Homework 4

Sriram Ashokkumar
908 216 3750

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 $x \sim \text{multinomial}(\theta)$, where the parameter vector $\theta = (\theta_1, \dots, \theta_k)$ with $\theta_i \geq 0$ and $\sum_{i=1}^k \theta_i = 1$. Note $x \in \{1, \dots, k\}$. 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.

In Strategy 1, the prediction is made by selecting the category with the highest probability, which is given by $\hat{x} = \arg\max_x \theta_x$. We want to calculate the expected 0-1 loss for this strategy. The expected 0-1 loss is the probability of making an incorrect prediction, which is equivalent to the probability of $\hat{x} \neq x$.
We can express this probability using the law of total probability as follows:

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= \sum_x \mathbb{E}[\mathbb{1}_{\hat{x} \neq x} \mid x] \cdot P(x) \\ &= \sum_x \mathbb{1}_{\hat{x} \neq x} \cdot P(x)\end{aligned}$$

Now, because we are using Strategy 1, $\hat{x} = \arg\max_x \theta_x$. This means that $\hat{x}$ is the category with the highest probability, and for any $x' \neq \hat{x}$, $\theta_{\hat{x}} \geq \theta_{x'}$.
So, we can rewrite the above expression as:

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= \sum_x \mathbb{1}_{\hat{x} \neq x} \cdot P(x) \\ &= \sum_x \mathbb{1}_{\hat{x} \neq x} \cdot \theta_x \\ &= \theta_{\hat{x}}\end{aligned}$$

The final expression shows that the expected 0-1 loss for Strategy 1 is equal to the probability of selecting the correct category, which is $\theta_{\hat{x}}$. This result makes intuitive sense because in Strategy 1, you always choose the category with the highest probability, so the expected loss is simply the probability of that category being the correct one.
Strategy 2: You mimic the world by generating a prediction $\hat{x} \sim \text{multinomial}(\theta)$. (Hint: your randomness and the world's randomness are independent)
In Strategy 2, the prediction is made by generating a random variable $\hat{x}$ that follows the same multinomial distribution as the true observation x, which is $\hat{x} \sim \text{multinomial}(\theta)$. To calculate the expected 0-1 loss for this strategy, we need to find the probability of making an incorrect prediction, which is the probability that $\hat{x} \neq x$.
We can calculate the expected 0-1 loss as follows:

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] &= P(\hat{x} \neq x) \\ &= 1 - P(\hat{x} = x)\end{aligned}$$

Now, because $\hat{x}$ follows the same multinomial distribution as x, $P(\hat{x} = x)$ is simply the probability of observing the true category x under the distribution θ. Therefore:

$$P(\hat{x} = x) = \theta_x$$

Substituting this into the previous expression:

$$\begin{aligned}\mathbb{E}[\mathbb{1}_{\hat{x}\neq x}] &= 1 - P(\hat{x} = x)\\ &= 1 - \theta_x\end{aligned}$$

So, the expected 0-1 loss for Strategy 2 is given by:

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

This result shows that the expected 0-1 loss for Strategy 2 is equal to 1 minus the probability of selecting the correct category, which is θ_x. This makes sense because in Strategy 2, you mimic the randomness of the world, and the expected loss 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 $x \sim \text{multinomial}(\theta)$. 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 $\hat{x}$.
We want to minimize the expected loss, which is given by:

$$\begin{aligned}\mathbb{E}[c_{x\hat{x}}] &= \sum_{x,\hat{x}} c_{x\hat{x}} \cdot P(x, \hat{x})\\ &= \sum_{x,\hat{x}} c_{x\hat{x}} \cdot P(\hat{x} \mid x) \cdot P(x)\end{aligned}$$

Now, we can express $P(\hat{x} \mid x)$ as follows:

$$P(\hat{x} \mid x) = \begin{cases} 1 & \text{if } \hat{x} = x \\ 0 & \text{otherwise}\end{cases}$$

This is because we want to minimize the expected loss, so we will always choose the category $\hat{x}$ that minimizes the loss. If $\hat{x} = x$, then the loss is 0, so we will always choose $\hat{x} = x$. If $\hat{x} \neq x$, then the loss is $c_{x\hat{x}}$, so we will never choose $\hat{x} \neq x$.
Substituting this into the previous expression:

$$\begin{aligned}\mathbb{E}[c_{x\hat{x}}] &= \sum_{x,\hat{x}} c_{x\hat{x}} \cdot P(\hat{x} \mid x) \cdot P(x)\\ &= \sum_{x,\hat{x}} c_{x\hat{x}} \cdot \mathbb{1}_{\hat{x}=x} \cdot P(x)\\ &= \sum_{x} c_{xx} \cdot P(x)\\ &= \sum_{x} c_{xx} \cdot \theta_x\end{aligned}$$

The final expression shows that the expected loss is minimized when we choose the category $\hat{x}$ that minimizes the loss, which is the category with the highest probability. This is the same as Strategy 1 from the previous question, so the optimal prediction is given by $\hat{x} = \arg\max_x \theta_x$.

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.)

   The formula for additive smoothing with parameter $\frac{1}{2}$ would be:

$$\hat{p}(y) = \frac{N_y + \frac{1}{2}}{N + \frac{1}{2} \cdot k}$$

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.

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

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.

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.

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.

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)

   1. Output Layer (Softmax Layer): The softmax layer computes the output as $\hat{y}_i = g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$ for $i = 1, 2, \ldots, k$, where z_i is the input to the softmax function for each output class. The cross-entropy loss is given by $L(x, y) = -\sum_{i=1}^k y_i \log(\hat{y}_i)$.

   The gradient of the loss with respect to the input to the softmax function z_i is:

   $$\frac{\partial L}{\partial z_i} = \hat{y}_i - y_i$$

   The update for the weights of the output layer (W_2) can be computed as:

   $$\Delta W_2 = \frac{\partial L}{\partial z} \cdot \sigma(W_1x)^T$$

   2. Hidden Layer: The hidden layer uses the sigmoid activation function: $a = \sigma(W_1x)$. The gradient of the loss with respect to the output of the hidden layer a is given by:

   $$\frac{\partial L}{\partial a} = (W_2^T \frac{\partial L}{\partial z}) \circ \sigma'(W_1x)$$

   Here, $\circ$ denotes element-wise multiplication, and $\sigma'(z)$ is the derivative of the sigmoid function.

   The update for the weights of the hidden layer (W_1) is:

   $$\Delta W_1 = \frac{\partial L}{\partial a} \cdot x^T$$

   3. Input Layer: The input layer simply passes on the data, so the gradient of the loss with respect to the input x is:

   $$\frac{\partial L}{\partial x} = W_1^T \frac{\partial L}{\partial a}$$

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 Errors:

   Learning Curve:

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)

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)