HOMEWORK 4

Anshuman Senapati
908 590 6916

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 link: https://github.com/Anshu1245/cs760-hw4

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)
Solution:
For strategy 1:
$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{i \neq argmax(\theta)} 1 \cdot P(x = i) = 1 - P(x = argmax(\theta)) = 1 - max(\theta)$

For strategy 2:
$\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{i=1}^{k} 1 \cdot P(x = i) \cdot P(\hat{x} \neq i)$, as the outcomes are independent.
$\implies \mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = \sum_{i=1}^{k} \theta_i \cdot (1 - \theta_i) = 1 - \sum_{i=1}^{k} \theta_i^2$

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 $\hat{x}$.

$\mathbb{E}[c_{x\hat{x}}] = \sum_i \sum_j c_{ij} P(x = i) P(\hat{x} = j) = \sum_j P(\hat{x} = j) \sum_i c_{ij} \cdot \theta_i$

For each *j*, different costs are weighted by the same probability θ_i. So, one good strategy could be to find that action *j* that minimizes this weighted cost.

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.)

   Solution:
   There are 10 documents of each language in the training set.
   $\implies \hat{p}(y = e) = \frac{10+1/2}{30+3*1/2} = 0.33 = \hat{p}(y = j) = \hat{p}(y = s)$

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.

   Solution:

   $\hat{p}(c_i|y = e) = \frac{count(c_i)+1/2}{\sum_i count(c_i)+27*1/2}$

   $\theta_e = (0.0645, 0.0267, 0.0538, 0.1792, 0.0662, 0.0579, 0.0554, 0.0602, 0.0205, 0.1054, 0.029, 0.0215,$
   $0.0801, 0.0175, 0.0189, 0.0472, 0.0093, 0.0155, 0.022, 0.0138, 0.0168, 0.0037, 0.0111, 0.0006, 0.0014, 0.0012, 0.0006)$

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

   Solution:

   $\theta_j = (0.0398, 0.1318, 0.0172, 0.1234, 0.0602, 0.0574, 0.097, 0.0567, 0.0142, 0.0912, 0.014, 0.057, 0.0039,$
   $0.0706, 0.0197, 0.0422, 0.0428, 0.0318, 0.0055, 0.0109, 0.0009, 0.0014, 0.0077, 0.0023, 0.0002, 0.0001)$

   $\theta_s = (0.0375, 0.0725, 0.0542, 0.1683, 0.1138, 0.0529, 0.0593, 0.0658, 0.0337, 0.0356, 0.1046, 0.0397,$
   $0.0499, 0.0059, 0.0072, 0.0003, 0.0086, 0.0066, 0.0243, 0.0077, 0.0027, 0.0025, 0.0258, 0.0001, 0.0045, 0.0079, 0.0082)$

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.

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 log probabilities are coming out to be really large negative numbers (~ -8000). The numbers would then be,
   $\hat{p}(x \mid y = e) = e^{-7841.8}$
   $\hat{p}(x \mid y = j) = e^{-8771.3}$
   $\hat{p}(x \mid y = s) = e^{-8467.0}$

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.

   $\hat{p}(y = e \mid x) = \frac{\hat{p}(x|y=e)\cdot p(y=e)}{\hat{p}(x|y=e)\cdot p(y=e)+\hat{p}(x|y=s)\cdot p(y=s)+\hat{p}(x|y=j)\cdot p(y=j)} = \frac{e^{-7841.8}*0.33}{e^{-7841.8}*0.33+e^{-8771.3}*0.33+e^{-8467.0}*0.33}$

   Similarly for the other classes. The predicted class label should be "English" as seen from the numbers above.

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.

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.

   Solution:
   This random shuffling does not affect the prediction of the Naive Bayes classifier. This is because of the conditional independence assumption in the modeling step. Conditional independence no more requires us to model every possible outcome of the joint distribution of $(X_1, ..., X_k, Y)$ and hence we are ignoring the sequential nature of the text data, which is why the random shuffling should not affect the classifier.

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)
   Solution:
   $h_1 = W^{(1)}x$
   $a_1 = \sigma(h_1)$
   $h_2 = W^{(2)}a_1$
   $\hat{y} = g(h_2)$

   $\frac{\partial L}{\partial h_{2i}} = \sum_i \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_{2i}} = \sum_{j \neq i}(\frac{-y_j}{\hat{y}_j} \cdot (-\hat{y}_j \cdot \hat{y}_j)) + (\frac{-y_i}{\hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i)) = \hat{y}_i - y_i$

   $h_{2i} = \sum_j a_{1j} w_{ij}^{(2)} \implies \frac{\partial L}{\partial w_{ij}^{(2)}} = \frac{\partial L}{\partial h_{2i}} \cdot \frac{\partial h_{2i}}{\partial w^{(2)}{}_{ij}} = (\hat{y} - y)a^T$

   $\frac{\partial L}{\partial h_{1i}} = \frac{\partial L}{\partial a_{1i}} \cdot \frac{\partial a_{1i}}{\partial h_{1i}} = (\sum_{j=1}^{k} \frac{\partial L}{\partial h_{2j}} \cdot \frac{\partial h_{2j}}{\partial a_{1i}}) \cdot a_{1i}(1 - a_{1i}) = \sum_{j=1}^{k}(\hat{y}_j - y_j) \cdot w_{ji}^{(2)} = W_2^T \cdot (\hat{y} - y)$

   $\frac{\partial L}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial h_{1i}} \cdot \frac{\partial h_{1i}}{\partial w_{ij}^{(1)}} = \frac{\partial L}{\partial h_1} \cdot X^T$

   The code for it is pushed to the repo in the link provided. Note: the derivations here are not in explicit details because it takes up a lot of time to type these up in LaTeX.

