Payam Sedighiani
21801298
final report of term project eee-485

Brain tumor classification using MRI images

Problem definition

The objectivity of this project is to classify brain MRI images in to two classes:1. No tumor exist .2. Tumor exists. For this purpose, multiple machine learning algorithms will be used such as KNN, Logistic Regression, and Multinomial naïve bayes, and CNN.

Dataset description

The chosen dataset consists of 3000 images in which 20 percent 600 images are chosen as test dataset in each method. Thus, the training is done on 2400 images. 1500 images are labeled as no tumor and 1500 are label as yes tumor. The images needed preprocessing steps for applying ML technics. The images are gray but after implementing codes, realized that some of the images have still three channels so for that to be taken care I used cv2 gray scale feature for loading the images. For making the code more efficient and faster, at first implemented PCA but due to low accuracies decided to resize the images. Also, the resizing needed to happened due to all of the images having different scales.

Figure 1:Brain MRI image with tumor

Figure 2:Brain MRI image with no tumor

The above dataset is taken from Kaggle and the link to the dataset is in references.

ML methods

KNN

The first method that was used is KNN because it always has a sufficient accuracy and can be implemented on any data with simplicity. The followings are the reasons for choosing KNN:

1. KNN has no training period hence it is faster
2. Easy to implement
3. For gray images since there is less complexity and the values are discrete, it is easy and sensible to use KNN because the distances are obvious between two pictures

KNN steps:

1. Test data and training Data extracted as NumPy arrays and preprocessing steps are:

Payam Sedighiani
21801298
final report of term project eee-485

2. Resizing images, gray scaling and reshaping each image to one row and the final is a one row NumPy array. Also, Labeled the tumor images as 1 and the no tumor one as -1

```
# yes dataset labeling(1)

list_of_pics_yes = list()
for i in range(0,1500):
    train_image_yes= imread('y'+str(i)+'.jpg')
    train_y = cv2.resize(train_image_yes, (25,25))
    if len(train_y.shape) == 3:
        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)
    train_y = np.reshape(train_y,625)
    train_y = np.append(train_y,1)
    list_of_pics_yes.append(train_y)
list_of_pics_yes = np.asarray(list_of_pics_yes)


# no dataset labeling(0)
list_of_pics_no = list()
for i in range(0,1500):
    train_image_no= imread('no'+str(i)+'.jpg')
    train_n = cv2.resize(train_image_no, (25,25))
    if len(train_n.shape) == 3:
        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)
    train_n = np.reshape(train_n,625)
    train_n = np.append(train_n,-1)
    list_of_pics_no.append(train_n)
list_of_pics_no = np.asarray(list_of_pics_no)
```

3. Found the Euclidian distance of each test data with the 3000 images
$d = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$.

```
tot = []
for t in (np.delete(test_dataset,-1,1)):
    ms= []
    for j in (np.delete(train_dataset,-1,1)):
        Sy = np.sqrt(np.sum(np.square(t-j)))
        ms.append(Sy)
        np.asarray(ms)
    tot.append(ms)
tot = np.asarray(tot)
```

4. Run the model for 4 values of k = [3,5,7,11]

```
# 4 values of k
k = [3,5,7,11]
for kth in k:
    acs = []
    print("k value is :",kth)
    for l,p,f in zip(a,b,(range(1,6))):
        print(f,"fold")
```

5. Used k-fold cross validation and used 5-fold and at the end took the average accuracy

```
# cross validation k-fold = 5-fold
a= [0,300,600,900,1200]
b = [300,600,900,1200,1500]


# 4 values of k
k = [3,5,7,11]
for kth in k:
    acs = []
    print("k value is :",kth)
    for l,p,f in zip(a,b,(range(1,6))):
        print(f,"fold")
```

6. Derived the confusion matrix for the data

7. Then found accuracy using the following formula:
   Accuracy = (TP+TN)/ (TP+TN) +(FP+FN)

KNN Results

```
k value is : 3
1 fold
confusion matrix =
 [[260   1]
 [ 40 299]]
accuracy =
 93.16666666666667 %
2 fold
confusion matrix =
 [[296  32]
 [  4 268]]
accuracy =
 94.0 %
3 fold
confusion matrix =
 [[238   1]
 [ 62 299]]
accuracy =
 89.5 %
4 fold
confusion matrix =
 [[270   3]
 [ 30 297]]
accuracy =
 94.5 %
5 fold
confusion matrix =
 [[154   0]
 [146 300]]
accuracy =
 75.66666666666667 %
average acurracy for k = 3  is  89.36
```

```
k value is : 5
1 fold
confusion matrix =
 [[236   3]
 [ 64 297]]
accuracy =
 88.83333333333333 %
2 fold
confusion matrix =
 [[281  29]
 [ 19 271]]
accuracy =
 92.0 %
3 fold
confusion matrix =
 [[192   1]
 [108 299]]
accuracy =
 81.83333333333333 %
4 fold
confusion matrix =
 [[246   7]
 [ 54 293]]
accuracy =
 89.83333333333333 %
5 fold
confusion matrix =
 [[128   2]
 [172 298]]
accuracy =
 71.0 %
average acurracy for k = 5  is  84.6999
```

```
k value is : 7
1 fold
confusion matrix =
 [[212   4]
 [ 88 296]]
accuracy =
 84.66666666666667 %
2 fold
confusion matrix =
 [[270  30]
 [ 30 270]]
accuracy =
 90.0 %
3 fold
confusion matrix =
 [[173   2]
 [127 298]]
accuracy =
 78.5 %
4 fold
confusion matrix =
 [[217  10]
 [ 83 290]]
accuracy =
 84.5 %
5 fold
confusion matrix =
 [[112   5]
 [188 295]]
accuracy =
 67.83333333333333 %
average acurracy for k = 7  is  81.1
```

```
k value is : 11
1 fold
confusion matrix =
 [[182   3]
 [118 297]]
accuracy =
 79.83333333333333 %
2 fold
confusion matrix =
 [[249  29]
 [ 51 271]]
accuracy =
 86.66666666666667 %
3 fold
confusion matrix =
 [[137   3]
 [163 297]]
accuracy =
 72.33333333333333 %
4 fold
confusion matrix =
 [[170  10]
 [130 290]]
accuracy =
 76.66666666666667 %
5 fold
confusion matrix =
 [[ 92   5]
 [208 295]]
accuracy =
 64.5 %
average acurracy for k = 11  is  76.
```

Payam Sedighiani
21801298
final report of term project eee-485

KNN Comments:

1. As the value of k increases the average accuracy of 5-fold decreases
2. As the value of k increases the code runs for longer time
3. Using k-fold cross validation method the bias is reduced
4. Also, as we increase the value of k, we see that we get a more distributed weight on the results.
5. Overall, the accuracies are above 80 percent which shows the average accuracy of around 80.

