HOMEWORK 4

Gary (Joe) Bales
908 457 3204

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.
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)
Under strategy 1:

$$\mathbb{E}[\mathbb{I}_{\hat{x} \neq x}] = \sum_{i=1}^{k} \mathbb{I}_{\hat{x} \neq x} \mathbb{P}(x = k) = 1 - \mathbb{P}(x = \arg\max_k \theta_k) = 1 - \max_k \theta_k$$

Under strategy 2:

$$\mathbb{E}[\mathbb{I}_{\hat{x} \neq x}] = \sum_{\hat{x}=1}^{k} \sum_{x=1}^{k} \mathbb{I}_{\hat{x} \neq x} \mathbb{P}(x) \mathbb{P}(\hat{x}) = (\sum_{\hat{x}=1}^{k} \sum_{x=1}^{k} \theta_x \theta_{\hat{x}}) - \sum_{\hat{x}=1}^{k} \sum_{x=1}^{k} \mathbb{I}_{\hat{x}=x} \theta_x \theta_{\hat{x}}$$

Which is the sum of all pairwise products of thetas subtracted by all the pairwise products of pairs with equal indices.

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 $\hat{x}$.
First we'll think of the loss as a function of x and $\hat{x}$, and write out the loss knowing that our prediction should be independent of the result (since we don't have any other data to base the prediction off of besides probabilities).

$$\mathbb{E}[c(x, \hat{x})] = \sum_{x=1}^{k} \sum_{\hat{x}=1}^{k} c(x, \hat{x}) \theta_x \mathbb{P}(\hat{x}) = \sum_{x=1}^{k} \theta_x (\sum_{\hat{x}=1}^{k} c(x, \hat{x}) \mathbb{P}(\hat{x}))$$

We want to minimize this loss with respect to $\hat{x}$. Let $\hat{x} \sim$ multinomial(ϕ). Then our loss becomes:

$$\mathbb{E}[c(x, \hat{x})] = \sum_{x=1}^{k} \theta_x (\sum_{\hat{x}=1}^{k} c(x, \hat{x}) \phi_{\hat{x}})$$

and we now just pick the combination of $\{\phi_i\}_{i=1}^k$ that minimizes the linear combination of it with each $c(x, j)$, which will depend on where the losses are incurred.

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!)
Quickly I'll put an image depicting how I read everything in via R for reference:

Initialize our dataframe

```
DF = cbind(data.frame(Language = c(rep("e",20),rep("j",20),rep("s",20))),
           data.frame(matrix(0,nrow = 60, ncol = 27)))
colnames(DF) = c("Language", " ", letters)
```

Function to go into one file and grab a vector of character counts with names

```
GrabCharCounts = function(lang, numb){
  #load txt file corresponding to language and number as vector of characters
  #leaving out \n character
  file_chars = (read_file(putDir(lang, numb)) %>% str_split(""))[[1]]
  file_chars_nonewline = file_chars[file_chars!="\n"]

  #this counts the number of each
  table_chars_nonewline = file_chars_nonewline %>% table()

  #this makes sure that it includes count of zero for ones that didn't show up
  emptytable = rep(0,27)
  names(emptytable) = c(" ", letters)
  emptytable[names(table_chars_nonewline)] = table_chars_nonewline
  return(emptytable)
}
```

Filling our initialized dataframe

```
for (j in 1:3){
  for (i in 1:20){
    DF[(j-1)*20 + i,-1] = GrabCharCounts(c("e","j","s")[j], i-1)
  }
}
```

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.)

   I began by first reading in each text file, and saving them into a 60 by 28 data matrix. This matrix consisted of a column for its language class, and a column for each of the 27 different characters, where the number of occurrences were recorded.

   Let $\phi_l = p(y = l)$ where $\sum_{l \in \{e,j,s\}} \phi_l = 1$

   Then as an example, here is our estimated prior probability for an English document, although the process is exactly the same for the other languages too.

$$\hat{p}(y = e) = \frac{(\sum_{i=1}^{n_{tr}} \mathbb{I}(y_i = e)) + \frac{1}{2}}{(\sum_{i=1}^{n_{tr}} 1) + \sum_{y \in \{e,j,s\}} \frac{1}{2}} = \frac{10 + \frac{1}{2}}{30 + \frac{3}{2}} = \frac{1}{3}$$

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.
Let $\theta_{i,l} = p(c_i|y = l)$ be the class-conditional probability of a given character being drawn. Also let $b_{i,l}$ be the number of i-th characters present in all documents of language l.

