HOMEWORK 4

Smit Shah
ID: 908 587 6440
https://github.com/smit-1999/NaiveBayes

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 this strategy, the prediction $\hat{x}$ is the outcome with the highest probability according to the parameter vector i.e. $\hat{x} \in \arg\max_x \theta_x$. Formula for expectation is given by $\sum_{x=1}^k x * p(x)$
$\therefore$ Expected 0-1 loss will be:

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

=

$$E[1(\hat{x} \neq x)] = \sum_{i=1}^{k} P(\hat{x} \neq x | x = i) \cdot P(x = i)$$

Since $\hat{x}$ is chosen as the outcome with the highest probability, $P(\hat{x} \neq x | x = i) = 1 - \theta_{max}$.
Therefore,

$$E[1(\hat{x} \neq x)] = \sum_{i=1}^{k} (1 - \theta_{max}) \cdot \theta_{max} = \sum_{i=1}^{k} \theta_{max} - \sum_{i=1}^{k} \theta_{max}^2$$

$$= \theta_{max} * (1 - \theta_{max})$$

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 this strategy, the prediction $\hat{x}$ is generated from the multinomial distribution with the parameter vector θ. The randomness in this prediction and the randomness in the world's observation (x) are independent.
The expected 0-1 loss for this strategy is the probability that the predicted value $\hat{x}$ is not equal to the true value x:

$$E[1(\hat{x} \neq x)] = P(\hat{x} \neq x) = 1 - P(\hat{x} = x)$$

In a multinomial distribution, the probability mass function is given by:

$$P(\hat{x} = i) = \theta_i$$

Therefore,

$$E[1(\hat{x} \neq x)] = 1 - P(\hat{x} = x) = 1 - \theta_x$$

So, the expected 0-1 loss for Strategy 2 is $1 - \theta_x$.

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 $\hat{x}$.

To derive the optimal prediction $\hat{x}$ that minimizes the expected loss $E[c_{x\hat{x}}]$, we need to find the value of $\hat{x}$ that minimizes the expected loss over all possible values of $\hat{x}$. Mathematically, we want to minimize the following expression:

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

Given that x follows a multinomial distribution with parameter vector θ, the joint probability $P(x = i, \hat{x} = j)$ can be expressed as $P(x = i) \cdot P(\hat{x} = j|x = i)$. Since $\hat{x}$ is your prediction, we are interested in minimizing $E[c_{x\hat{x}}]$ with respect to $\hat{x}$. Therefore, the problem can be formulated as follows:

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

To find the optimal $\hat{x}$, we minimize $E[c_{x\hat{x}}]$ with respect to $\hat{x}$ by setting the derivative with respect to $\hat{x}$ to zero. This gives us the following expression:

$$\frac{d}{d\hat{x}}E[c_{x\hat{x}}] = \sum_{i=1}^{k} c_{i\hat{x}} \cdot P(x = i) - \sum_{i=1}^{k}\sum_{j=1}^{k} c_{ij} \cdot P(x = i) \cdot P(\hat{x} = j|x = i) = 0$$

Solving this equation for $\hat{x}$ gives the optimal prediction that minimizes the expected loss $E[c_{x\hat{x}}]$. Please note that the solution for $\hat{x}$ might depend on the specific form of the cost matrix c_{ij} and the conditional probabilities $P(\hat{x} = j|x = i)$ which are derived from the multinomial distribution with parameter vector θ. The solution will vary based on the values in the cost matrix and the multinomial probabilities.

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.)

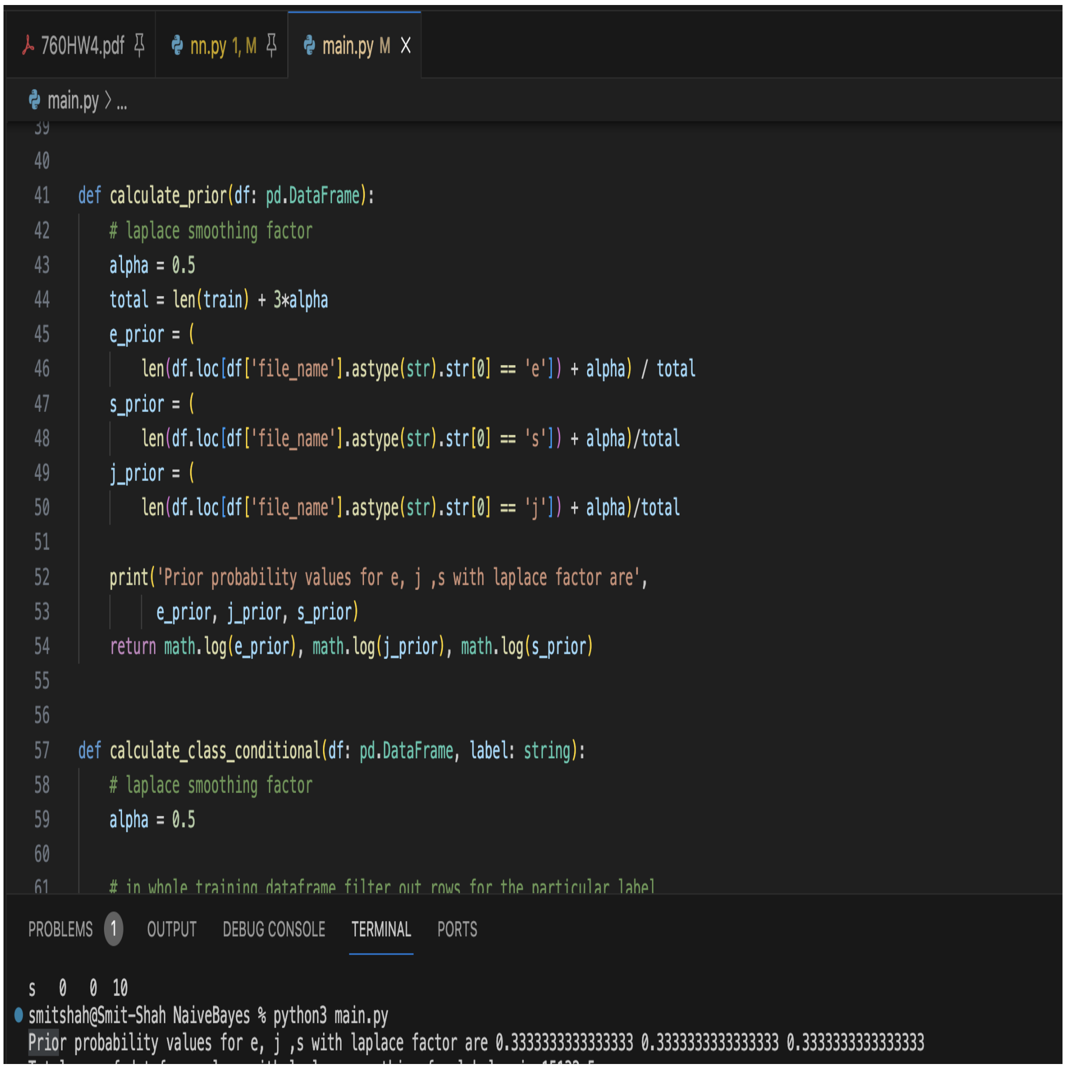
We will be using a character-based multinomial Naïve Bayes model. You need to view each document as a bag of characters, including space. We have made sure that there are only 27 different types of printable characters (a to z, and space) – there may be additional control characters such as new-line, 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.)