LOGISTIC REGRESSION

The second used ML method is logistic regression and it is chosen due to the following reasons:

1. It is very efficient for classification problems
2. It is very fast in classifying
3. It gives a good measure for how a predictor is good or not
4. Also, it gives a good simulation for what are the losses and training accuracies

Logistic regression steps:

1. The data is preprocessed for this method similar to KNN with some differences. In here the yes tumor images are labeled as 1 and no tumor images are labeled as 0 and each image after resizing is changed to one row NumPy array.

```
list_of_pics_yes = list()
for i in range(0,1500):
    train_image_yes= imread('y'+str(i)+'.jpg')
    train_y = cv2.resize(train_image_yes, (35,35))
    if len(train_y.shape) == 3:
        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)
    train_y = np.reshape(train_y,1225)
    train_y = np.append(train_y,1)
    list_of_pics_yes.append(train_y)
list_of_pics_yes = np.asarray(list_of_pics_yes)


# no dataset labeling(0)
list_of_pics_no = list()
for i in range(0,1500):
    train_image_no= imread('no'+str(i)+'.jpg')
    train_n = cv2.resize(train_image_no, (35,35))
    if len(train_n.shape) == 3:
        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)
    train_n = np.reshape(train_n,1225)
    train_n = np.append(train_n,0)
    list_of_pics_no.append(train_n)
list_of_pics_no = np.asarray(list_of_pics_no)
```

The training data and test data are normalized as well

```
x_labels = train_dataset[:,-1]
y_labels = test_dataset[:,-1]

train_dataset = train_dataset[:,:-1]

test_dataset = test_dataset[:,:-1]
# normalizing

xtrainnorm = (train_dataset-(np.amin(train_dataset))+0)/((np.amax(train_dataset))-(np.amin(train_dataset)))

xtestnorm = (test_dataset-(np.amin(train_dataset))+0)/((np.amax(train_dataset))-(np.amin(train_dataset)))
```

Payam Sedighiani
21801298
final report of term project eee-485

To be able to implement the logistic regression formulas for weight, bias and predictors the data is transposed and reshaped:

```
xtrainnorm = xtrainnorm.T
xtestnorm = xtestnorm.T

ytrain = x_labels.reshape(1,xtrainnorm.shape[1])
ytest = y_labels.reshape(1,xtestnorm.shape[1])
```

2. For initializing weights and biases three methods were used: 1. Zeros 2. Normal gaussian distribution 3. Uniform distribution.

```
if c == 1:
  b = np.random.normal(0, 1)
  w = np.random.normal(0, 1, n)
  w = np.reshape(w, (n, 1))
elif c== 2:
  b = np.random.uniform(0, n)
  w = np.random.uniform(0, 1, n)
  w = np.reshape(w, (n, 1))

else:
  w = np.zeros((n,1))
  b = 0
```

In the logistic function that was written, the C variable is used to select one of the initialization methods.

3. Then the following piece of code was written for running the logistic regression to be able to update weights and biases and finally give predictors using the following formulas:

$$W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_n \end{bmatrix}_{nx1} \quad X = \begin{bmatrix} . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix}_{nxm}$$

$$Y = [. \quad . \quad . \quad . \quad .]_{1xm}$$

$$\sigma = \frac{1}{(1+e^{-x})} \text{ .... (sigmoid}$$

$$A = \sigma(W^T * X + b) \text{ ....}$$

Where w is the weights, x are the inputs images, y are the labels corrosponding to images, sigma is for defining sigmoid function and A is predictions.

Payam Sedighiani
21801298
final report of term project eee-485

```python
# logistic regression

def logistic(xtrain,ytrain,learningrate,epochs,c,x_test,y_test):
    m = xtrain.shape[1]
    n = xtrain.shape[0]
  # gaussian or not

  # stochastic gradient decent

    if c == 1:
      b = np.random.normal(0, 1)
      w = np.random.normal(0, 1, n)
      w = np.reshape(w, (n, 1))
    elif c== 2:
      b = np.random.uniform(0, n)
      w = np.random.uniform(0, 1, n)
      w = np.reshape(w, (n, 1))

    else:
      w = np.zeros((n,1))
      b = 0
    costlist= []
    acs = []
    js = []
    for i in range(epochs):
      k = np.dot(w.T,xtrain) + b
      j = sig(k)
      # the cost is observed to see whether the algorithm is working or not
      cost = -(1/m)*np.sum(ytrain*np.log(j)+(1-ytrain)*np.log(1-j))
      dw = (1/m)*np.dot(j-ytrain,xtrain.T)
      db = (1/m)*np.sum(j-ytrain)
      w = w - learningrate*dw.T
      b = b - learningrate*db
      ac,jp = accuracy(x_test,y_test,w,b)
      costlist.append(cost)
      acs.append(ac)
      js.append(jp)
      if(i%(epochs/10)) == 0:
          print(cost)


    return w, b, costlist,acs,js
```

4. Accuracies are measured using the following function:

```python
def accuracy(x,y,w,b):
  k = np.dot(w.T,x) + b
  j = sig(k)

  j = j>0.5
  j = np.array(j,dtype = 'int64')

  acuracy = (1-np.sum(np.absolute(j-y))/y.shape[0])*100
  return acuracy,j
```

