# HOMEWORK 4

>>VISHAL V NAIK<<
>>9084602235<<

**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 it 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 : https://github.com/vvnaik2/ece760.git

## 1 Best Prediction

### 1.1 Under 0-1 Loss (10 pts)

Suppose the world generates a single observation $x \sim$ 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 $\theta$ 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.

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

   Given the $\theta$ , and predicting x with the highest probability $\theta$ is like giving constant output to all the observations world generates. Therefore
   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}]$ is nothing but the chances where world observation is not the x with highest probability which is therefore
   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}] = 1 - \arg\max_x \theta_x$

2. Strategy 2: You mimic the world by generating a prediction $\hat{x} \sim$ multinomial$(\theta)$. (Hint: your randomness and the world's randomness are independent)

   In this case, Loss is nothing but both the multinomials give different outputs. This probability can be simplified to total minus the probability when they give same outcome.
   $\mathbb{E}[\mathbb{1}_{\hat{x} \neq x}]$
   $= P[\hat{x} \neq x]$
   $= 1 - P[\hat{x} = x]$
   $= 1 - \sum_{i=1}^{k} P_{\theta i}[\hat{x} = x]$
   $= 1 - \sum_{i=1}^{k} P_{\theta i}[\hat{x}] * P_{\theta i}[x]$
   $= 1 - \sum_{i=1}^{k} \theta_i^2$
   [P(X,Y)= P(x) * P(Y)]
   since X,Y are independent

### 1.2 Under Different Misclassification Losses (6 pts)

Like in the previous question, the world generates a single observation $x \sim$ 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 of 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 $\hat{x}$.

By trying to formulize the expectation with probabilities, we can try to find the optimal $\hat{x}$

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}P(x=i,\hat{x}=j) = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}P(x=i)*P(\hat{x}=j)$$

$$\mathbb{E}[c_{x\hat{x}}] = \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij}\theta_i*P(\hat{x}=j) = \sum_{j=1}^{k}\sum_{i=1,i\neq j}^{k} c_{ij}\theta_i*P(\hat{x}=j) = \sum_{j=1}^{k} P(\hat{x}=j)\sum_{i=1,i\neq j}^{k} c_{ij}\theta_i$$

To minimize the loss, we need to minimize the $\sum_{i=1,i\neq j}^{k} c_{ij}\theta_i$. Assuming to predict only one value , that means prob = 1 which makes $\hat{x} = argmin_j \sum_{i=1,i\neq j}^{k} c_{ij}\theta_i$

## 2 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 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 the 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!)
In the following questions, you may use the additive smoothing technique to smooth categorical data, in case the estimated probability is zero. Given $N$ data samples $\{\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, where $\boldsymbol{x}^{(i)} = [x_1^{(i)}, \ldots, x_j^{(i)}, \ldots, x_{M_i}^{(i)}]$ is a bag of characters, $M_i$ is the total number of characters in $\boldsymbol{x}^{(i)}$, $x_j^{(i)} \in S, y^{(i)} \in L$ and we have $|S| = K_S, |L| = K_L$. Here $S$ is the set of all character types, and $L$ is the set of all classes of data labels. Then by the additive smoothing with parameter $\alpha$, we can estimate the conditional probability as

$$P_\alpha(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + \alpha}{(\sum_{b_s \in S}\sum_{i=1}^{N}\sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + K_S\alpha},$$

where $a_s \in S, c_k \in L$. Similarly, we can estimate the prior probability

$$P_\alpha(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + \alpha}{N + K_L\alpha},$$

where $c_k \in L$ and $N$ is the number of training samples.