   $\hat{p}(y = e)$ =

   $$(b_j + \alpha)/((\sum_{j=1}^{k} b_j) + \alpha * k)$$

   Here $\alpha = 0.5$ since it is given in question and $k = 3$ since y has 3 possible values. The same can be extrapolated for other priors. The answer we get for $\hat{p}(y = e)$ = 10.5/31.5 = 0.333, and it will be same for other priors.



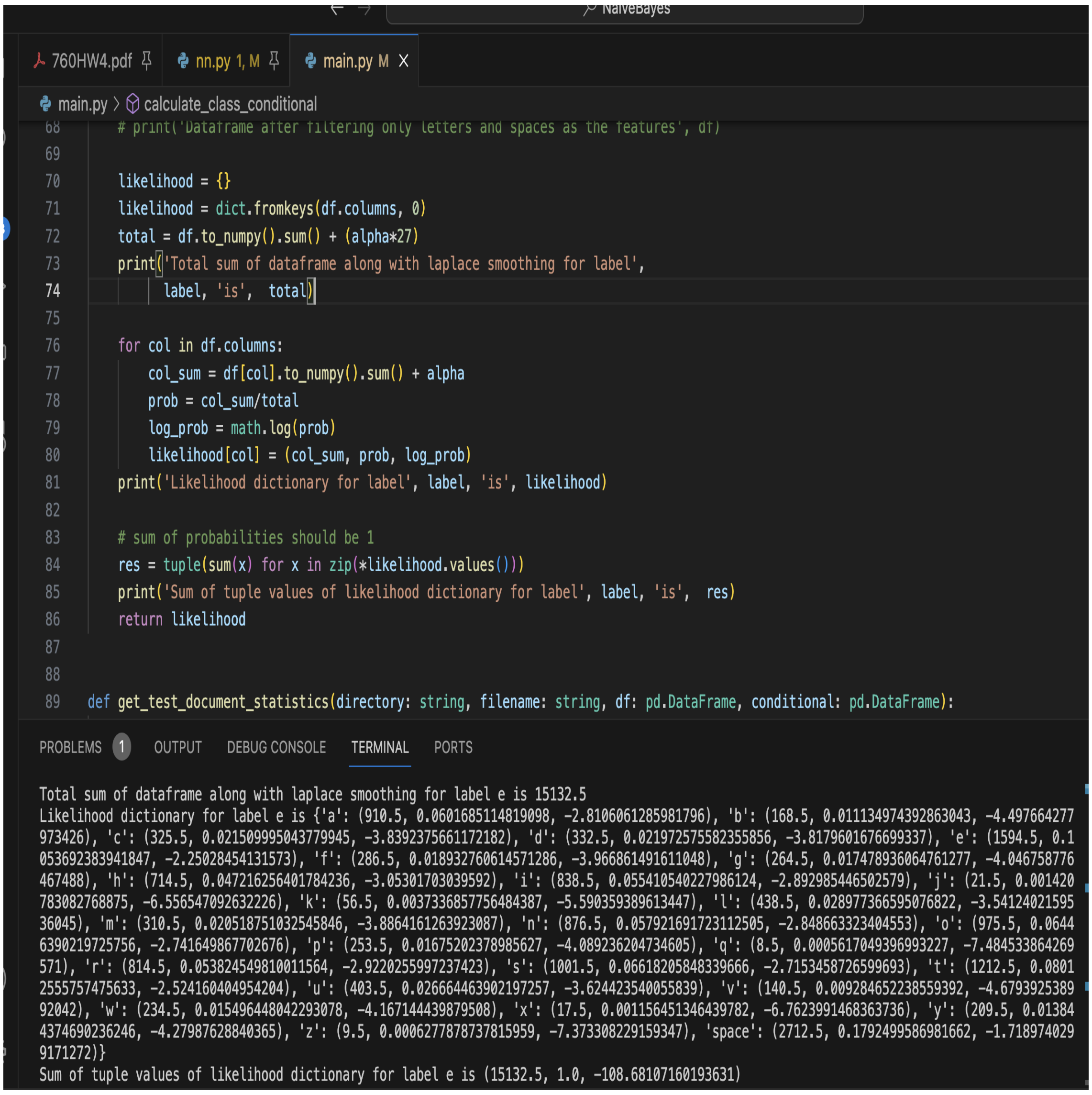
```python

def calculate_prior(df: pd.DataFrame):
    # laplace smoothing factor
    alpha = 0.5
    total = len(train) + 3*alpha
    e_prior = (
        len(df.loc[df['file_name'].astype(str).str[0] == 'e']) + alpha) / total
    s_prior = (
        len(df.loc[df['file_name'].astype(str).str[0] == 's']) + alpha)/total
    j_prior = (
        len(df.loc[df['file_name'].astype(str).str[0] == 'j']) + alpha)/total

    print('Prior probability values for e, j ,s with laplace factor are',
          e_prior, j_prior, s_prior)
    return math.log(e_prior), math.log(j_prior), math.log(s_prior)


def calculate_class_conditional(df: pd.DataFrame, label: string):
    # laplace smoothing factor
    alpha = 0.5

    # in whole training dataframe filter out rows for the particular label
```

```
s   0   0  10
smitshah@Smit-Shah NaiveBayes % python3 main.py
Prior probability values for e, j ,s with laplace factor are 0.3333333333333333 0.3333333333333333 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.