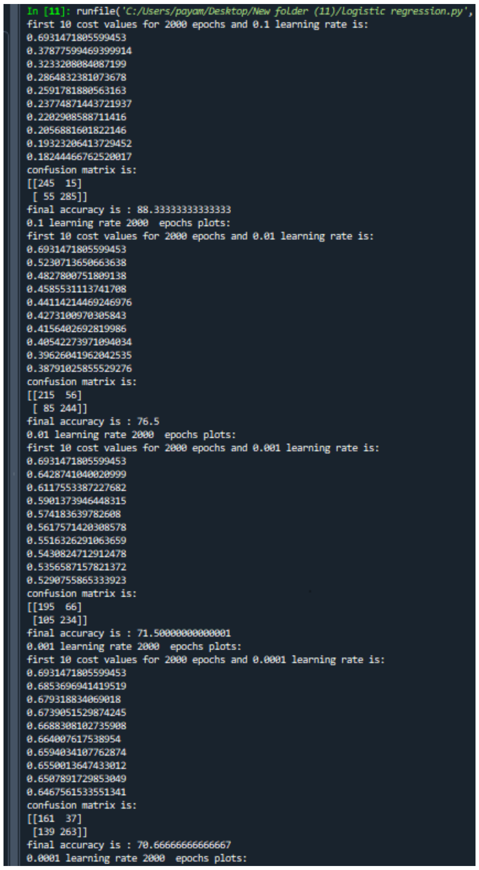
5. For analyzing and tuning the hyperparameters, 4 values for epochs and 4 values for learning rate are chosen. Also, three ways of initializing is used as mention before.

```python
# print(y_labels.shape)
epochs = [2000,5000,7000,10000]
learningrate = [10**-1,10**-2,10**-3,10**-4]

for e in epochs:
    for l in learningrate:

        print("first 10 cost values for",e,"epochs and",l,"learning rate is:")
        w,b,costlist,acs,jp = logistic(xtrainnorm,x_labels,learningrate = l,epochs = e,c = 0,
                                       x_test = xtestnorm,y_test = y_labels)
        jpp = np.asarray(jp)[-1][0]

        c = conf(jpp,y_labels)
        print("confusion matrix is:")
        print(c)
        print("final accuracy is :",acs[-1])

        plt.plot(np.arange(e),costlist)
        plt.xlabel('epochs')
        plt.ylabel('cost')
        plt.figure()
        plt.plot(np.arange(e),acs)
        plt.xlabel('epochs')
        plt.ylabel('accuracy')
        plt.figure()
```

Payam Sedighiani
21801298
final report of term project eee-485
LOGISTIC REGRESSION Results

**NOTE: Due to limited number of pages I just put the values and plots for 2000 and 10000 epochs for all of the learning rates.

Cost for 2000 epochs, at each learning rate, and gaussian initialization:



```
In [11]: runfile('C:/Users/payam/Desktop/New folder (11)/logistic regression.py',
first 10 cost values for 2000 epochs and 0.1 learning rate is:
0.6931471805599453
0.3787759946939914
0.3233208084087199
0.2864832381073678
0.2591781880563163
0.2377487144372193
0.2202908588711416
0.2056881601822146
0.1932320641372945
0.1824446676252017
confusion matrix is:
[[245  15]
 [ 55 285]]
final accuracy is : 88.33333333333333
0.1 learning rate 2000  epochs plots:
first 10 cost values for 2000 epochs and 0.01 learning rate is:
0.6931471805599453
0.5230713650663638
0.4827800751809138
0.4585531113741708
0.4411421446924697
0.4273100970305843
0.4156402692819986
0.4054227397109403
0.3962604196204253
0.3879102585552927
confusion matrix is:
[[215  56]
 [ 85 244]]
final accuracy is : 76.5
0.01 learning rate 2000  epochs plots:
first 10 cost values for 2000 epochs and 0.001 learning rate is:
0.6931471805599453
0.6428741040020999
0.6117553387227682
0.5901373946448315
0.574183639782608
0.5617571420308578
0.5516326291063659
0.5430824712912478
0.5356587157821372
0.5290755865333923
confusion matrix is:
[[195  66]
 [105 234]]
final accuracy is : 71.50000000000001
0.001 learning rate 2000  epochs plots:
first 10 cost values for 2000 epochs and 0.0001 learning rate is:
0.6931471805599453
0.6853696941419519
0.679318834069018
0.6739051529874245
0.6688308102735908
0.664007617538954
0.6594034107762874
0.6550013647433012
0.6507891729853049
0.6467561533551341
confusion matrix is:
[[161  37]
 [139 263]]
final accuracy is : 70.66666666666667
0.0001 learning rate 2000  epochs plots:
```

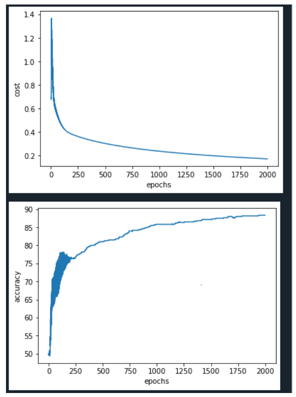


Figure 5: epochs 2000, learning rate 0.1

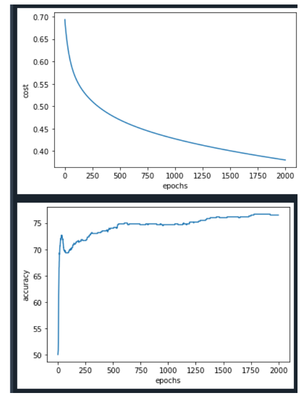


Figure 6:epochs 2000, learning rate 0.01

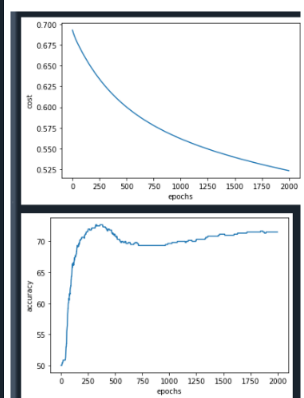


Figure 4: epochs 2000, learning rate 0.001

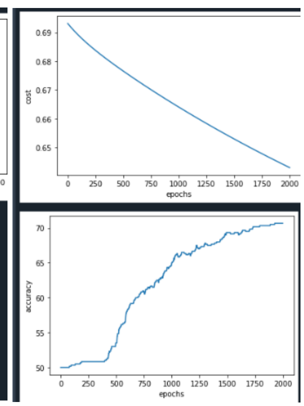
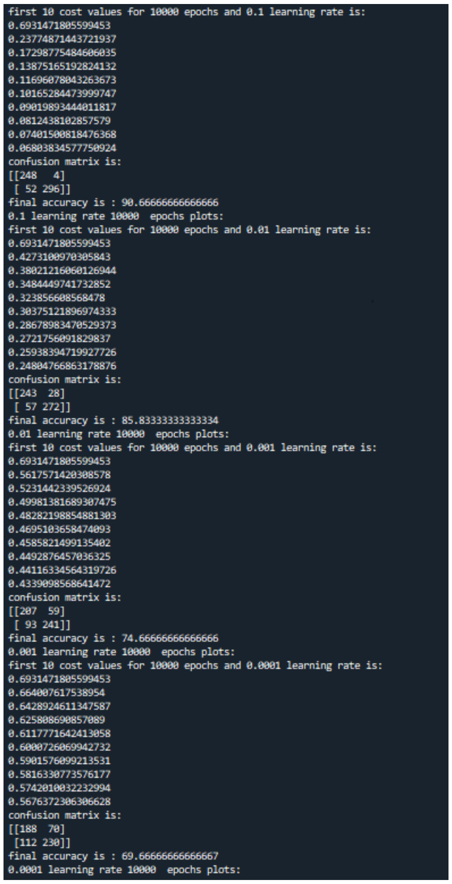


Figure 3: epochs 2000, learning rate 0.0001

Figure 7:cost and accuracy and confusion matrix for 2000 and 4 values of learning rate

As we can see from the above images, the cost decreases in each learning rate and epoch. For 2000 epochs the best learning rate value is 0.1. since it has a more efficient plot that its giving.

