HOMEWORK 4

>>Hemanth Sridhar Nakshatri<<
>>nakshatri@wisc.edu, 9085807346<<

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.

$$\text{Let } L(x, \hat{x}) = \begin{cases} 1 & \text{if } \hat{x} = x \\ 0 & \text{Otherwise} \end{cases}$$

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

For this, it's a fixed prediction. Always predict the outcome with the highest probability.
If the outcome is denoted by $\hat{x}$, then the Expected loss,

$$\mathbb{E}(L(x, \hat{x})) = P(x \neq \hat{x}) = (1 - \theta_{\hat{x}})$$

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)
Here $P(\hat{x} = x|x) = 1 - \theta_x$
Thus the expected loss is given by,

$$\mathbb{E}(L(x, \hat{x})) = \sum_{i=1}^{k} P(x = i)P(\hat{x} \neq i|x = i)\mathbb{E}(L(x, \hat{x})) = \sum_{i=1}^{k} \theta_i(1 - \theta_i)$$

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 $\hat{x}$.
Given:

- $x \sim multinomial(\theta)$
- $c_{ij} \geq 0$
- $c_{ii} = 0$

Expected loss is given by,

$$E[L(x,j)] = \sum_{i=1}^{k} P(x=i) \cdot c_{ij} = \sum_{i=1}^{k} \theta_i \cdot c_{ij}$$

For $\hat{x}$, we need minimze the loss for all possible values of j.

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

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.)

   With additive smoothing parameter = 0.5,
   $P(x = c|y = l) = \theta_{c_i,l} = \dfrac{count(c,l) + 0.5}{count(l) + (0.5 * 3)}$
   where,

   - *c* is one of the 27 characters,
   - *l* is the language (E,S or J).
   - *count(c,l)* = frequency of *c* in Language *l*.
   - *count(l)* = total characters in *l*

   Prior probabilities = {1/3, 1/3, 1/3} (*Included in code as well*)
   From code: ***Prior Probabilities = {'e': 0.3333333333333333, 'j': 0.3333333333333333, 's': 0.3333333333333333}***

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.

   English- Class Conditional Probability =
   'a': 0.05946366109599689,
   'b': 0.011553545560541285,
   'c': 0.022294456418047556,
   'd': 0.02187047309472494,

'e': 0.10504186835317811,
'f': 0.019114581493127936,
'g': 0.018125287072041835,
'h': 0.047097480832420595,
'i': 0.05564781118609335,
'j': 0.0013072819135780658,
'k': 0.009321845740734198,
'l': 0.029431509027311592,
'm': 0.020174539801434476,
'n': 0.0588983499982334,
'o': 0.06476345263752958,
'p': 0.01706532876373529,
'q': 0.000600643041373706,
'r': 0.05352789456948027,
's': 0.06561141928417483,
't': 0.07953220506660072,
'u': 0.026604953538494153,
'v': 0.009504292831148642,
'w': 0.0155107232448857,
'x': 0.00123661802635763,
'y': 0.013461470515493058,
'z': 0.000600643041373706,
'': 0.17888563049853373

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

Spanish - Class Conditional Probability =
'a': 0.10569930661254862,
'b': 0.008286825638423811,
'c': 0.037104684593269065,
'd': 0.03879587349906985,
'e': 0.11354642313546423,
'f': 0.008489768307119906,
'g': 0.007475054963639438,
'h': 0.004701505158126163,
'i': 0.04982242516489092,
'j': 0.006798579401319127,
'k': 0.00030441400304414006,
'l': 0.05340774564518857,
'm': 0.025334009808895653,
'n': 0.05408422120750888,
'o': 0.0726196516150854,
'p': 0.02445459157787925,
'q': 0.007948587857263656,
'r': 0.05902249281244715,
's': 0.0654490106544901,
't': 0.03588702858109251,
'u': 0.033789954337899546,
'v': 0.005851513614070692,
'w': 0.00010147133434804668,
'x': 0.0025367833587011668,
'y': 0.007677997632335532,
'z': 0.0028750211398613224,
' ': 0.16874682902080163

Japanese - Class Conditional Probability =
'a': 0.13259992953059546,
'b': 0.011157655717809184,

'c': 0.0051286066632736955,
'd': 0.017734800140938808,
'e': 0.060016442861057825,
'f': 0.003954116587714834,
'g': 0.014211329914262225,
'h': 0.03151548369416279,
'i': 0.09720862858708844,
'j': 0.002231531143561837,
'k': 0.058763653447128375,
'l': 0.0016051364365971107,
'm': 0.03887562150099832,
'n': 0.0568061699878636,
'o': 0.09172767490114708,
'p': 0.000822143052891203,
'q': 0.00011744900755588615,
'r': 0.04145949966722781,
's': 0.04247739106604549,
't': 0.056962768664604785,
'u': 0.06988215949575226,
'v': 0.0002740476842970677,
'w': 0.01992718161531535,
'x': 3.9112918997144755e-05,
'y': 0.01389813256077986,
'z': 0.007790784167873781,
' ': 0.12375210429471871

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.