```python
    # print('Dataframe after filtering only letters and spaces as the features', df)

    likelihood = {}
    likelihood = dict.fromkeys(df.columns, 0)
    total = df.to_numpy().sum() + (alpha*27)
    print('Total sum of dataframe along with laplace smoothing for label',
          label, 'is',  total)

    for col in df.columns:
        col_sum = df[col].to_numpy().sum() + alpha
        prob = col_sum/total
        log_prob = math.log(prob)
        likelihood[col] = (col_sum, prob, log_prob)
    print('Likelihood dictionary for label', label, 'is', likelihood)

    # sum of probabilities should be 1
    res = tuple(sum(x) for x in zip(*likelihood.values()))
    print('Sum of tuple values of likelihood dictionary for label', label, 'is',  res)
    return likelihood


def get_test_document_statistics(directory: string, filename: string, df: pd.DataFrame, conditional: pd.DataFrame):
```

```
Total sum of dataframe along with laplace smoothing for label e is 15132.5
Likelihood dictionary for label e is {'a': (910.5, 0.0601685114819098, -2.8106061285981796), 'b': (168.5, 0.011134974392863043, -4.497664277
973426), 'c': (325.5, 0.021509995043779945, -3.8392375661172182), 'd': (332.5, 0.021972575582355856, -3.8179601676699337), 'e': (1594.5, 0.1
053692383941847, -2.25028454131573), 'f': (286.5, 0.018932760614571286, -3.966861491611048), 'g': (264.5, 0.017478936064761277, -4.046758776
467488), 'h': (714.5, 0.047216256401784236, -3.05301703039592), 'i': (838.5, 0.055410540227986124, -2.892985446502579), 'j': (21.5, 0.001420
783082768875, -6.556547092632226), 'k': (56.5, 0.0037336857756484387, -5.590359389613447), 'l': (438.5, 0.028977366595076822, -3.54124021595
36045), 'm': (310.5, 0.020518751032545846, -3.8864161263923087), 'n': (876.5, 0.057921691723112505, -2.848663323404553), 'o': (975.5, 0.0644
6390219725756, -2.741649867702676), 'p': (253.5, 0.01675202378985627, -4.089236204734605), 'q': (8.5, 0.0005617049396993227, -7.484533864269
571), 'r': (814.5, 0.053824549810011564, -2.9220255997237423), 's': (1001.5, 0.06618205848339666, -2.7153458726599693), 't': (1212.5, 0.0801
2555757475633, -2.524160404954204), 'u': (403.5, 0.026664463902197257, -3.624423540055839), 'v': (140.5, 0.009284652238559392, -4.6793925389
92042), 'w': (234.5, 0.015496448042293078, -4.167144439879508), 'x': (17.5, 0.001156451346439782, -6.7623991468363736), 'y': (209.5, 0.01384
4374690236246, -4.27987628840365), 'z': (9.5, 0.0006277878737815959, -7.373308229159347), 'space': (2712.5, 0.1792499586981662, -1.718974029
9171272)}
Sum of tuple values of likelihood dictionary for label e is (15132.5, 1.0, -108.68107160193631)
```

The formula for the calculation of likelihood is in the image where we first compute the sum of frequencies for a character and add $\alpha = 0.5$ and then divide it by sum of the counts across all letters and space and add $27 * \alpha$.
prob =$(columnSum + \alpha)/(sumofallcolumns + (numberofcolumns) * (\alpha))$