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 the prior probabilities.

$$P_{0.5}(Y = c_k) = \frac{(\sum_{i=1}^{N} \mathbb{1}[y^{(i)} = c_k]) + 0.5}{N + 3*0.5},$$

```
S = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',' ']
L = ['e','s','j']
Ks = len(S)
Kl = len(L)
alpha = 0.5
f_array = []
print("size of S : {} size of L : {}".format(Ks,Kl))
directory = "."


num_of_e_file = len(glob.glob(directory+'/e?.txt')) + alpha
num_of_s_file = len(glob.glob(directory+'/s?.txt')) + alpha
num_of_j_file = len(glob.glob(directory+'/j?.txt')) + alpha
total_num_of_files = num_of_e_file + num_of_s_file + num_of_j_file - 3*alpha + Kl*alpha

prob_prior_e_file = num_of_e_file/total_num_of_files
prob_prior_s_file = num_of_s_file/total_num_of_files
prob_prior_j_file = num_of_j_file/total_num_of_files

print ("Prior Probability of E files : {}".format(prob_prior_e_file))
print ("Prior Probability of S files : {}".format(prob_prior_s_file))
print ("Prior Probability of J files : {}".format(prob_prior_j_file))
```

```
size of S : 27 size of L : 3
    Prior Probability of E files : 0.3333333333333333
    Prior Probability of S files : 0.3333333333333333
    Prior Probability of J files : 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 $\theta_e$ which is a vector with 27 elements.

$$P_{0.5}(a_s \mid y = c_k) = \frac{(\sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = a_s, y^{(i)} = c_k]) + 0.5}{(\sum_{b_s \in S} \sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbb{1}[x_j^{(i)} = b_s, y^{(i)} = c_k]) + 27 * 0.5},$$

```
def calc_cond_prob(f,num_of_files):
    print("\n Calculating conditional probabilities for characters in {} files".format(str(f)))
    dic = 'dic' + str(f)
    dic = {}
    for x in range(num_of_files):
      fname = str(f) + str(x) + '.txt'
      print(fname)
      file = open(str(fname),'r')
      f_data = file.read()
      for char in S:
        if dic.get(char) != None:
          dic[char] =  dic[char] + f_data.count(char)
        else :
          dic[char] = f_data.count(char)
      file.close()
    print("Count of each char in {} files".format(str(f)))
    print(dic)
    dic_sum = sum(dic.values())
    cond_prob = dic
    for char in S:
      cond_prob[char] = (cond_prob[char] + alpha) / (dic_sum + Ks*alpha)
    print("Conditional probability of each char in {} files".format(str(f)))
    print(cond_prob)
    return cond_prob

file_index = []
cond_prob_e = calc_cond_prob('e',10)
cond_prob_s = calc_cond_prob('s',10)
cond_prob_j = calc_cond_prob('j',10)
```

```
Count of each char in e files
{'a': 910, 'b': 168, 'c': 325, 'd': 332, 'e': 1594, 'f': 286, 'g': 264, 'h': 714, 'i': 838, 'j': 21, 'k': 56, 'l': 438, 'm': 310, 'n': 876, 'o': 975, 'p':
Class conditional probability of e
0.0601685114819098
0.011134974392863043
0.021509995043779945
0.021972575582355856
0.1053692383941847
0.018932760614571286
0.017478936064761277
0.047216256401784236
0.055410540227986124
0.001420783082768875
0.0037336857756484387
0.028977366595076822
0.020518751032545846
0.057921691723112505
0.06446390219725756
0.01675202378985627
0.0005617049396993227
0.053824549810011564
0.06618205848339666
0.08012555757475633
0.026664463902197257
0.009284652238559392
0.015496448042293078
0.001156451346439782
0.013844374690236246
0.0006277878737815959
0.1792499586981662
```

3. Print $\theta_j, \theta_s$, the class conditional probabilities for Japanese and Spanish.

```
Count of each char in j files
{'a': 1885, 'b': 155, 'c': 78, 'd': 246, 'e': 861, 'f': 55, 'g': 200, 'h': 454, 'i': 1388, 'j': 33, 'k': 821, 'l': 20, 'm': 569, 'n': 811, 'o': 1304, 'p':
Class conditional probability of j
0.13176561025891 89
0.010866906600510151
0.005485866033054963
0.017226318180229 92
0.06020475907613823
0.0038785422271917 26
0.014011670568503443
0.03176211607673224
0.09703343932352633
0.0023411020650616725
0.05740941332681086
0.001432614696530277
0.03979873510604843
0.056710576889479 02
0.09116321324993885
0.0008735455466648031
0.00010482546559977637
0.042803731786575 35
0.042174778992976 7
0.056990111464411755
0.070617421992382 69
0.0002445927530661449
0.019742129354624 55
3.4941821866592126e-05
0.014151437855969 81
0.007722142632516 86
0.12344945665466997
```

