 \\ 0.07249236841293824 \\ 0.02426690512164287 \\ 0.007677839104560451 \\ 0.05929511886774999 \\ 0.06577040485954797 \\ 0.03561407295488884 \\ 0.03370232185254849 \\ 0.00588942678301625 \\ 9.250408559711388e-05 \\ 0.0024976103111220747 \\ 0.007862847275754679 \\ 0.0026826184823163022 \\ 0.16826493170115014 \end{bmatrix}$$

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.

$$x_e^{10} = \begin{bmatrix} 164 \\ 32 \\ 53 \\ 57 \\ 311 \\ 55 \\ 51 \\ 140 \\ 140 \\ 3 \\ 6 \\ 85 \\ 64 \\ 139 \\ 182 \\ 53 \\ 3 \\ 141 \\ 186 \\ 225 \\ 65 \\ 31 \\ 47 \\ 4 \\ 38 \\ 2 \\ 498 \end{bmatrix}$$

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.

$$\log \hat{p}(\boldsymbol{x} \mid y = e) = -7841.87$$
$$\log \hat{p}(\boldsymbol{x} \mid y = j) = -8771.43$$
$$\log \hat{p}(\boldsymbol{x} \mid y = s) = -8467.28$$
$$\hat{p}(\boldsymbol{x} \mid y = e) = e^{-7841.87}$$
$$\hat{p}(\boldsymbol{x} \mid y = j) = e^{-8771.43}$$
$$\hat{p}(\boldsymbol{x} \mid y = s) = e^{-8467.28}$$

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.
By Bayes rule,

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

Then the log-probabilities could be computed as:

$$\log \hat{p}(y \mid \boldsymbol{x}) = \log \hat{p}(\boldsymbol{x} \mid y) + \log \hat{p}(y) - \log \hat{p}(\boldsymbol{x})$$

where, $\hat{p}(y)$ is the prior probability of y, which is $1/3$. And $\hat{p}(\boldsymbol{x})$ could be calculated as $\sum_{y'} \hat{p}(\boldsymbol{x} \mid y')\,\hat{p}(y')$. we could calculate all components in the functions with the posterior probabilities we get in the last question. Using all the components into the log-probabilities and then exponential it and normalize themtosumto 1. We can get

$$\hat{p}(y = e \mid \boldsymbol{x}) \approx 1$$
$$\hat{p}(y = j \mid \boldsymbol{x}) \approx 0$$
$$\hat{p}(y = s \mid \boldsymbol{x}) \approx 0$$

To predict the class, we need to select $\hat{y}$ that maximizes $\hat{p}(y \mid \boldsymbol{x})$. And we can see the $\hat{p}(\boldsymbol{x})$ has nothing to do with the maximization process. So, we only need to maximize the $\hat{p}(\boldsymbol{x} \mid y)\hat{p}(y)$. That is

$$\hat{y} = \underset{y \in \{e,j,s\}}{\operatorname{argmax}} \log \hat{p}(y \mid \boldsymbol{x})$$

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

We can see when $y = e$, we have largest $\hat{p}(\boldsymbol{x} \mid y)\hat{p}(y)$. So, the predicted class label of x is 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.

	English	Spanish	Japanese
English	10	0	0
Spanish	0	10	0
Japanese	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.
Shuffling has no impact on the performance of a Naive Bayes classifier. In my code file, I shuffled e14.txt and then make the prediction. The result shows the prediction is same as that before shuffling. This is beacuse our Naive Bayes classifier creates a bag of words, a vector that captures character frequency without regard for their positions in the text. Consequently, the algorithm computes conditional probabilities for each feature (character) given the class labels, independently of the character order of the input data.

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)
   Let z becomes the weighted input to the final softmax layer.

$$\frac{\partial \ell}{\partial z_j} = -\frac{\partial}{\partial z_j} \sum_{i=1}^{k} y_i \cdot \frac{\partial}{\partial z_j} (\log \hat{y})$$

   We have $\hat{y} = z_i - \log\left(\sum_{j=1}^{k} e^{z_j}\right)$, and y is one hot encoded, we know $\sum_{i=1}^{k} y_i = 1$. After plugging all, we have

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

   Then, we can calculate the gradient w.r.t to the weights of the second layer W2:

$$\frac{\partial \ell}{\partial W_2} = \frac{\partial \ell}{\partial z} * \frac{\partial z}{\partial W_2} = (\hat{y} - y)^T a$$