The likelihood vector for e looks something like this: ['a': 0.0601685114819098, 'b': 0.011134974392863043, 'c': 0.021509995043779945, 'd': 0.021972575582355856, 'e': 0.1053692383941847, 'f': 0.018932760614571286, 'g': 0.017478936064761277, 'h': 0.047216256401784236, 'i': 0.055410540227986124, 'j': 0.001420783082768875, 'k': 0.0037336857756484387, 'l': 0.028977366595076822, 'm': 0.020518751032545846, 'n': 0.057921691723112505, 'o': 0.06446390219725756, 'p': 0.01675202378985627, 'q': 0.0005617049396993227, 'r': 0.053824549810011564, 's': 0.06618205848339666, 't': 0.08012555757475633, 'u': 0.026664463902197257, 'v': 0.009284652238559392, 'w': 0.015496448042293078, 'x': 0.001156451346439782, 'y': 0.013844374690236246, 'z': 0.0006277878737815959, 'space': 0.1792499586981662]

3. Print θ_j, θ_s and include in final report the class conditional probabilities for Japanese and Spanish. For Japanese, we get the following values ['a': 0.1317656102589189, 'b': 0.010866906600510151, 'c': 0.005485866033054963, 'd': 0.01722631818022992, 'e': 0.06020475907613823, 'f': 0.003878542227191726, 'g': 0.014011670568503443, 'h': 0.03176211607673224, 'i': 0.09703343932352633, 'j': 0.002341102065061 6725, 'k': 0.05740941332681086, 'l': 0.001432614696530277, 'm': 0.03979873510604843, 'n': 0.05671057688947902, 'o': 0.09116321324993885, 'p': 0.0008735455466648031, 'q': 0.00010482546559977637, 'r': 0.04280373178657535, 's': 0.0421747789929767, 't': 0.056990111464411755, 'u': 0.07061742199238269, 'v': 0.0002445927530661449, 'w': 0.01974212935462455, 'x': 3.4941821866592126e-05, 'y': 0.01415143785596981, 'z': 0.00772214263251686, 'space': 0.12344945665466997]

For Spanish we get the following values, ['a': 0.10456045141993771, 'b': 0.008232863618143134, 'c': 0.03752582405722919, 'd': 0.039745922111559924, 'e': 0.1138108599796491, 'f': 0.00860287996053159, 'g': 0.0071844839813758445, 'h': 0.0045327001942585795, 'i': 0.049859702136844375, 'j': 0.006629459467793161, 'k': 0.0002775122567913416, 'l': 0.052943171656748174, 'm': 0.02580863988159477, 'n': 0.054176559464709693, 'o': 0.07249236841293824, 'p': 0.02426690512164287, 'q': 0.007677839104560451, 'r': 0.05929511886774999, 's': 0.06577040485954797, 't': 0.03561407295488884, 'u': 0.03370232185254849, 'v': 0.00588942678301625, 'w': 9.250408559711388e-05, 'x': 0.0024976103111220747, 'y': 0.007862847275754679, 'z': 0.0026826184823163022, 'space': 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 and include in final report.

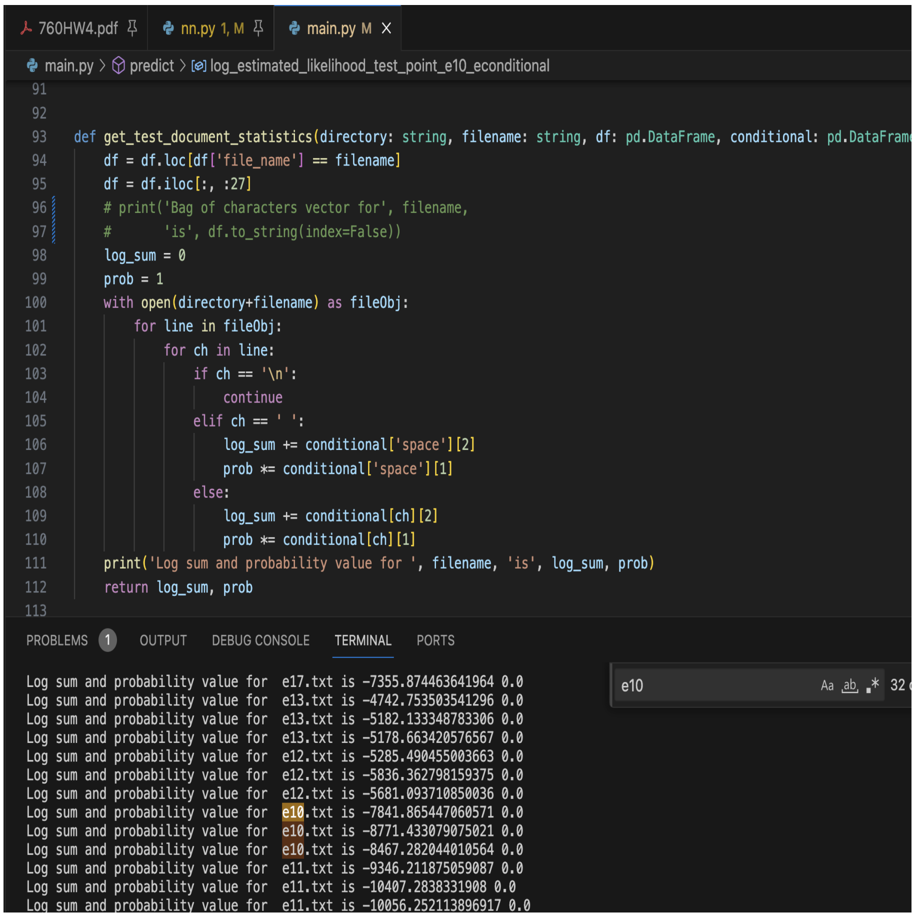
   For the bag of characters model we need to have the frequencies of each character in the file. Bag of characters vector for e10.txt is
   a b c d e f g h i j k l m n o p q r s t u v w x y z space
   164 32 53 57 311 55 51 140 140 3 6 85 64 139 182 53 3 141 186 225 65 31 47 4 38 2 498

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.

   $log\hat{p}(x \mid y = e)$ = Log sum and probability value for e conditional for the file e10.txt is -7841.865447060571, 0.0.
   $log\hat{p}(x \mid y = j)$ = Log sum and probability value for j conditional for the file e10.txt is -8771.433079075021, 0.0.
   $log\hat{p}(x \mid y = s)$ =Log sum and probability value for s conditional for the file e10.txt is -8467.282044010564, 0.0.

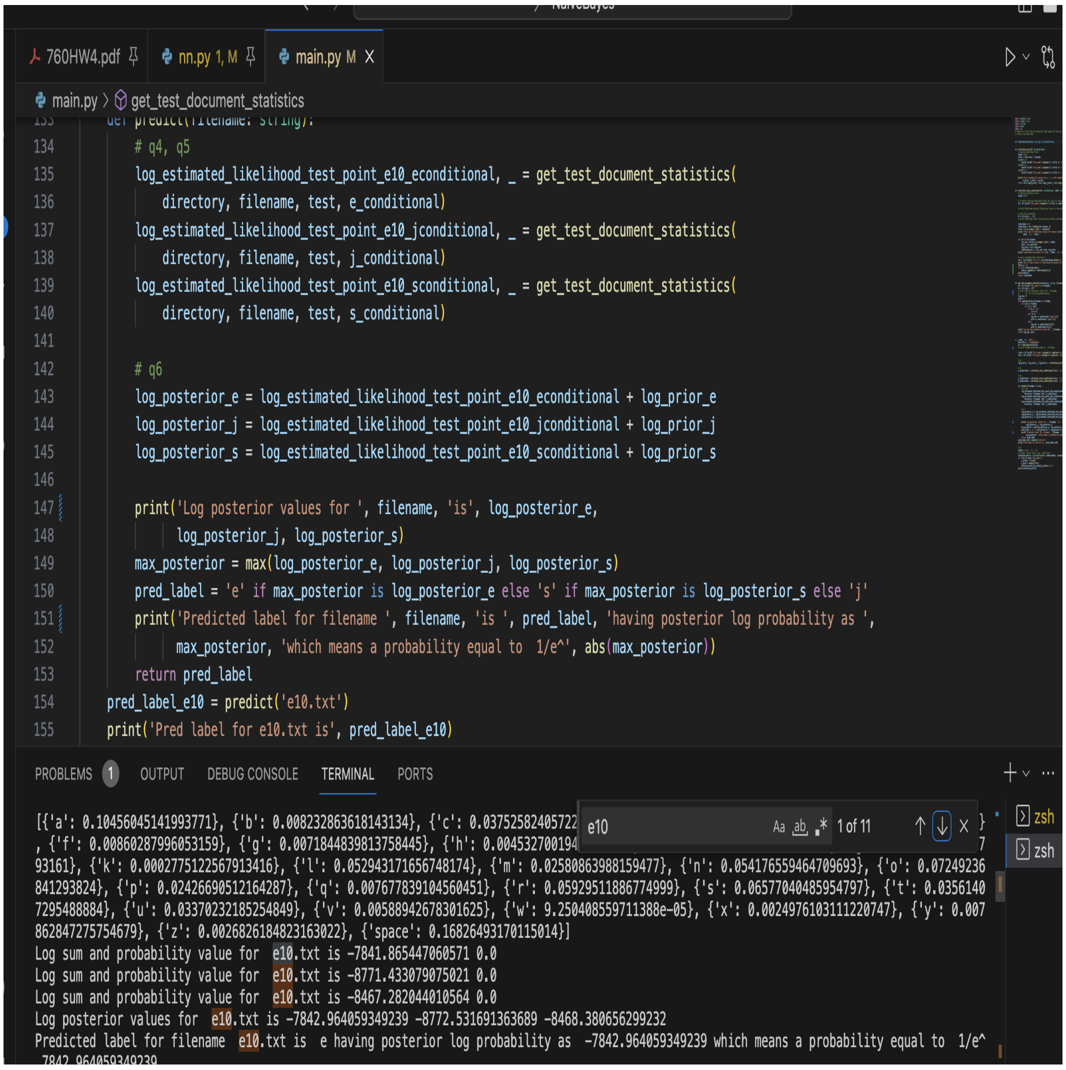