Payam Sedighiani
21801298
final report of term project eee-485
Cost for 10000 epochs, at each learning rate, and gaussian initialization:



```
first 10 cost values for 10000 epochs and 0.1 learning rate is:
0.6931471805599453
0.23774871443721937
0.17298775484606035
0.13875165192824132
0.11696078043263673
0.10165284473999747
0.09019893444011817
0.0812438102857579
0.07401500818476368
0.06803834577750924
confusion matrix is:
[[248   4]
 [ 52 296]]
final accuracy is : 90.66666666666666
0.1 learning rate 10000  epochs plots:
first 10 cost values for 10000 epochs and 0.01 learning rate is:
0.6931471805599453
0.4273100970305843
0.38021216060126944
0.3484449741732852
0.323856608568478
0.30375121896974333
0.28678983470529373
0.2721756091829837
0.25938394719927726
0.24804766863178876
confusion matrix is:
[[243  28]
 [ 57 272]]
final accuracy is : 85.83333333333334
0.01 learning rate 10000  epochs plots:
first 10 cost values for 10000 epochs and 0.001 learning rate is:
0.6931471805599453
0.5617571420308578
0.5231442339526924
0.49981381689307475
0.48282198854881303
0.4695103658474093
0.4585821499135402
0.4492876457036325
0.44116334564319726
0.4339098568641472
confusion matrix is:
[[207  59]
 [ 93 241]]
final accuracy is : 74.66666666666666
0.001 learning rate 10000  epochs plots:
first 10 cost values for 10000 epochs and 0.0001 learning rate is:
0.6931471805599453
0.664007617538954
0.6428924611347587
0.625808690857089
0.6117771642413058
0.6000726069942732
0.5901576099213531
0.5816330773576177
0.5742010032232994
0.5676372306306628
confusion matrix is:
[[188  70]
 [112 230]]
final accuracy is : 69.66666666666667
0.0001 learning rate 10000  epochs plots:
```

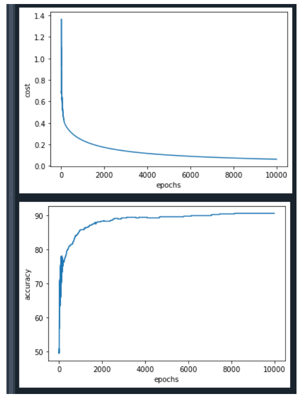


Figure 10: epochs 10000, learning rate 0.1

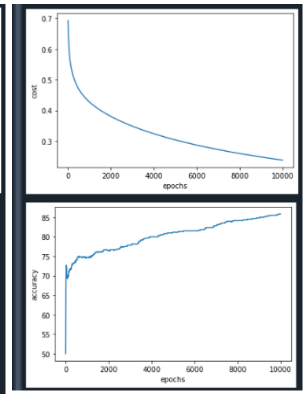


Figure 11:epochs 10000, learning rate 0.01

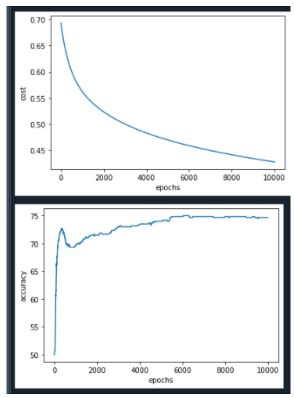


Figure 9: epochs 10000, learning rate 0.001

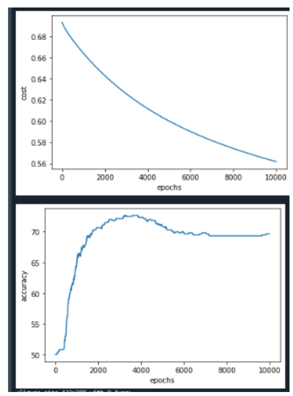


Figure 8: epochs 10000, learning rate

Figure 12: cost and accuracy and confusion matrix for 10000 and 4 values of learning rate

as it can be seen from the above plots the learning rate 0.1 is giving better results and higher accuracy for 10000 epochs.

Overall, the 10000 epochs for 0.1 learning rate with gaussian initialization was chosen as tuned params.

LOGISTIC REGRESSION comments:

1. The code works properly and the cost is decreasing
2. The tuned hyperparameters are learning rate, epochs and initialization of weight and bias
3. For learning rate 0.1 and 0.0001 we were getting better result for each epoch value
4. At 7000 epochs and 10000 epochs with 0.1 learning rate the accuracies of 90% and 90.66% was achieved
5. The random gaussian initialization helps the predicators to less bias, hence it was better than zero and normal distributions

Payam Sedighiani
21801298
final report of term project eee-485

MULTINOMIAL NAÏVE BAYES

This method was chosen due to the following results:

1. It is very fast
2. With less training data can give good predictors
3. It gives decent accuracy for binary and multi class classification

This method follows the naïve bayes rule of probability and for the implementation of this method the following formulas were used from bag of words method:

$$\hat{y}_i = \arg\max_{y_k} \left(\log \mathbf{P}(Y = y_k) + \sum_{j=1}^{|V|} t_{w_j,i} * \log \mathbf{P}(X_j \mid Y = y_k) \right)$$

And for optimizing the formula I found the MAP as the following:

$$\theta_{j \mid y=y_k} \equiv \frac{T_{j,y=y_k} + \alpha}{(\sum_{j=1}^{|V|} T_{j,y=y_k}) + \alpha * |V|}$$

$$\pi_{y=y_k} \equiv \mathbf{P}(Y = y_k) = \frac{N_{y_k}}{N}$$

Multinomial naïve bayes steps:

1. First extracted the images with labeling tumor as 1 and no tumor as 2

```
# yes_train labeling(1)

list_of_pics_yes = list()
for i in range(0,1500):
    train_image_yes= imread('y'+str(i)+'.jpg')
    train_y = cv2.resize(train_image_yes, (20,20))
    if len(train_y.shape) == 3:
        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)
    train_y = train_y/255
    train_y = np.reshape(train_y,400)
    train_y = np.append(train_y,1)
    list_of_pics_yes.append(train_y)
list_of_pics_yes = np.asarray(list_of_pics_yes)


# no_train labeling(2)
list_of_pics_no = list()
for i in range(0,1500):
    train_image_no= imread('no'+str(i)+'.jpg')
    train_n = cv2.resize(train_image_no, (20,20))
    if len(train_n.shape) == 3:
        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)
    train_n = train_n/ 255
    train_n = np.reshape(train_n,400)
    train_n = np.append(train_n,2)
    list_of_pics_no.append(train_n)
list_of_pics_no = np.asarray(list_of_pics_no)
```

