HOMEWORK 4

Dario Placencio - 907 284 6018

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 $\theta = (\theta_1, \ldots, \theta_k)$ 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.

Give outcome as i^*, such that: $i^* = \arg\max_i \theta_i$.
Now, there are two possible scenarios:

1. $x = i^*$ which occurs with probability θ_{i^*}. In this case, the 0-1 loss is 0.
2. $x \neq i^*$ which occurs with probability $1 - \theta_{i^*}$. In this case, the 0-1 loss is 1.

Therefore, the expected 0-1 loss for Strategy 1 is:

$\mathbb{E}[\nVdash_{\hat{x}\neq x}] = 1 - \theta_{i^*}$

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)

The prediction $\hat{x}$ is also drawn from the multinomial distribution with parameter vector θ.

Since the randomness and the world's randomness are independent, the probability that $\hat{x} = x$ is the sum of the probabilities that both we and the world draw the same outcome:

$\mathbb{P}(\hat{x} = x) = \sum_{i=1}^k \theta_i^2$

The probability that $\hat{x} \neq x$ is:

$1 - \sum_{i=1}^k \theta_i^2$

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

$\mathbb{E}[\nVdash_{\hat{x}\neq x}] = 1 - \sum_{i=1}^k \theta_i^2$

2 Best Prediction Under Different Misclassification Losses (6 pts)

Like in the previous question, the world generates a single observation $x \sim \text{multinomial}(\theta)$. 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 $\hat{x}$.

Given the loss matrix c_{ij}, the goal is to minimize the expected loss:

$\mathbb{E}[c_{x\hat{x}}]$

To find the optimal prediction $\hat{x}$, let's consider the loss incurred by predicting each possible j and then choose the j that minimizes this expected loss.

For a fixed prediction $\hat{x} = j$, the expected loss is:

$\mathbb{E}[c_{xj}] = \sum_{i=1}^{k} \theta_i c_{ij}$

This is because the probability that the true observation is i is θ_i and the loss incurred in that case is c_{ij}.

To find the prediction j that minimizes this expected loss, let's compute $\mathbb{E}[c_{xj}]$ for each j and choose the j that gives the smallest value.

The optimal prediction $\hat{x}$ is then:

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

Thus, to minimize the expected loss, we should predict j that minimizes the weighted sum of the loss, where the weights are given by the probabilities θ_i.

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.)

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.

   Shuffling the characters in a document does not affect the Naive Bayes classifier's prediction when using a character-based model. This is because:

   1. Independence Assumption: The Naive Bayes model assumes that characters are conditionally independent given the language. This means the occurrence of one character does not depend on the occurrence of another, given the language.

   2. Representation: Documents are represented as bag-of-characters, which only considers character counts, not their order. Shuffling doesn't change these counts.

   Mathematically, the likelihood term is: $p(x \mid y) = \prod_{i=1}^{d} \theta_{i,y}^{x_i}$

   Where $\theta_{i,y}$ is the probability of character i given language y and x_i is the count of character i. Since only the counts x_i matter and they remain unchanged with shuffling, the prediction remains the same.

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)

   Given:

$$\hat{y} = f(x) = g(W_2 \sigma(W_1 x))$$

Where: $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function. $g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ is the softmax function. Loss function $L(x, y) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$.

Forward Pass:

1. Compute the input to the hidden layer:
$$a_1 = W_1 x$$
2. Compute the output of the hidden layer:
$$h = \sigma(a_1)$$
3. Compute the input to the output layer:
$$a_2 = W_2 h$$
4. Compute the output of the network:
$$\hat{y} = g(a_2)$$

Backward Pass:

1. Gradient of the loss with respect to the output:
$$\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}$$
2. Gradient of the loss with respect to a_2 (input of softmax):
$$\frac{\partial L}{\partial a_2} = \hat{y} - y$$
3. Gradient of the loss with respect to W_2:
$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial a_2} h^T$$
4. Gradient of the loss with respect to h:
$$\frac{\partial L}{\partial h} = W_2^T \frac{\partial L}{\partial a_2}$$
5. Gradient of the loss with respect to a_1 (input of sigmoid):
$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial h} \cdot h \cdot (1 - h)$$
6. Gradient of the loss with respect to W_1:
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a_1} x^T$$

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)

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)