```


def get_test_document_statistics(directory: string, filename: string, df: pd.DataFrame, conditional: pd.DataFram
    df = df.loc[df['file_name'] == filename]
    df = df.iloc[:, :27]
    # print('Bag of characters vector for', filename,
    #       'is', df.to_string(index=False))
    log_sum = 0
    prob = 1
    with open(directory+filename) as fileObj:
        for line in fileObj:
            for ch in line:
                if ch == '\n':
                    continue
                elif ch == ' ':
                    log_sum += conditional['space'][2]
                    prob *= conditional['space'][1]
                else:
                    log_sum += conditional[ch][2]
                    prob *= conditional[ch][1]
    print('Log sum and probability value for ', filename, 'is', log_sum, prob)
    return log_sum, prob

```

```
Log sum and probability value for  e17.txt is -7355.874463641964 0.0
Log sum and probability value for  e13.txt is -4742.753503541296 0.0
Log sum and probability value for  e13.txt is -5182.133348783306 0.0
Log sum and probability value for  e13.txt is -5178.663420576567 0.0
Log sum and probability value for  e12.txt is -5285.490455003663 0.0
Log sum and probability value for  e12.txt is -5836.362798159375 0.0
Log sum and probability value for  e12.txt is -5681.093710850036 0.0
Log sum and probability value for  e10.txt is -7841.865447060571 0.0
Log sum and probability value for  e10.txt is -8771.433079075021 0.0
Log sum and probability value for  e10.txt is -8467.282044010564 0.0
Log sum and probability value for  e11.txt is -9346.211875059087 0.0
Log sum and probability value for  e11.txt is -10407.2838331908 0.0
Log sum and probability value for  e11.txt is -10056.252113896917 0.0
```

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.

   $\hat{p}(y = e \mid x) = 1/e^{7841}$
   $\hat{p}(y = j \mid x) = 1/e^{8771}$ $\hat{p}(y = s \mid x) = 1/e^{8467}$



```python
    def predict(filename: string):
        # q4, q5
        log_estimated_likelihood_test_point_e10_econditional, _ = get_test_document_statistics(
            directory, filename, test, e_conditional)
        log_estimated_likelihood_test_point_e10_jconditional, _ = get_test_document_statistics(
            directory, filename, test, j_conditional)
        log_estimated_likelihood_test_point_e10_sconditional, _ = get_test_document_statistics(
            directory, filename, test, s_conditional)

        # q6
        log_posterior_e = log_estimated_likelihood_test_point_e10_econditional + log_prior_e
        log_posterior_j = log_estimated_likelihood_test_point_e10_jconditional + log_prior_j
        log_posterior_s = log_estimated_likelihood_test_point_e10_sconditional + log_prior_s

        print('Log posterior values for ', filename, 'is', log_posterior_e,
              log_posterior_j, log_posterior_s)
        max_posterior = max(log_posterior_e, log_posterior_j, log_posterior_s)
        pred_label = 'e' if max_posterior is log_posterior_e else 's' if max_posterior is log_posterior_s else 'j'
        print('Predicted label for filename ', filename, 'is ', pred_label, 'having posterior log probability as ',
              max_posterior, 'which means a probability equal to  1/e^', abs(max_posterior))
        return pred_label
    pred_label_e10 = predict('e10.txt')
    print('Pred label for e10.txt is', pred_label_e10)
```