Payam Sedighiani
21801298
final report of term project eee-485

2. Initialized the cross validation for k-fold

```
# train dataset and labels dataset
a= [0,300,600,900,1200]
b = [300,600,900,1200,1500]

acs_a = []
for l,p in zip(a,b):
    train_dataset = np.concatenate(((np.concatenate((list_of_pics_no[p:1500],list_of_pics_no[0:l]),
                                axis = 0),np.concatenate((list_of_pics_yes[p:1500],list_of_pics_yes[0:l]),axis = 0))),axis = 0)
    train_dataset_withoutlabels = np.delete(train_dataset,-1,1)
    # print(train_dataset_withoutlabels.shape)
    list_of_pics_ytest = list_of_pics_yes[l:p]

    # # no_test labeling(-1)
    list_of_pics_ntest = list_of_pics_no[l:p]
    test_dataset = np.concatenate((list_of_pics_ntest,list_of_pics_ytest),axis = 0)
    test_dataset_withoutlabels = np.delete(test_dataset,-1,1)
    # print(test_dataset)
    # print(test_dataset_withoutlabels)
    # train labes
    train_labels = train_dataset[:,-1]
    train_labels_yes =np.count_nonzero((train_labels)==1)
    train_labels_no  =np.count_nonzero((train_labels)==2)

    # test labels
    test_labels = test_dataset[:,-1]

    test_labels_yes =np.count_nonzero((test_labels)==1)
    test_labels_no  =np.count_nonzero((test_labels)==2)
```

3. Found the values of P(y=yk) and then found teta1 and teta2 since we have two classes

```
#  probablity of classes

prob_class =[train_labels_yes/3000,train_labels_no/3000]


# print(test_dataset_withoutlabels.shape)

# tetas for class 1
a =1

T1 = 0
TT1 = []
for k in range(400):
  T1 = 0
  for i, j in zip(train_dataset_withoutlabels[:, [k]], train_labels):
    if j == 1 and i != 0:
      T1+=1
  TT1.append(T1)
TT1 = np.asarray(TT1)
sigT1 = np.array(sum(TT1))
TETA1 = np.log(TT1/sigT1)

TETA1a = np.log((TT1+a)/(sigT1 +a*400))
# print(TT1)
# print(sigT1)
# print(TETA1)

# print('sssssssss')
# tetas for class 2
T2 = 0
TT2 = []
for k in range(400):
  T2 = 0
  for i, j in zip(train_dataset_withoutlabels[:, [k]], train_labels):
    if j == 2 and i != 0:
      T2+=1
  TT2.append(T2)
TT2 = np.asarray(TT2)
sigT2 = np.array(sum(TT2))
TETA2 = np.log(TT2/sigT2)


TETA2a = np.log((TT2+a)/(sigT2 +a*400))
# print(TT2)
# print(sigT2)
# print(TETA2.shape)
```

4. Then found the predictions

```
# finding predictions

y1 = (prob_class[0]) + np.dot(test_dataset_withoutlabels , TETA1a)
y1 = np.exp(y1)
y2 = (prob_class[1]) + np.dot(test_dataset_withoutlabels , TETA2a)
y2 = np.exp(y2)
```

5. Used confusion matrix and accuracy matrix

```
def conf(y_pre, y_tes):
  con_matrix = np.zeros((2, 2), dtype=int)
  for x in range(len(y_pre)):
    if y_pre[x] == 2 and y_tes[x] == 2:
      con_matrix[1][1] += 1  # tn
    elif y_pre[x] == 2 and y_tes[x] == 1:
      con_matrix[1][0] += 1  # fn
    elif y_pre[x] == 1 and y_tes[x] == 2:
      con_matrix[0][1] += 1  # fp
    else:
      con_matrix[0][0] += 1  # tp

  return np.asarray(con_matrix)
```

Payam Sedighiani
21801298
final report of term project eee-485

Multinomial naïve bayes Results:

The result after cross validation as the average for 5-fold is 65.7 percent.

```
In [16]: runfile('C:/Users/payam/Desktop/New folder (11)/naive
confusion matrix =
 [[194  69]
 [106 231]]
accuracy =
 70.83333333333333 %
confusion matrix =
 [[238  44]
 [ 62 256]]
accuracy =
 82.33333333333333 %
confusion matrix =
 [[132  60]
 [168 240]]
accuracy =
 62.0 %
confusion matrix =
 [[104  69]
 [196 231]]
accuracy =
 55.833333333333336 %
confusion matrix =
 [[105  60]
 [195 240]]
accuracy =
 57.5 %
the average accuracy for 5-fold cross validation is 65.7
```

Multinomial naïve bayes comments:

1. The result is almost sufficient; however, the other methods have given better results.
2. For hyperparameters teta I used MLE but apparently this method is not good enough for image dataset as the others
3. The cross validation has significantly removed the bias in the data set as we see there is 82% accuracy and 55.8%

CNN

This method is chosen because it is one of the most used and most convenient methods for image classification for the following reasons:

1. High speed for less biased and high accuracy
2. Do not require humans to find important features
3. It learns so it does not require human supervision

Payam Sedighiani
21801298
final report of term project eee-485

CNN Steps:

1. The data is preprocessed as following for gray scaling, resizing:

```
list_of_pics_yes = []
yes_labels = []

for i in range(0,1500):
    train_image_yes= cv2.imread('y'+str(i)+'.jpg',0)
    train_y = cv2.resize(train_image_yes, (30,30))
    list_of_pics_yes.append(train_y)
    yes_labels.append(1)
list_of_pics_yes = np.asarray(list_of_pics_yes)
yes_labels = np.asarray(yes_labels)

# no dataset labeling(0)
list_of_pics_no = []
no_labels = []
for i in range(0,1500):
    train_image_no= cv2.imread('no'+str(i)+'.jpg',0)
    train_n = cv2.resize(train_image_no, (30,30))
    list_of_pics_no.append(train_n)
    no_labels.append(0)
list_of_pics_no = np.asarray(list_of_pics_no)
no_labels = np.asarray(no_labels)
```