Then given that the document is in English, our estimated probability of any given character using smoothing parameter 1/2, where 27 is the number of different characters, is:

$$\hat{\theta}_{i,l} = \hat{p}(c_i|y = e) = \frac{b_{i,e} + \frac{1}{2}}{(\sum_j b_{j,e}) + \frac{27}{2}}$$

Here is some code of how I found each of these values given a language condition:

This function will return us the estimated character probability given a language:

```
giveClassCondProb = function(lang, k, df = train_DF){
  #Get the subset of observations under language condition
  subDF = df[df$Language==lang,]
  #Then I just stored the numerator and denominator from the previous equation
  #for our class-conditional probability estimate
  num = (subDF[,1+k]%>%sum()) + (1/2)
  denom = (subDF[,-1]%>%apply(MARGIN=2,sum)%>%sum()) + (27/2)
  return(num/denom)
}
```

Here we use it to report every character's English-conditional probability:

```
giveClassCondProbs = function(lang, df = train_DF){
  #initialize a table to fill
  econdprobs = rep(0,27)
  names(econdprobs) = c(" ",letters)
  #filling and returning it
  for (k in 1:27){
    econdprobs[k] = giveClassCondProb(lang, k, df)
  }
  econdprobs %>% return()
}
```

```
giveClassCondProbs("e")
```

```
                    a             b             c             d             e
0.1792499587 0.0601685115 0.0111349744 0.0215099950 0.0219725756 0.1053692384
           f             g             h             i             j             k
0.0189327606 0.0174789361 0.0472162564 0.0554105402 0.0014207831 0.0037336858
           l             m             n             o             p             q
0.0289773666 0.0205187510 0.0579216917 0.0644639022 0.0167520238 0.0005617049
           r             s             t             u             v             w
0.0538245498 0.0661820585 0.0801255576 0.0266644639 0.0092846522 0.0154964480
           x             y             z
0.0011564513 0.0138443747 0.0006277879
```

3. Print θ_j, θ_s and include in final report the class conditional probabilities for Japanese and Spanish.
I just reuse the function I defined previously but for Japanese and Spanish:
Here we just repeat the same thing but for Japanese:

```
giveClassCondProbs("j")
```

```
                        a             b             c             d             e
1.234495e-01 1.317656e-01 1.086691e-02 5.485866e-03 1.722632e-02 6.020476e-02
           f             g             h             i             j             k
3.878542e-03 1.401167e-02 3.176212e-02 9.703344e-02 2.341102e-03 5.740941e-02
           l             m             n             o             p             q
1.432615e-03 3.979874e-02 5.671058e-02 9.116321e-02 8.735455e-04 1.048255e-04
           r             s             t             u             v             w
4.280373e-02 4.217478e-02 5.699011e-02 7.061742e-02 2.445928e-04 1.974213e-02
           x             y             z
3.494182e-05 1.415144e-02 7.722143e-03
```

and Spanish:

```
giveClassCondProbs("s")
```

```
                        a             b             c             d             e
1.682649e-01 1.045605e-01 8.232864e-03 3.752582e-02 3.974592e-02 1.138109e-01
           f             g             h             i             j             k
8.602880e-03 7.184484e-03 4.532700e-03 4.985970e-02 6.629459e-03 2.775123e-04
           l             m             n             o             p             q
5.294317e-02 2.580864e-02 5.417656e-02 7.249237e-02 2.426691e-02 7.677839e-03
           r             s             t             u             v             w
5.929512e-02 6.577040e-02 3.561407e-02 3.370232e-02 5.889427e-03 9.250409e-05
           x             y             z
2.497610e-03 7.862847e-03 2.682618e-03
```

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.
Since I stored all 60 text files as rows in a data frame, I just went back and printed out the row corresponding to e10.txt

```
(DF[11,-1])
```

```
    a  b  c  d  e  f  g  h  i j k  l  m  n  o  p q  r  s  t  u  v
11 498 164 32 53 57 311 55 51 140 140 3 6 85 64 139 182 53 3 141 186 225 65 31
    w x  y z
11 47 4 38 2
```

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.

We already defined a function that will grab us estimated class-conditional probabilities of characters. We can use those to now estimate the class-conditional probability of an observation.

(To avoid underflow, as recommended by the problem's prompt, we'll do arithmetic in log probability land and then report the final answer in probability land, i.e. perform computations using $e^{\log(\hat{p}(x|y=l))} = e^{\sum_{i=1}^{d} x_i \log(\theta_{i,l})}$).

Code and results:

Here are some functions that return to us an estimate for conditional probability or conditional log-probability

```
findProbOfNewChars = function(test = DF[11,-1], train = train_DF, lang){
  test2 = test %>% as.numeric()
  logprobs = giveClassCondProbs(lang, df = train) %>% as.numeric() %>% log()
  return(exp((test2*logprobs)%>%sum()))
}
findLogProbOfNewChars = function(test = DF[11,-1], train = train_DF, lang){
  test2 = test %>% as.numeric()
  logprobs = giveClassCondProbs(lang, df = train) %>% as.numeric() %>% log()
  return(((test2*logprobs)%>%sum()))
}
```

Here are the values to report:

```
data.frame( English =
              c(findLogProbOfNewChars(lang = "e"),findProbOfNewChars(lang = "e")),
            Japanese =
              c(findLogProbOfNewChars(lang = "j"),findProbOfNewChars(lang = "j")),
            Spanish =
              c(findLogProbOfNewChars(lang = "s"),findProbOfNewChars(lang = "s")),
            row.names = c("logProb of X given language", "Prob of X given language") )
```

```
                            English  Japanese   Spanish
logProb of X given language -7841.865 -8771.433 -8467.282
Prob of X given language        0.000     0.000     0.000
```

I have reported the log probabilities because exponentiating such largely negative numbers in R results in exactly zero results.