```
[{'a': 0.10456045141993771}, {'b': 0.008232863618143134}, {'c': 0.03752582405722                                    }
, {'f': 0.00860287996053159}, {'g': 0.0071844839813758445}, {'h': 0.004532700194                                    7
93161}, {'k': 0.0002775122567913416}, {'l': 0.052943171656748174}, {'m': 0.02580863988159477}, {'n': 0.054176559464709693}, {'o': 0.07249236
841293824}, {'p': 0.02426690512164287}, {'q': 0.007677839104560451}, {'r': 0.05929511886774999}, {'s': 0.06577040485954797}, {'t': 0.0356140
7295488884}, {'u': 0.03370232185254849}, {'v': 0.00588942678301625}, {'w': 9.250408559711388e-05}, {'x': 0.0024976103111220747}, {'y': 0.007
862847275754679}, {'z': 0.0026826184823163022}, {'space': 0.16826493170115014}]
Log sum and probability value for  e10.txt is -7841.865447060571 0.0
Log sum and probability value for  e10.txt is -8771.433079075021 0.0
Log sum and probability value for  e10.txt is -8467.282044010564 0.0
Log posterior values for  e10.txt is -7842.964059349239 -8772.531691363689 -8468.380656299232
Predicted label for filename  e10.txt is  e having posterior log probability as  -7842.964059349239 which means a probability equal to  1/e^
 7842.964059349239
```

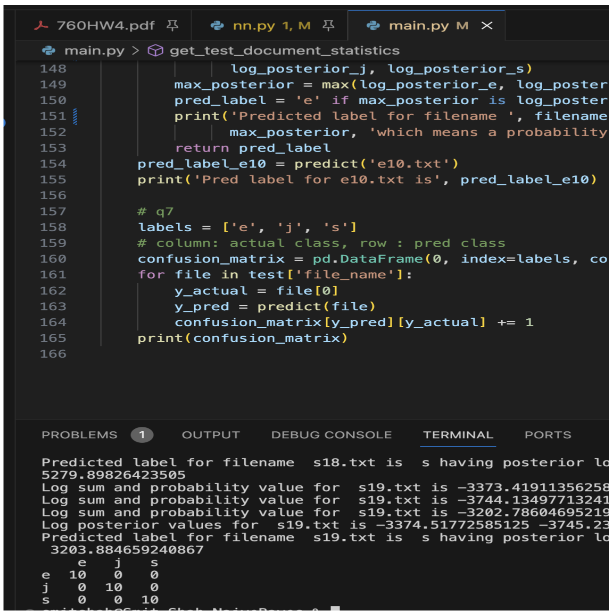
7. Evaluate the performance of your classifier on the test set (files 10.txt to 19.txt in three languages). Present the performance using a confusion matrix. A confusion matrix summarizes the types of errors your classifier makes, as shown in the table below. The columns are the true language a document 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.

   ACTUAL

   PREDICTED

	e	j	s
e	**10**	0	0
j	0	**10**	0
s	0	0	**10**

   30 test documents from e10-e19.txt, j10-j19.txt and s10-s19.txt. This means that the accuracy of Naive Bayes classifier is a whooping 100%!



```python
                        log_posterior_j, log_posterior_s)
            max_posterior = max(log_posterior_e, log_poster
            pred_label = 'e' if max_posterior is log_poster
            print('Predicted label for filename ', filename
                        max_posterior, 'which means a probability
            return pred_label
        pred_label_e10 = predict('e10.txt')
        print('Pred label for e10.txt is', pred_label_e10)

        # q7
        labels = ['e', 'j', 's']
        # column: actual class, row : pred class
        confusion_matrix = pd.DataFrame(0, index=labels, co
        for file in test['file_name']:
            y_actual = file[0]
            y_pred = predict(file)
            confusion_matrix[y_pred][y_actual] += 1
        print(confusion_matrix)
```