```
Count of each char in s files
{'a': 1695, 'b': 133, 'c': 608, 'd': 644, 'e': 1845, 'f': 139, 'g': 116, 'h': 73, 'i': 808, 'j': 107, 'k': 4, 'l': 858, 'm': 418, 'n': 878, 'o': 1175, 'p':
Class conditional probability of s
0.10456045141993771
0.008232863618143134
0.037525824057229 19
0.039745922111559924
0.1138108599796491
0.00860287996053159
0.007184483981375 8445
0.004532700194258 5795
0.049859702136844375
0.006629459467793161
0.0002775122567913416
0.052943171656748174
0.025808639881594 77
0.054176559464709693
0.072492368412938 24
0.024266905121642 87
0.007677839104560451
0.059295111886774999
0.065770404859547 97
0.035614072954888 84
0.033702321852548 49
0.005894267830162 5
9.250408559711388e-05
0.002497610311122 0747
0.007862847275754679
0.002682618482316 3022
0.16826493170115014
```

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 $x$.

```
#Qusetion 2 Part 4

dic_e10 = {}
file = open('e10.txt','r')
f_data = file.read()
print("Count of each char in e10 file")
for char in S:
  dic_e10[char] = f_data.count(char)
  print("char : {}  count  : {}".format(char,dic_e10[char]))
file.close()
```

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

5. For the $x$ of e10.txt, 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. $y$. Also, Store all probabilities here and below in $\log()$ internally to avoid underflow. This also means you need to do arithmetic in log space.

```
# Question 2 Part 5

def calc_p_cap(dic_test,cond_prob_dict):
  p_cap = 0
  for char in S:
    p_cap = p_cap + (dic_test[char]*np.log(cond_prob_dict[char]))
  print("p_cap : {}".format(p_cap))
  return p_cap
print("Printing p_cap for e, s and j respectively")
p_cap_e = calc_p_cap(dic_e10,cond_prob_e)
p_cap_s = calc_p_cap(dic_e10,cond_prob_s)
p_cap_j = calc_p_cap(dic_e10,cond_prob_j)
```

```
Printing p_cap for e, s and j respectively
p_cap : -7841.865447060635
p_cap : -8467.282044010557
p_cap : -8771.433079075032
```

6. For the $x$ of e10.txt, use the 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 $x$.

$\log_e(\hat{p}(y = e \mid x)) = -7842.975477$
$\log_e(\hat{p}(y = s \mid x)) = -8468.392044$
$\log_e(\hat{p}(y = j \mid x)) = -8772.54309$
Predicted class = $e$

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.

| | English | Spanish | Japanese |
|---|---|---|---|
| English | | | |
| Spanish | | | |
| Japanese | | | |

```
# Question 2 part 7

c_mat = []
c_mat = np.array([0]*9).reshape(3,3)
i = 0
for f in L:
  for x in range(10):
    fname = str(f) + str(x+10) + '.txt'
    print(fname)
    file = open(str(fname),'r')
    f_data = file.read()
    dic = str(f) + str(x+10)
    dic = {}
    for char in S:
      dic[char] = f_data.count(char)
    file.close()
    p_cap_e = calc_p_cap(dic,cond_prob_e)
    p_cap_s = calc_p_cap(dic,cond_prob_s)
    p_cap_j = calc_p_cap(dic,cond_prob_j)
    pred = np.argmax([p_cap_e,p_cap_s,p_cap_j])
    c_mat[i][pred] = c_mat[i][pred] + 1
  i = i + 1
print("Printing the confusion matrix (Format as specified in the question) :")
print(c_mat)
```

```
Printing the confusion matrix (Format as specified in the question) :
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