2. To perform CNN the following three classes are written:
   1. Convolution class: in this class, the images are becoming patches and convolved with a filter. The chosen filter number is 10 and 3 by 3 dimensional. And at the end after updating the parameters it returns them. The following piece of code is for this class:

```
class conv:
    def __init__(self,n_filters, s_filter):
        self.n_filters = n_filters
        self.s_filter = s_filter
        self.c_filter = np.random.randn(n_filters,s_filter,s_filter )/(s_filter * s_filter)

    # generating buffer for saving patches
    def patch(self,img):

        h,w = img.shape
        self.img = img
        for i in range(h-(self.s_filter) +1):
            for j in range(w-(self.s_filter) +1):
                img_p = img[i: (i+self.s_filter), j : (j + self.s_filter)]
                yield img_p, i, j

    def f_propagation(self,img):
        h,w = img.shape
        output_conv = np.zeros((h - self.s_filter+1,w-self.s_filter+1,self.n_filters))
        for img_pch,i,j in self.patch(img):
            output_conv[i,j] = np.sum(img_pch *self.c_filter, axis = (1,2))

        return output_conv


    def b_propagation(self,out_dl, alpha):

        params_df = np.zeros(self.c_filter.shape)

        for img_pch,i,j in self.patch(self.img):
            for l in range(self.n_filters):
                params_df[l] += img_pch*out_dl[i,j,l]

        self.c_filter -= alpha * params_df
        return params_df
```

Payam Sedighiani
21801298
final report of term project eee-485

2. Max_pool: in this class we do max pool operation which reduces the dimensionality of images by making small patches and then makes assumptions on the features using forward propagation and backward propagation to find loss. In this class, no updating params is happening.

```
class max_pool:

    def __init__(self,s_filter):
        self.s_filter =s_filter

    def patch(self,img):
        new_h = img.shape[0]//self.s_filter
        new_w = img.shape[1]//self.s_filter
        self.img = img

        # extraction of image pacthes
        for i in range(new_h):
            for j in range(new_w):
                img_p = img[(i*self.s_filter): (i *self.s_filter + self.s_filter),
                            (j*self.s_filter): (j *self.s_filter + self.s_filter)]
                yield img_p,i,j

    def f_propagation(self,img):
        h,w, n_filters = img.shape
        output_fp = np.zeros((h//self.s_filter,w // self.s_filter, n_filters))

        for img_pch,i,j in self.patch(img):
            output_fp[i,j] = np.amax(img_pch,axis = (0,1))

        return output_fp

    def b_propagation(self,out_dl):
        pool_dl = np.zeros(self.img.shape)
        for img_pch,i,j in self.patch(self.img):
            h,w,n_filters = img_pch.shape
            val_max = np.amax(img_pch,axis = (0,1))

            for l in range(h):
                for a in range(w):
                    for b in range(n_filters):
                        if img_pch[l,a,b] == val_max[b]:
                            pool_dl[i *self.s_filter + l, j *self.s_filter+a,b] = out_dl[i,j,b]
            return pool_dl
```

Payam Sedighiani
21801298
final report of term project eee-485

3. Soft_max: in this class, the params made in convolution layer and the patches from max_pool class is used and through forward and backward propagation updates the params and gives predicators

```
class soft_max:
    def __init__(self,x_n,s_n):
        self.w = np.random.randn(x_n,s_n)/ x_n
        self.b = np.zeros(s_n)

    def f_propagation(self,img):

        self.img_first = img.shape
        # flatteining the cubes
        img_changed = img.flatten()
        self.changed_x = img_changed
        val_out = np.dot(img_changed,self.w)+self.b
        self.y = val_out
        out_ex = np.exp(val_out)
        # probability output
        return out_ex/np.sum(out_ex,axis=0)

    def b_propagation(self,out_dl,alpha):

        for i,g in enumerate(out_dl):
            if g == 0:
                continue
            trequ = np.exp(self.y)
            tot = np.sum(trequ)

            # output(z) gradient

            dydz = - trequ[i] * trequ/(tot**2)
            dydz[i] = trequ[i]*(tot-trequ[i])/(tot**2)

            # weight and bias and input gradients with respect to tot

            dzdw = self.changed_x
            dzdb = 1
            dzdx = self.w

            # loss gradient
            dldz = g *dydz

            # loss gradient with respect to bias,weights,input

            dldw = dzdw[np.newaxis].T @ dldz[np.newaxis]
            dldb = dldz * dzdb
            dldx = dzdx @ dldz
        self.w -= alpha *dldw
        self.b -= alpha *dldb

        return dldx.reshape(self.img_first)
```

3. in this step the training is happening using the following function:

```
def train_cnn(img,label,alpha = 0.05):
    # forward propagation
    output_f, ent_loss, acs_preds = forward_cnn(img,label)
    grad = np.zeros(10)
    grad[label] = -1/output_f[label]

    # back propagation

    grad_b =softmax_ob.b_propagation(grad, alpha)
    grad_b =maxpool_ob.b_propagation(grad_b)
    grad_b = convolution_ob.b_propagation(grad_b, alpha)

    return ent_loss,acs_preds
```

Payam Sedighiani
21801298
final report of term project eee-485

4.Then it is passed through forward CNN which first makes convolution, then max_pool operation and lastly sends the params to be updated in soft max class.

```
def forward_cnn(img,label):
    m = 0.5
    out_f = convolution_ob.f_propagation((img/255) - m)
    out_f = maxpool_ob.f_propagation(out_f)
    out_f = softmax_ob.f_propagation(out_f)

    # accuracy and entropy_loss

    ent = -np.log(out_f[label])
    accuracy_preds = 1 if np.argmax(out_f) == label else 0
    return out_f,ent,accuracy_preds
```

5.The tuning hyperparameters are learning rate and epochs. At each epoch 100 steps are done and the accuracies are measured. The tuned learning rate was 0.05.

```
for i in range(epochs):
    print("epoch :",i)

    shuffle = np.random.permutation(len(x_train))
    x_train = x_train[shuffle]
    y_train = y_train[shuffle]

    # start training

    loss = 0
    pred_corrects = 0
    epc = 0
    accs = []

    los = []
    for i,(j,l) in enumerate(zip(x_train,y_train)):
        if i % 100 == 0 :
            epc+= 1
            accs.append(pred_corrects)
            los.append(loss/100)
            print("step",i)
            print("loss average:", loss/100)
            print("accuracy:",pred_corrects,"%")
            loss =0
            pred_corrects = 0


        ent_l, acs_pred = train_cnn(j,l)
        loss += ent_l
        pred_corrects+=acs_pred

test_los = []
test_acs = []
test_epochs = 6
for i in range(test_epochs):

    for j,l in zip(x_test,y_test):
        output_f,l_1,acs = forward_cnn(j, l)
        loss += l_1
        pred_corrects += acs

        n_tests = len(x_test)
        test_acs.append(pred_corrects/n_tests)
        test_los.append(1-(loss/(n_tests*100)))
    print('test loss:',1-(loss/(n_tests*100)))
    print('test accuracy:', (pred_corrects/n_tests)*100,'%')
```