Bag-of-Words Count Vector for e10.txt:
a: 164
b: 32
c: 53
d: 57
e: 311
f: 55
g: 51
h: 140
i: 140
j: 3
k: 6
l: 85
m: 64
n: 139
o: 182
p: 53
q: 3
r: 141
s: 186
t: 225
u: 65
v: 31
w: 47
x: 4
y: 38
z: 2
: 498

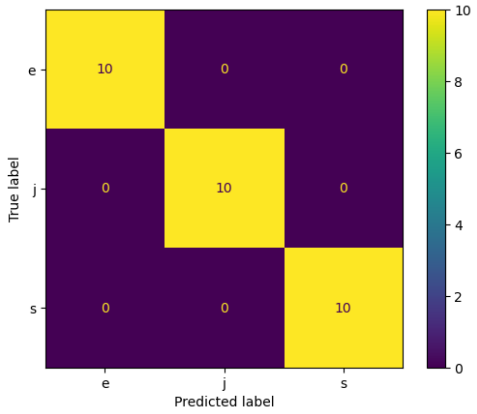

Figure 1: Confusion matrix plot

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.

$\hat{p}(x|y = e),\ \hat{p}(x|y = j),\ \hat{p}(x|y = s)$ is obtained from the code as shown below,

Log Likelihood for English: -7840.032136631908
Log Likelihood for Japanese: -8763.842626863328
Log Likelihood for Spanish: -8455.713565886454
Likelihood for English: 1.3101020210153362350944769 36E-3405
Likelihood for Japanese: 8.1565571740823875707055763 01E-3807
Likelihood for Spanish: 5.3735065308354823879510041 36E-3673

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.

Posteriors $\hat{p}(y = e|x),\ \hat{p}(y = j|x),\ \hat{p}(y = s|x)$ is given as,

Log Posterior for English: -7841.130748920576
Log Posterior for Japanese: -8764.941239151996
Log Posterior for Spanish: -8456.812178175122
Posterior for English: 4.3670067367185426824734985 49E-3406
Posterior for Spanish: 1.7911688436121372282160161 46E-3673
Posterior for Japanese: 2.7188523913612660564880216 86E-3807

Predicted class:
Predicted class label: 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.

Fig. 1 denotes the confusion matrix plot for the designed Naive Bayes classifier against the test dataset.

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 the characters or words in the test document doesn't affect the Naive Bayes classifier's performnce in any way.

   - One of the core principles of this classifier is it's "naive" assumption that every feature i.e every character's occurence is independent of each other.
   - Thus the individual probability of each character, .ie $P(c_1, c_2, ...c_n|y) = P(c_1|y) \times P(c_2|y).....P(c_n|y)$ will not change.
   - Thus shuffling the order of characters in a test document would not affect the Naive Bayes classifier's prediction because it only considers the frequency of each character type, not their order.
   - ***The key mathematical justification is the assumption of feature independence in the Naive Bayes 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.

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

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) **Forward Propagation** Let's break down the forward propagation steps:

   (a) **Linear Layer 1:**
   $a = W_1x$

   (b) **Activation Layer 1 (Sigmoid):**
   $z = \sigma(a)$
   Where $\sigma(x) = \frac{1}{1+e^{-x}}$

   (c) **Linear Layer 2:**
   $b = W_2z$

   (d) **Activation Layer 2 (Softmax):**
   $\hat{y} = g(b)$
   Where $g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

   Loss Function The given loss function is cross-entropy:
   $L(x, y) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$

   Backward Propagation

   (a) **Gradient w.r.t. Output of Softmax:**
   $\frac{\partial L}{\partial b} = \hat{y} - y$

   (b) **Gradient w.r.t. W_2:** Using the chain rule:
   $\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial b} \cdot z^T = (\hat{y} - y) \cdot z^T$

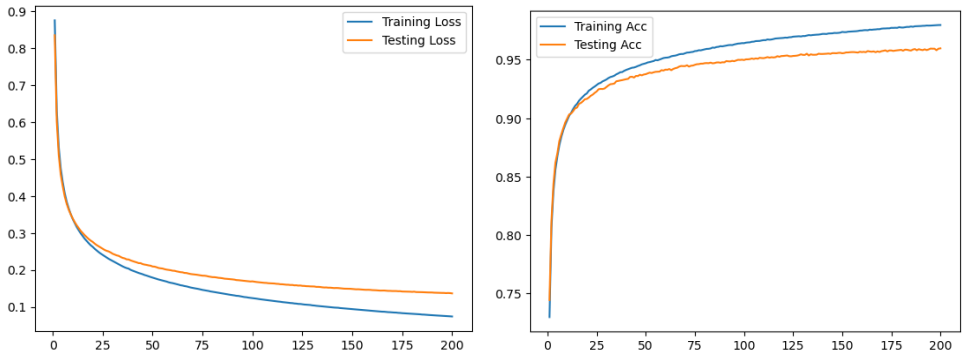

Figure 2: Plots for train vs test loss and accuracy for MNIST dataset using Numpy