8. Take a test document. 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 of the file does not change the predictions of the classifier.
Naive bayes uses the count of each character and not the ordering of characters.
In mathematical terms it assumes conditional independence of features
$P(X1, X2, .....Xn, Y) = P(X1, X2, .....Xn|Y)P(Y)$
$= (\prod_{k=1}^{n} P(Xk|Y))P(Y)$

```
# Question 2 part 8

dic_e12 = {}
dic_e12_new = {}
file = open('e12.txt','r')
f_data = file.read()
print("Contents of e12.txt before shuffling")
print(f_data)
new_f_data = random.sample(f_data,len(f_data))
print("Contents of e12.txt after shuffling")
print(new_f_data)
for char in S:
  dic_e12[char] = f_data.count(char)
  dic_e12_new[char] = new_f_data.count(char)
file.close()
print("\n Count of each char in e12 file")
print(dic_e12)
print("\n Count of each char in e12 file after shuffling")
print(dic_e12_new)
print("Printing p_cap of e12 for e, s and j respectively")
p_cap_e = calc_p_cap(dic_e12,cond_prob_e)
p_cap_s = calc_p_cap(dic_e12,cond_prob_s)
p_cap_j = calc_p_cap(dic_e12,cond_prob_j)
print("Printing p_cap of shuffled e12 for e, s and j respectively")
p_cap_e = calc_p_cap(dic_e12_new,cond_prob_e)
p_cap_s = calc_p_cap(dic_e12_new,cond_prob_s)
p_cap_j = calc_p_cap(dic_e12_new,cond_prob_j)
```

```
 Count of each char in e12 file
{'a': 122, 'b': 23, 'c': 59, 'd': 36, 'e': 196, 'f': 35, 'g': 15, 'h': 89, 'i': 103, 'j': 0, 'k': 8, 'l': 48, 'm': 38, 'n': 106, 'o': 111, 'p': 25, 'q': 1,

 Count of each char in e12 file after shuffling
{'a': 122, 'b': 23, 'c': 59, 'd': 36, 'e': 196, 'f': 35, 'g': 15, 'h': 89, 'i': 103, 'j': 0, 'k': 8, 'l': 48, 'm': 38, 'n': 106, 'o': 111, 'p': 25, 'q': 1,
Printing p_cap of e12 for e, s and j respectively
p_cap : -5285.490455003693
p_cap : -5681.093710850029
p_cap : -5836.3627981594045
Printing p_cap of shuffled e12 for e, s and j respectively
p_cap : -5285.490455003693
p_cap : -5681.093710850029
p_cap : -5836.3627981594045
```

# 3 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.

$$\hat{y} = f(x) = g(W_3\sigma(W_2\sigma(W_1x)))$$

Suppose $x \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{d_1 \times d}$, $W_2 \in \mathbb{R}^{d_2 \times d_1}$, $W_3 \in \mathbb{R}^{k \times d_2}$ 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)

Let $x$ and $Y$ be the inputs and the true label for the Neural Network

$$W_1X = Z_1 \quad A_1 = \sigma(Z_1)$$
$$W_2A_1 = Z_2 \quad A_2 = \sigma(Z_2)$$
$$W_3A_2 = Z_3 \quad \hat{Y} = g(Z_3)$$

$$L'_{W_3} = \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial W_3}$$
$$L'_{W_3} = \frac{\partial L}{\partial W_3} = (\hat{Y} - Y) * A_2^T \quad (1)$$

$$
\begin{aligned}
L'_{W_2} &= \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial A_2} * \frac{\partial A_2}{\partial Z_2} * \frac{\partial Z_2}{\partial W_2} \\
L'_{W_2} &= \frac{\partial L}{\partial W_2} = diag(A2') * W_3^T * (\hat{Y} - Y) * A_1^T \\
&= diag(A2 * (1 - A2)) * W_3^T * (\hat{Y} - Y) * A_1^T
\end{aligned} \tag{2}
$$

$$
\begin{aligned}
L'_{W_1} &= \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{Y}} * \frac{\partial \hat{Y}}{\partial Z_3} * \frac{\partial Z_3}{\partial A_2} * \frac{\partial A_2}{\partial Z_2} * \frac{\partial Z_2}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} \\
L'_{W_1} &= \frac{\partial L}{\partial W_1} = diag(A1') * W_2^T * diag(A2') * W_3^T * (\hat{Y} - Y) * X^T \\
&= diag(A1 * (1 - A1)) * W_2^T * diag(A2 * (1 - A2)) * W_3^T * (\hat{Y} - Y) * X^T
\end{aligned} \tag{3}
$$