```
Predicted label for filename  s18.txt is  s having posterior lo
5279.89826423505
Log sum and probability value for  s19.txt is -3373.41911356258
Log sum and probability value for  s19.txt is -3744.13497713241
Log sum and probability value for  s19.txt is -3202.78604695219
Log posterior values for  s19.txt is -3374.51772585125 -3745.23
Predicted label for filename  s19.txt is  s having posterior lo
 3203.884659240867
    e   j   s
e  10   0   0
j   0  10   0
s   0   0  10
```

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.

   I believe that it will not affect the Naive Bayes classifier's prediction on the document. This stems from the fundamental principle and assumption about Naive Bayes that each character is independent of one another due to which we multiply the probability of each terms. It's only the presence or absence of specific characters that are affecting the probability value.

   The Naive Bayes classifier doesn't consider word ordering between words in the document.

   The key mathematical step that justifies this is the application of the chain rule of probability. According to this rule:

$$P(L, C_1, C_2, ..., C_n) = P(L) \times P(C_1|C) \times P(C_2|L, C_1) \times ... \times P(C_n|L, WC_1, C_2, ..., C_{n-1})$$

   Where: - L is the class label, - $C_1, C_2, ..., C_n$ are the chars in the document.

   Naive Bayes simplifies this by assuming that each word's occurrence is independent of the others given the class label. Therefore, the above equation becomes:

$$P(L, C_1, C_2, ..., C_n)P(L) \times P(C_1|L) \times P(C_2|L) \times ... \times P(C_n|L)$$

   Changing the order of characters in a document will simply change the order of terms in the multiplication and will not affect the final result, hence shuffling would result in no effect.

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,

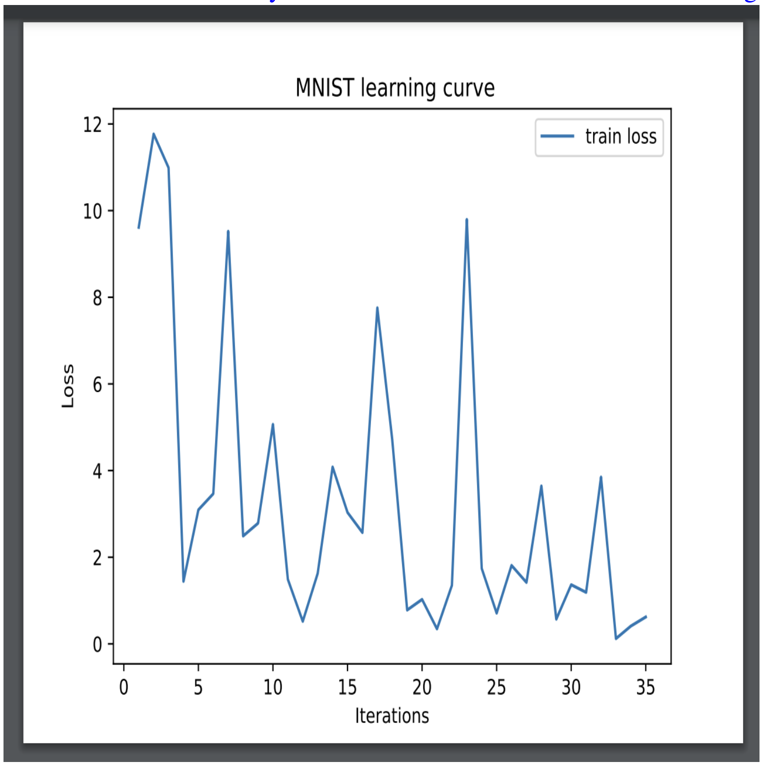
$$L(x, y) = -\sum_{i=1}^{k} y\log(\hat{y})$$

1. Derive backpropagation updates for the above neural network. (5 pts)

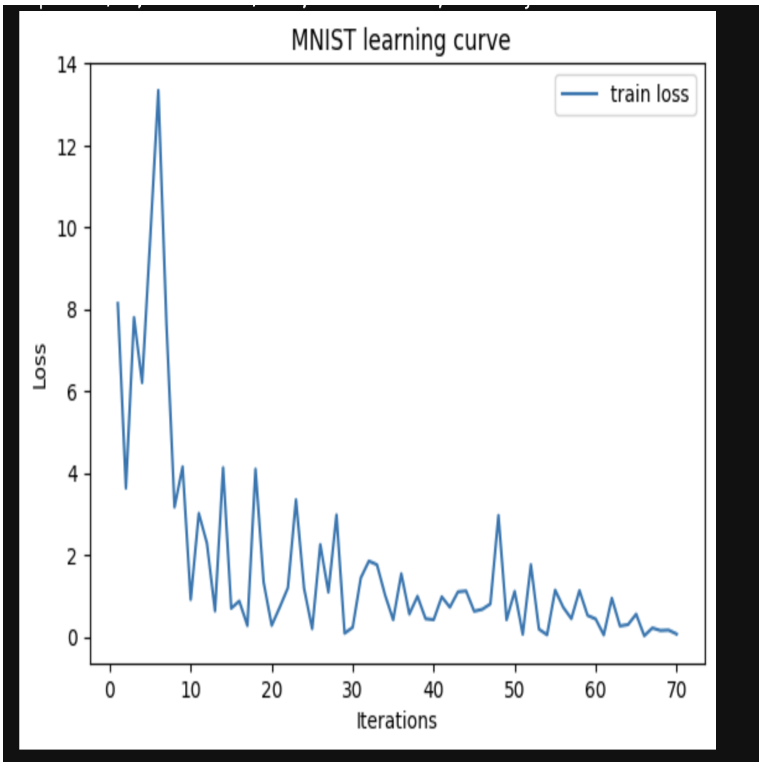
$\frac{\partial L}{\partial W_{11}^{(2)}} = \frac{\partial L}{\partial \hat{y}_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial W_{11}^{(2)}}$

where $\frac{\partial a}{\partial z} = \sigma(z) * (1 - \sigma(z)) and$
$\frac{\partial z_1}{\partial W_{11}} = x_1$