Payam Sedighiani
21801298
final report of term project eee-485

CNN Results:

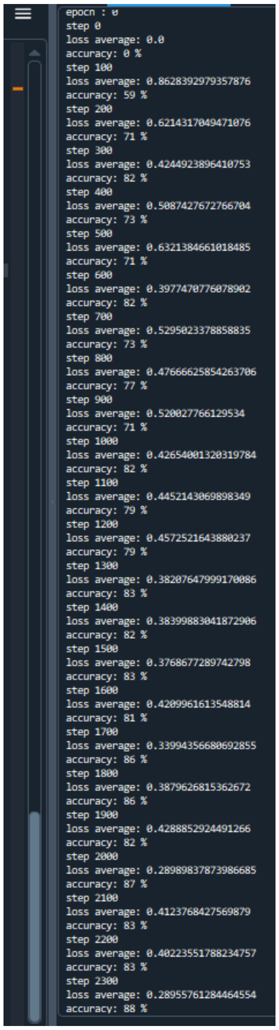


Figure 12: epoch 1 with 15 steps of 100 by 100

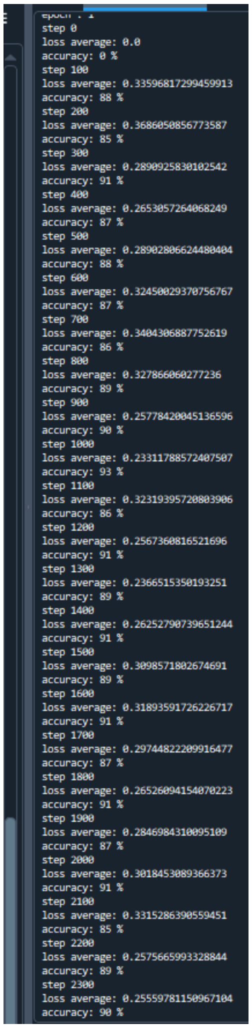


Figure 13: epoch 2 with 15 steps of 100 by 100

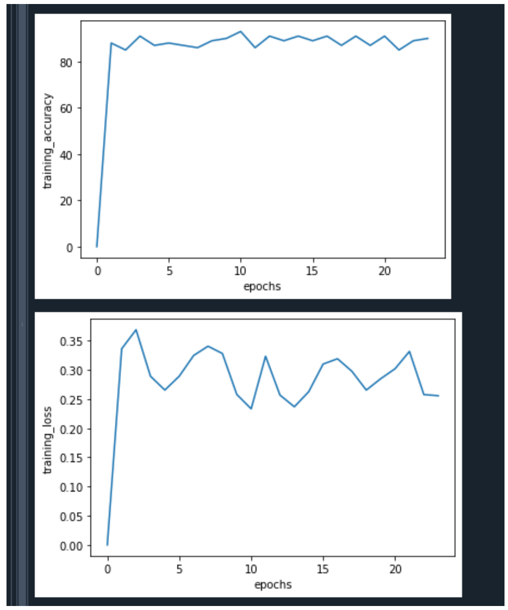


Figure 14: Training accuracy and loss for 15 steps of 100 by 100

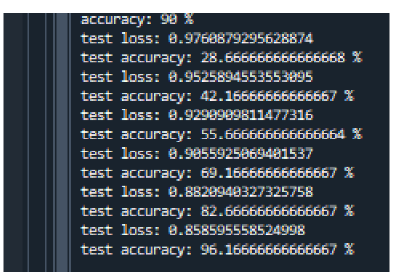


Figure 16: 6 epochs of 6 step of 100 by 100

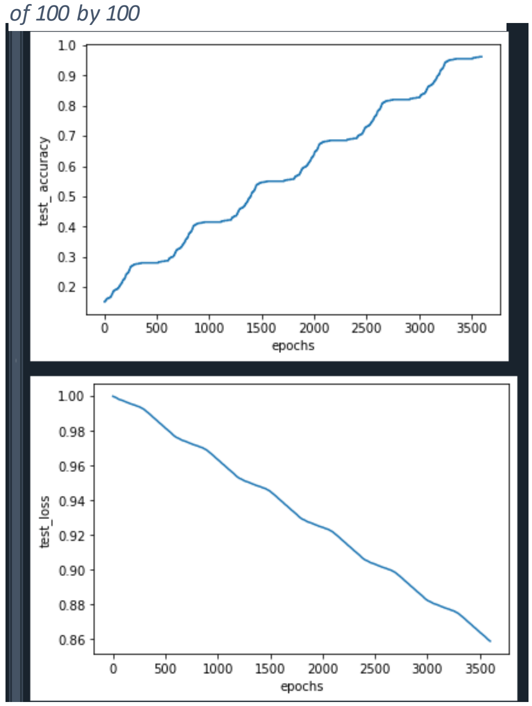


Figure 15:test accuracy and loss for 15 steps of 100 by 100

Payam Sedighiani
21801298
final report of term project eee-485

As it can be seen from the above results, the accuracies are increasing overall and the loss is decreasing for each step. The plots also indicate the mentioned results and we see that the accuracy approaches to 90%. The test accuracy is 96%.

CNN comments:

1. The params are updating correctly and the accuracy are very good.
2. The learning rate and epochs are tuned and the best learning rate found is 0.05. also 0.1 gives a decent value of accuracy at each epoch
3. For epochs the 2400 steps of 100 was enough

Payam Sedighiani
21801298
final report of term project eee-485

References

EE485 and CS464 materials

Dataset link:

https://storage.googleapis.com/kaggle-data-sets/740566/2809126/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com%2F20221217%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20221217T203621Z&X-Goog-Expires=259200&X-Goog-SignedHeaders=host&X-Goog-Signature=18fe2cf37473fc7f98a7dff6e5ccc32538524421e45348041b1e1e2020b847239d937b1404aac9a7c63745ac3bbcc45ea8260f882e99f04f8692bd28edcf7cfe980a92100c6543af5f55057cbfe46b87a68e1d4ee9ad1cbd3ddd67df9a1c24915184a5528ef88221faceae8e7761cee7b00874306806bbfcad20f1d8166baba447f006bee9d962cbe515fd80ba38ba88c4120be406786bda70c8186c57dbe51a91d99c2b1caf10ad7b156bc254ba2e6fc0144d3424125118f4363b76714a5a51fae773039b3dc50a9e4dd665f651581b20e7a53137d5ec8db0492a8fcfe139826982a09f9d12f5bf407b9605755641aaef83dc317a8e6ad291000bf89665afb0