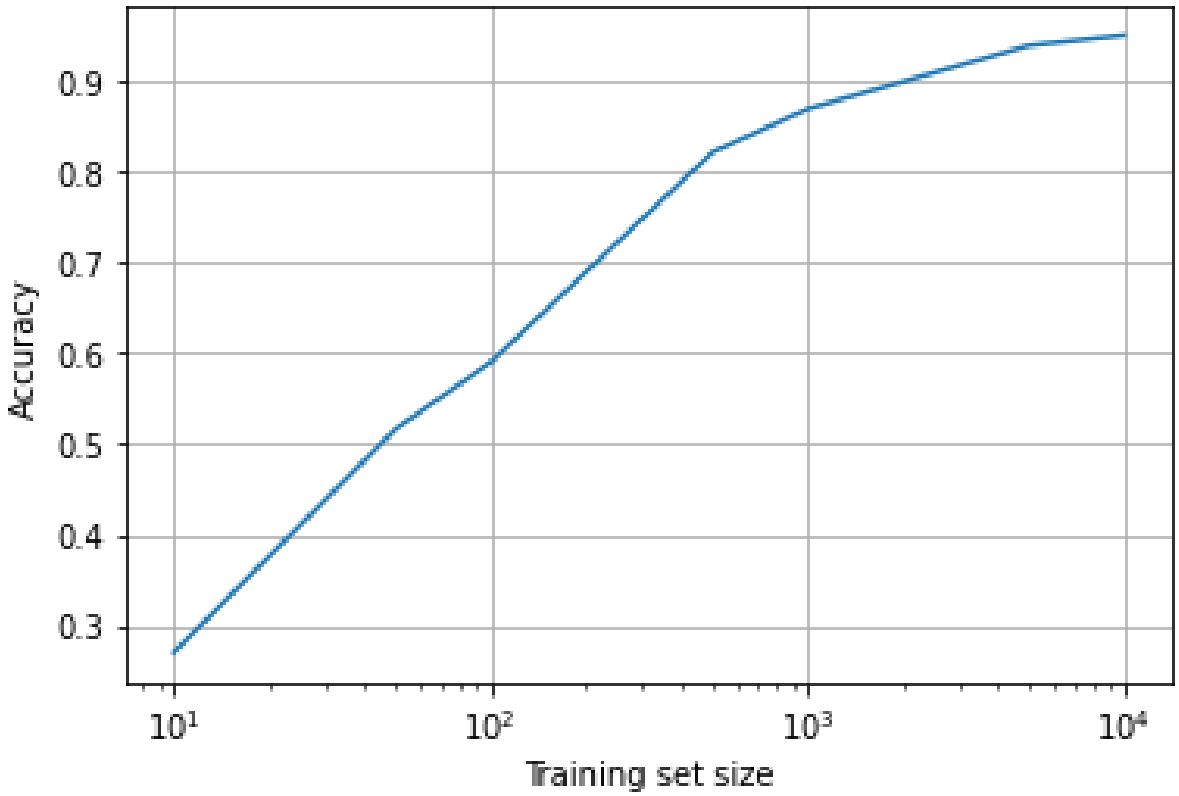
$$
\begin{aligned}
W_1(t+1) &= W_1(t) - \alpha * L'_{W_1} \\
W2(t+1) &= W_2(t) - \alpha * L'_{W_2} \\
W_3(t+1) &= W_3(t) - \alpha * L'_{W_3}
\end{aligned}
$$

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)

   Below run is with a batch size of 1. Other parameters are mentioned in the screenshots.

```
Training set size : 10 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 2708
Number of errors (wrong predictions) : 7292
Accuracy : 0.2708

Training set size : 50 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 5162
Number of errors (wrong predictions) : 4838
Accuracy : 0.5162

Training set size : 100 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 5900
Number of errors (wrong predictions) : 4100
Accuracy : 0.59
```