$\frac{\partial L}{\partial W_{11}} = \frac{\partial L}{\partial a_{11}} \times \frac{\partial a_{11}}{\partial W_{11}^1} = (\hat{y} - y) * (W_{11}^2) * (a_{11})(1 - a_{11}) * x_1$

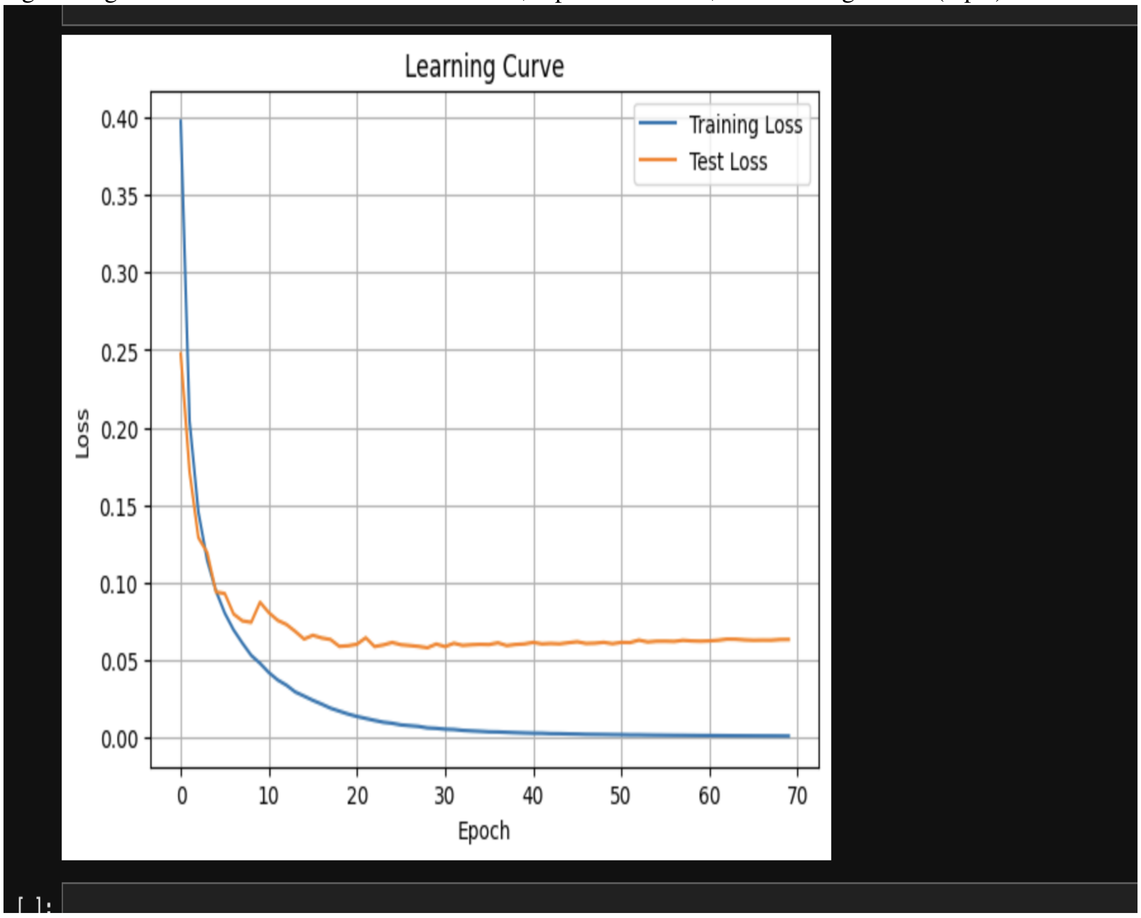
Implement it in NumPy or PyTorch using basic linear algebra operations. (e.g. You are not allowed to use auto-grad, built-in optimizer, model, etc. in this step. You can use library functions for data loading, processing, etc.). Evaluate your implementation on MNIST dataset, report test errors and learning curve. (10 pts)
On training the model for 35 iterations with weights initialized randomly , learning rate as 0.1 and batch size of 32 we receive an accuracy of 0.83 and here is the chart for the training loss



On training the model for 70 iterations with weights initialized randomly , learning rate as 0.1 and batch size of 32 we receive an accuracy of 0.92 and here is the chart for the training loss

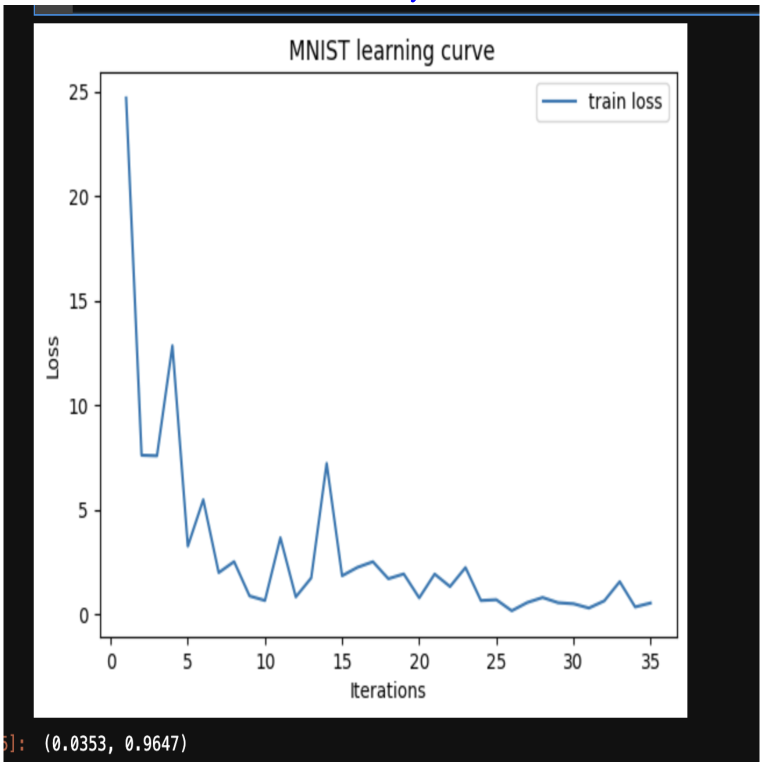


Implement the same network in PyTorch (or any other framework). You can use all the features of the framework e.g. auto-grad etc. Evaluate it on MNIST dataset, report test errors, and learning curve. (2 pts)



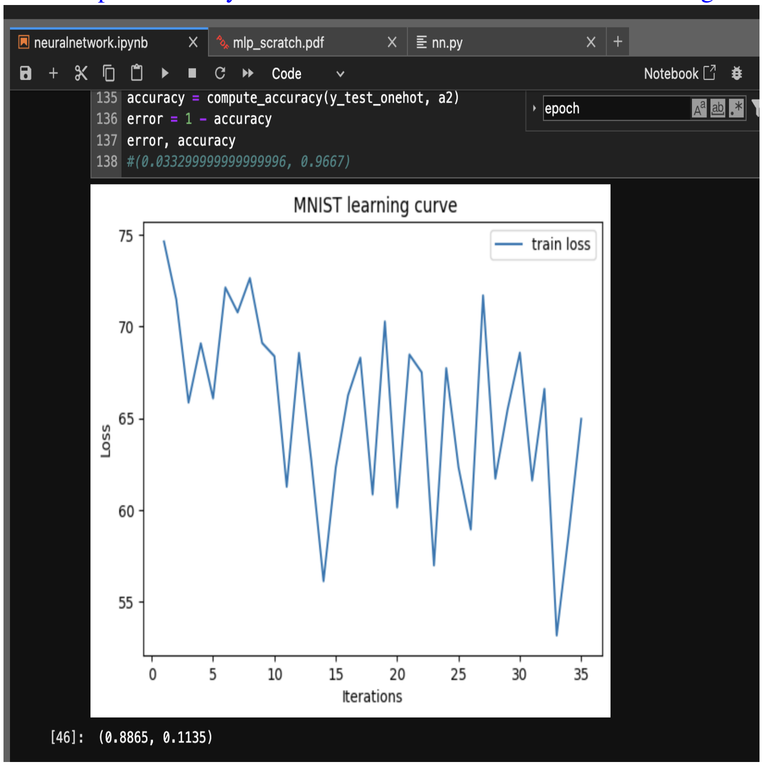
Try different weight initialization a) all weights initialized to 0, and b) initialize the weights randomly between -1 and 1. Report test error and learning curves for both. (You can use either of the implementations) (3 pts)
On training the model for 35 iterations with weights initialized randomly from -1 to 1, learning rate as 0.1 and

batch size of 32 we receive an accuracy of 0.96 and here is the chart for the training loss



```
]: (0.0353, 0.9647)
```

On training the model for 35 iterations with weights initialized as 0, learning rate as 0.1 and batch size of 32 we see a drop in accuracy of 0.88 and here is the chart for the training loss



```
accuracy = compute_accuracy(y_test_onehot, a2)
error = 1 - accuracy
error, accuracy
#(0.033299999999999996, 0.9667)
```

```
[46]: (0.8865, 0.1135)
```

Note: The results have random initialization of weights in a few cases and hence cannot be compared. Also, a random permutation of the dataset is taken which might affect the batch size of 32 loss computation.
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)