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)

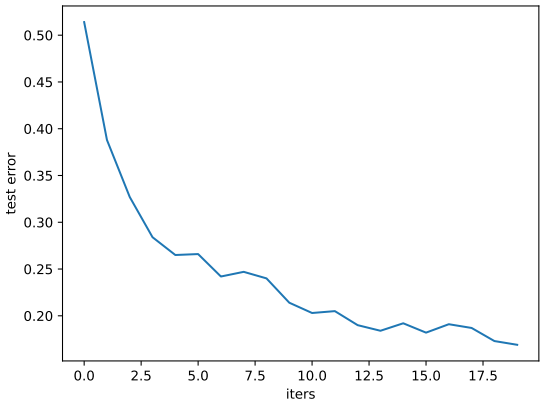

Figure 1: Test error for Numpy implementation

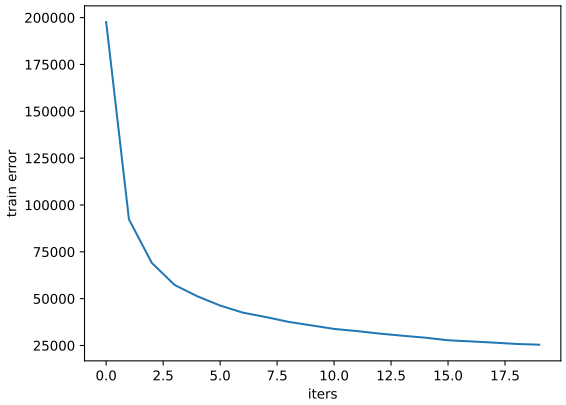

Figure 2: Training curve. Learning rate is 0.0003

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)

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)

   For (a) weights initialized to 0, the results are as follows,

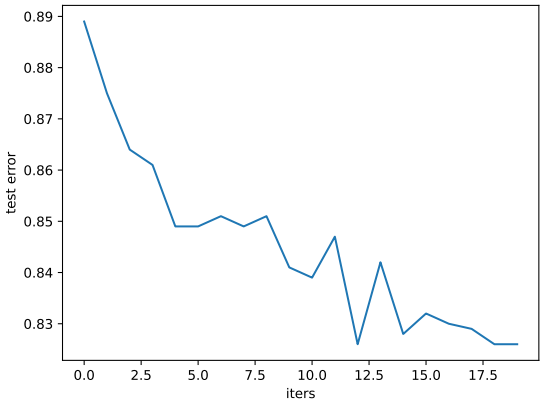

Figure 3: Test error for zero init weights

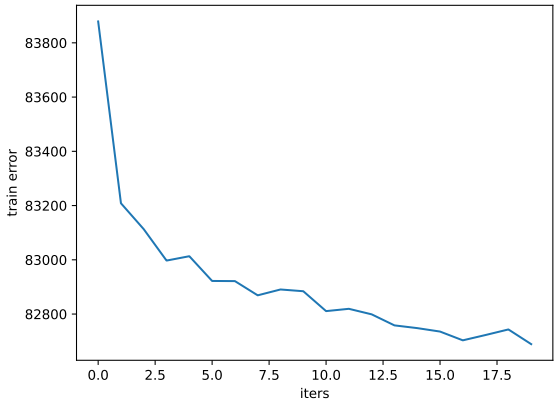

Figure 4: Train error for zero init weights

The learning is much worse in zero weights initialization as is clearly visible from the above plots.

For (b) weights randomly chosen from (-1, 1) the results are as follows,

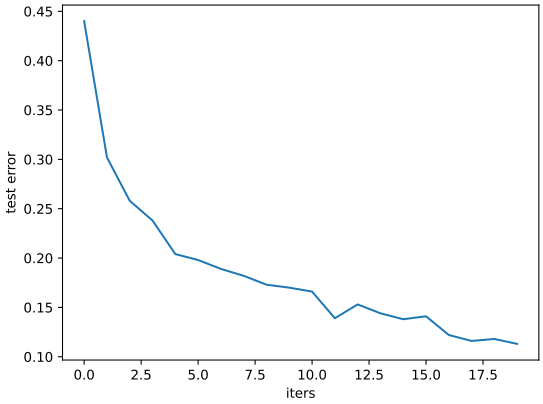

Figure 5: Test error

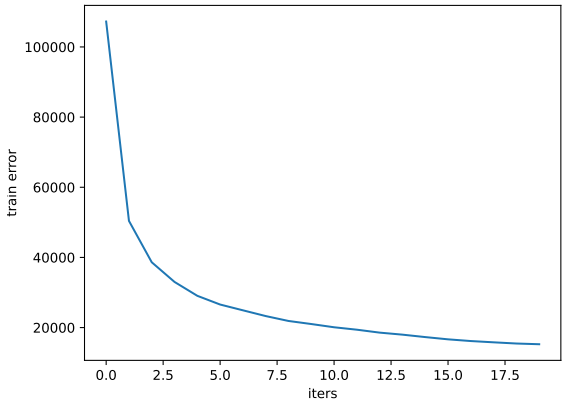

Figure 6: Train error

Again, as can be seen from the above plot this has much better test accuracy than the one initialized with zero weights.

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)