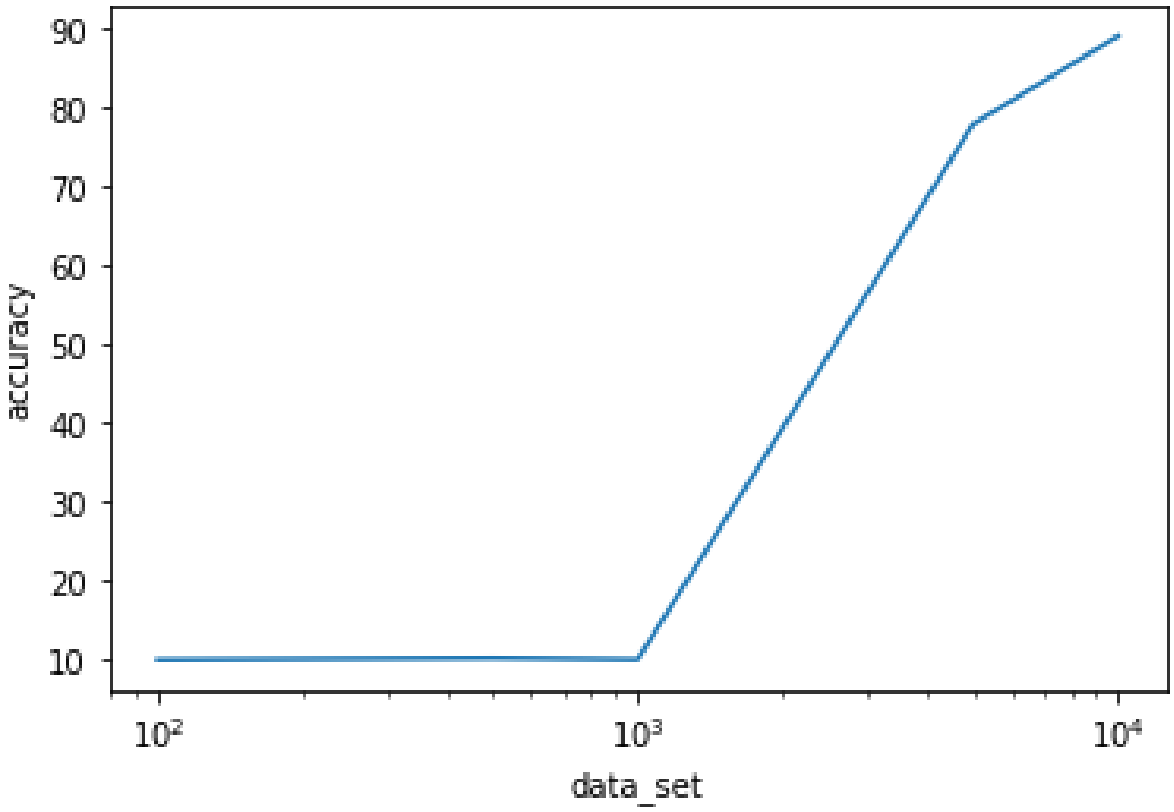
```
Training set size : 500 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 8207
Number of errors (wrong predictions) : 1793
Accuracy : 0.8207

Training set size : 1000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 8688
Number of errors (wrong predictions) : 1312
Accuracy : 0.8688

Training set size : 5000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 9385
Number of errors (wrong predictions) : 615
Accuracy : 0.9385

Training set size : 10000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 9498
Number of errors (wrong predictions) : 502
Accuracy : 0.9498
```