(c) **Gradient w.r.t. z:**
$\frac{\partial L}{\partial z} = W_2^T \cdot (\hat{y} - y)$

(d) **Gradient w.r.t. Output of Sigmoid:**
The derivative of the sigmoid function $\sigma(x)$ is $\sigma(x)(1 - \sigma(x))$. So, the gradient w.r.t. the output of the sigmoid function is:
$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z} \cdot z \cdot (1 - z)$

(e) **Gradient w.r.t. W_1:** Using the chain rule:
$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a} \cdot x^T$
$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z} \cdot z \cdot (1 - z) \cdot x^T$

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)

   - Weights were initialized randomly between -1 to +1.
   - Number of hidden units = 200.
   - Learning rate = 0.01
   - Epochs = 200
   - After 200 epochs,
     - **Training Loss = 0.0739**
     - **Training Accuracy = 0.9798**
     - **Testing Loss = 0.1365**
     - **Testing Accuracy = 0.9598**

   Figure. 2 denotes the train-vs-test loss and accuracy variation with number of epochs. It can be seen that test accuracy begins to increase slowly compared to the training accuracy with consecutive epochs.

   The learning curve and the error rates are visualized in Fig. 2 as well.

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)

   The same model was trained using Tensorflow.

   - Number of hidden units = 200.
   - Learning rate = 0.01
   - Epochs = 200
   - After 200 epochs,
     - **Training Loss = 0.0683**
     - **Training Accuracy = 0.9814**
     - **Testing Loss = 0.0898**

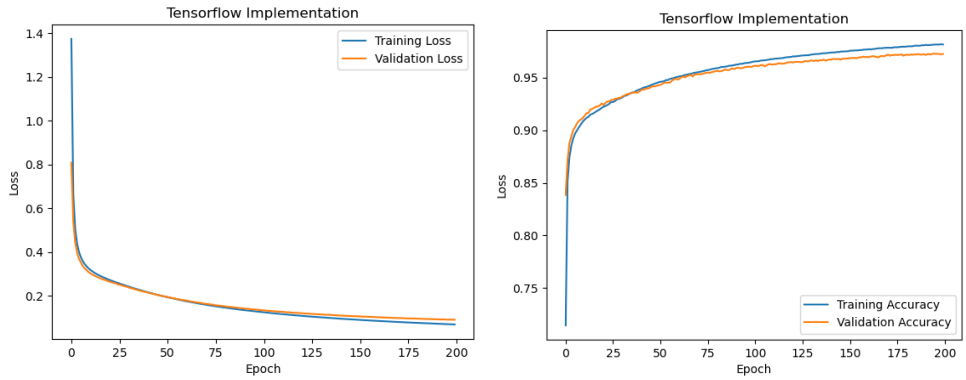

Figure 3: Plots for train vs test loss and accuracy for MNIST dataset using TENSORFLOW

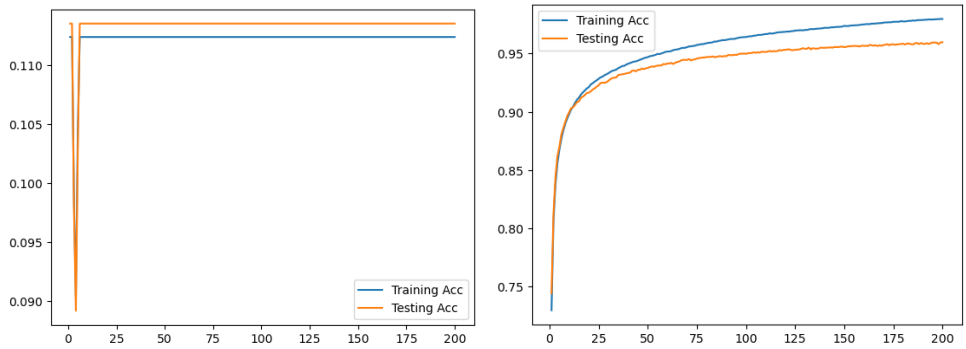

Figure 4: Weights initalized to 0 vs (-1 to +1).

– **Testing Accuracy = 0.9724**

Figure. 3 depicts the learning curve for the same model trained with Tensorflow. It can be seen that the learning curve is very similar to the model designed from scratch (Fig. 2).

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.

- The weights are not updated with each epoch since the dot product with 0 matrix results in 0.
- The model ends up predicting 0 for everything.

(b) For weights randomly initalized between -1 and +1, the model learns pretty well with each epoch. The batch size consider for the following results is 32.

Figure. 4 depicts the train and test accuracy or the learning curve for models with weights initialized to 0 or in the range -1 to +1.

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)