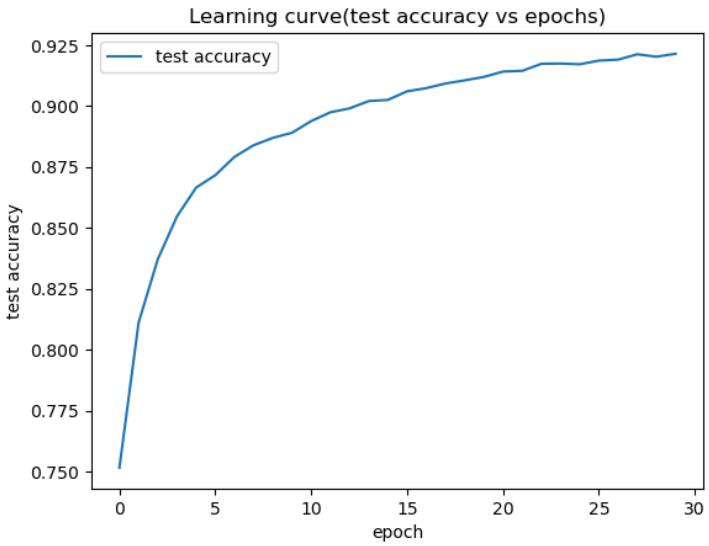
   where $z = W_2 a$, a is the output of the first layer.
   Then we calculate the gradient for W1:

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial \ell}{\partial z} * \frac{\partial z}{\partial a} * \frac{\partial a}{\partial z_1} * \frac{\partial z_1}{\partial W_1}$$

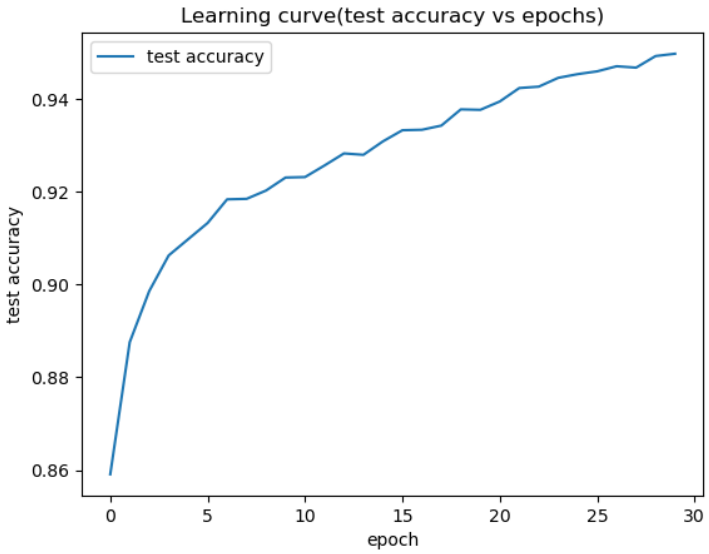
   Where $z1 = W_1 X$ is the weighted input to the sigmoid function. We have $\frac{\partial z}{\partial a} = W_2^T$, $\frac{\partial a}{\partial z_1} = \sigma'(z_1) = \sigma(z_1) * \sigma(1 - z_1)$, and $\frac{\partial z_1}{\partial W_1} = X$. So,

$$\frac{\partial \ell}{\partial W_1} = (\hat{y} - y) W_2^T \sigma(z_1) \sigma(1 - z_1) X$$

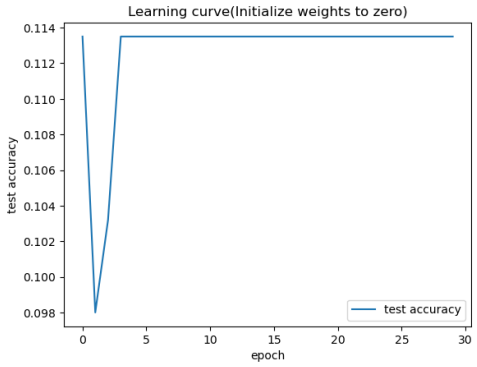
2. Implement it in NumPy or PyTorch using basic linear algebra operations. (e.g. You are not allowed to use auto-grad, built-in optimizer, model, etc. in this step. You can use library functions for data loading, processing, etc.). Evaluate your implementation on MNIST dataset, report test errors and learning curve. (10 pts)
   batch size = 32, learning rate = 0.01, Epochs =30
   The weights were initialized by sampling randomly from a normal distribution from -1 to 1. The accuracy on the test set was 92.14% and test error is 7.86%.



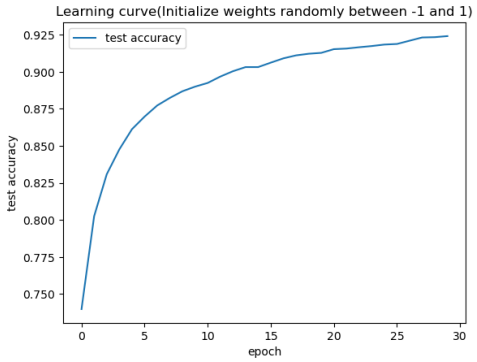
3. Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)
   batch size = 32, learning rate = 0.01, Epochs =30
   Using PyTorch with the same hyperparameters, default random weight setup. The accuracy on the test set was 94.98% and test error is 5.02%.



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) Use the Pytorch implementation.
batch size = 32, learning rate = 0.01, Epochs =30
a) all weights initialized to 0
The accuracy on the test set was 11.35% and test error is 88.65%.



b) initialize the weights randomly be- tween -1 and 1.
The accuracy on the test set was 92.41% and test error is 7.59%.



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)