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.
Due to underflow, I won't be able to show these values precisely. This is because in finding the denominator of Bayes' rule, $\hat{p}(x) = \sum_l \hat{p}(x|y = l)\hat{p}(y = l)$, there is no way for me to avoid getting something that evaluates to zero. I can still compare the numerators to get a prediction, but I won't be able to find the exact posterior probability.

   Notice that

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_t p(x|t)p(t)}$$

$$\ldots = \exp(-\log(\frac{\sum_t p(x|t)p(t)}{p(x|y)p(y)}))$$

$$\ldots = \exp(-\log(1 + \frac{\sum_{t \neq y} p(x|t)p(t)}{p(x|y)p(y)}))$$

   And note that for any t, because our class probability estimates are the same regardless of class,

$$\frac{p(x|t)p(t)}{p(x|y)p(y)} = \exp\{\log p(x|t) + \log p(t) - \log p(x|y) - \log p(y)\} = \exp\{\log p(x|t) - \log p(x|y)\}$$

   So we arrive at this expression for the class probability conditioned on our observed features:

$$p(y|x) = \exp(-\log(1 + \sum_{t \neq y} \exp\{\log p(x|t) - \log p(x|y)\}))$$

```
findDiffInLog = function(test = DF[11,-1], train = train_DF, lang1, lang2){
  return( findLogProbOfNewChars(test, train, lang1) - findLogProbOfNewChars(test, train, lang2) )
}

findA = function(test = DF[11,-1], train = train_DF, lang){
  vec = c("e","j","s")[!(c("e","j","s")%in%c(lang))]
  tosum = 1
  for (i in 1:2){
    tosum = tosum + exp(findDiffInLog(test, train, vec[i], lang))
  }
  return(tosum)
}

findPosterior = function(test = DF[11,-1], train = train_DF, lang){
  return( exp( -log( findA(test, train, lang) ) ) )
}
```

   Here are the reported posterior probability estimates:

```
data.frame( English = findPosterior(lang = "e"),
            Japanese = findPosterior(lang = "j"),
            Spanish = findPosterior(lang = "s"),
            Prediction = "English")
```

```
  English Japanese       Spanish Prediction
1       1        0 2.426739e-272    English
```

   The estimates for posterior probabilities of English and Japanese classes were close enough to 1 and 0 respectively that R simply outputs those values.

Though to determine the actual classification, it is enough to compare log class-conditional likelihoods because:

$$\arg\max_y \hat{p}(y|x) = \arg\max_y \frac{\hat{p}(x|y)\hat{p}(y)}{\hat{p}(x)} = \arg\max_y \{\log \hat{p}(x|y) + \log \hat{p}(y)\} = \arg\max_y \log \hat{p}(x|y)$$

and also since $\hat{p}(y = l) = \frac{1}{3}$ for any $l \in \{e, j, s\}$.

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.

   Here we'll define functions to give us our predictions for the test subset of observations.

```
chooseLanguage = function(test, train = train_DF){
  langs = c("e","j","s")
  vec = c(findLogProbOfNewChars(test, lang = "e"),
          findLogProbOfNewChars(test, lang = "j"),
          findLogProbOfNewChars(test, lang = "s"))
  return(langs[which.max(vec)])
}

gatherAllPredictions = function(){
  vec = rep(" ",30)
  for (t in 1:3){
    for (i in 1:10){
      vec[i + ((t-1)*10)] = chooseLanguage(test = DF[((20*(t-1))+i+10),-1])
    }
  }
  return(vec)
}

(caret::confusionMatrix(
  data = factor(gatherAllPredictions()),
  reference = factor(c(rep("e",10),rep("j",10),rep("s",10)))
  ))$table
```

```
          Reference
Prediction  e  j  s
         e 10  0  0
         j  0 10  0
         s  0  0 10
```

   Our model predicted everything it in the language it actually was in.

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.
   Conditioned on a class, our model assumes that each feature is independent of the other, meaning that regardless of what order we put the characters in, it will still only use the number of characters observed in the text for each of the 27 characters.

   The model is not intended to identify when someone feeds it absolute nonsense. Its only purpose is to try to distinguish between the language of the texts given by use of its character counts!

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)
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)