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)

| data set | test error |
|---|---|
| 100 | 9088 |
| 500 | 8765 |
| 1000 | 8023 |
| 5000 | 2111 |
| 10000 | 1043 |

learning rate = 0.003
number of epochs = 20
batch size = 64

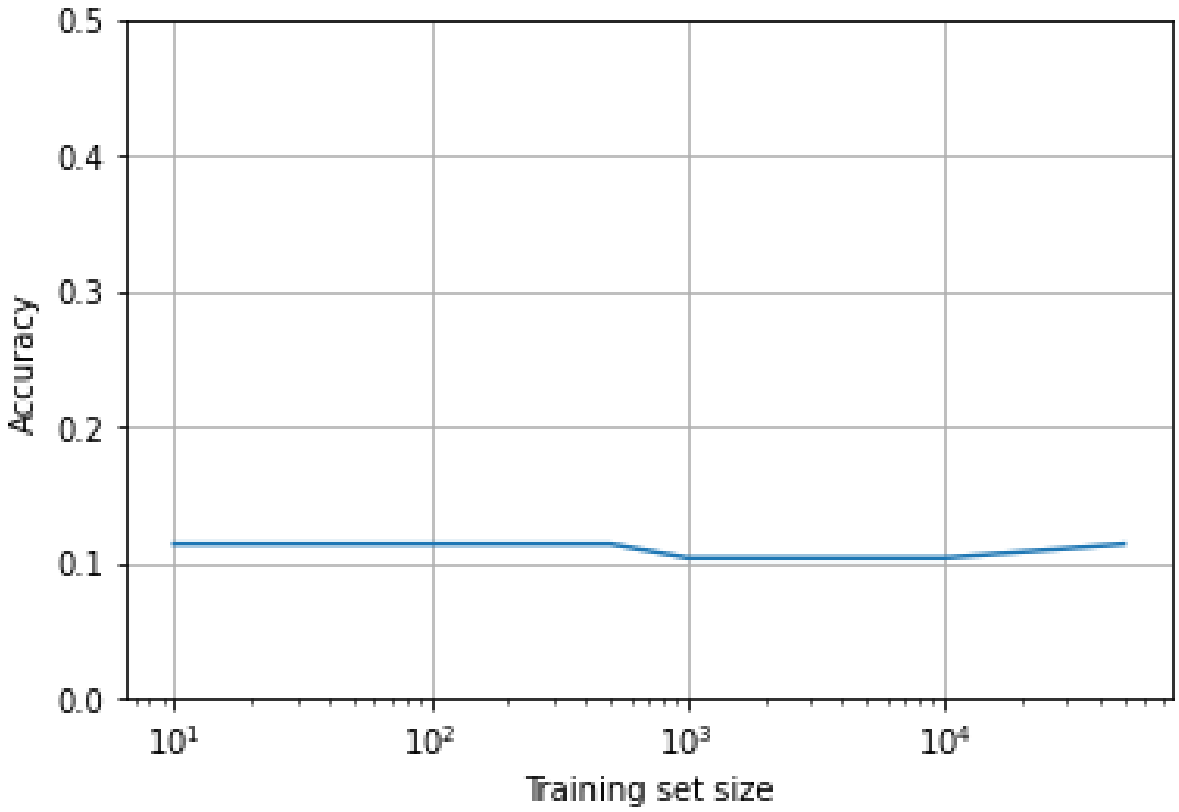



4. Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)

   The above run (Question 2) is with weights initialized between -1 and 1. The same implementation is run with initializing the weights to 0. Results shown below:

```
Training set size : 10 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1135
Number of errors (wrong predictions) : 8865
Accuracy : 0.1135

Training set size : 50 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1135
Number of errors (wrong predictions) : 8865
Accuracy : 0.1135

Training set size : 100 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1135
Number of errors (wrong predictions) : 8865
Accuracy : 0.1135
```

```
Training set size : 500 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1135
Number of errors (wrong predictions) : 8865
Accuracy : 0.1135

Training set size : 1000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1028
Number of errors (wrong predictions) : 8972
Accuracy : 0.1028

Training set size : 5000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1028
Number of errors (wrong predictions) : 8972
Accuracy : 0.1028

Training set size : 10000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1028
Number of errors (wrong predictions) : 8972
Accuracy : 0.1028

Training set size : 50000 , Learning rate : 0.1
Total number of predictions (test set size) : 10000
Number of correct predictions : 1135
Number of errors (wrong predictions) : 8865
Accuracy : 0.1135

[0.1135, 0.1135, 0.1135, 0.1135, 0.1028, 0.1028, 0.1028, 0.1135]
```




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 $d_1 = 300$, $d_2 = 200$, $d_3 = 100$. 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)