Payam Sedighiani
21801298
final report of term project eee-485

Appendix

KNN:

```
import numpy as np

from matplotlib.image import imread

import cv2



# yes dataset labeling(1)


list_of_pics_yes = list()

for i in range(0,1500):

    train_image_yes= imread('y'+str(i)+'.jpg')

    train_y = cv2.resize(train_image_yes, (25,25))

    if len(train_y.shape) == 3:

        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)

    train_y = np.reshape(train_y,625)

    train_y = np.append(train_y,1)

    list_of_pics_yes.append(train_y)

list_of_pics_yes = np.asarray(list_of_pics_yes)



# no dataset labeling(0)

list_of_pics_no = list()

for i in range(0,1500):

    train_image_no= imread('no'+str(i)+'.jpg')

    train_n = cv2.resize(train_image_no, (25,25))
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    if len(train_n.shape)  == 3:

        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)

    train_n = np.reshape(train_n,625)

    train_n = np.append(train_n,-1)

    list_of_pics_no.append(train_n)

list_of_pics_no = np.asarray(list_of_pics_no)



# conf_matrix


def conf(y_pre, y_tes):

  con_matrix = np.zeros((2, 2), dtype=int)

  for x in range(len(y_pre)):

    if y_pre[x] == -1 and y_tes[x] == -1:

      con_matrix[1][1] += 1  # true negative

    elif y_pre[x] == -1 and y_tes[x] == 1:

      con_matrix[1][0] += 1  # false negative

    elif y_pre[x] == 1 and y_tes[x] == -1:

      con_matrix[0][1] += 1  # false positive

    else:

      con_matrix[0][0] += 1  # true positive


  return np.asarray(con_matrix)



# accuracy


def accuracy(con_matrix):
```

Payam Sedighiani
21801298
final report of term project eee-485

```
  return ((con_matrix[0][0] + con_matrix[1][1]) * 100) / (np.sum(con_matrix))


# cross validation k-fold = 5-fold

a= [0,300,600,900,1200]

b = [300,600,900,1200,1500]




# 4 values of k

k = [3,5,7,11]


for kth in k:

    c = 0

    acs = []

    print("k value is :",kth)


    for l,p in zip(a,b):

        c+=1

        print(c,"fold")


    # y_test dataset labeling(-1)


        list_of_pics_ytest = list_of_pics_yes[l:p]




    # # no_test labeling(-1)


        list_of_pics_ntest = list_of_pics_no[l:p]
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    test_dataset = np.concatenate((list_of_pics_ntest,list_of_pics_ytest),axis  = 0)


  # train dataset and labels dataset


    train_dataset = np.concatenate(((np.concatenate((list_of_pics_no[p:1500],list_of_pics_no[0:l]),axis = 0),np.concatenate((list_of_pics_yes[p:1500],list_of_pics_yes[0:l]),axis  = 0))),axis = 0)
    x_labels = train_dataset[:,-1]
    y_labels = test_dataset[:,-1]


    # finding  distances(euclidine)


    tot = []
    for t in (np.delete(test_dataset,-1,1)):
        ms= []
        for j in (np.delete(train_dataset,-1,1)):
            Sy = np.sqrt(np.sum(np.square(t-j)))
            ms.append(Sy)
            np.asarray(ms)
        tot.append(ms)
    tot = np.asarray(tot)


    # sorting the indicies and getting the relative labels
    sortedd = np.argsort(tot)
    predicted_dist_labels=  []
    for i in sortedd:


        predicted_dist_labels.append(x_labels[i])
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# knn =11 and voting about no or yes based on the the majority
yes_lists =[]
no_lists = []
for f in range(600):
   yes =np.count_nonzero((predicted_dist_labels[f][0:kth])==1)
   yes_lists.append(yes)
   no  =np.count_nonzero((predicted_dist_labels[f][0:kth])==-1)
   no_lists.append(no)
# print(yes_lists)
count_no  = 0
count_yes= 0


y_pre  = []
for b,d in zip(no_lists,yes_lists):
   if b>d:
      count_no+=1
      y_pre.append(-1)
   elif b ==d:
      count_yes+=1
      y_pre.append(-1)
   else:
      count_yes+=1
      y_pre.append(1)




a = conf(y_pre,  y_labels)
```

```
        acs.append(accuracy(a))

        print('confusion  matrix =','\n', a)

        print('accuracy  = ','\n',accuracy(a),'%')


    print("average  acurracy for k =",kth," is ",(sum(acs)/5))
```

LOGISTIC REGRESSION:

```
# -*- coding: utf-8 -*-

"""

Created on Sun Dec 18 23:20:38 2022


@author: payam

"""


import numpy as np

import matplotlib.pyplot as plt

from matplotlib.image import imread

import cv2


list_of_pics_yes = list()

for i in range(0,1500):

    train_image_yes= imread('y'+str(i)+'.jpg')

    train_y = cv2.resize(train_image_yes, (35,35))
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    if len(train_y.shape)  == 3:

        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)

    train_y = np.reshape(train_y,1225)

    train_y = np.append(train_y,1)

    list_of_pics_yes.append(train_y)

list_of_pics_yes = np.asarray(list_of_pics_yes)



# no dataset labeling(0)

list_of_pics_no = list()

for i in range(0,1500):

    train_image_no= imread('no'+str(i)+'.jpg')

    train_n = cv2.resize(train_image_no, (35,35))

    if len(train_n.shape)  == 3:

        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)

    train_n = np.reshape(train_n,1225)

    train_n = np.append(train_n,0)

    list_of_pics_no.append(train_n)

list_of_pics_no = np.asarray(list_of_pics_no)



list_of_pics_ytest = list_of_pics_yes[300:600]



# # no_test labeling(-1)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
list_of_pics_ntest = list_of_pics_no[300:600]



test_dataset = np.concatenate((list_of_pics_ntest,list_of_pics_ytest),axis  = 0)


# train dataset and labels dataset


train_dataset = 
np.concatenate(((np.concatenate((list_of_pics_no[600:1500],list_of_pics_no[0:300]),axis  = 0),

                        np.concatenate((list_of_pics_yes[600:1500],list_of_pics_yes[0:300]),axis  = 0))),axis = 
0)

x_labels = train_dataset[:,-1]

y_labels = test_dataset[:,-1]


train_dataset = train_dataset[:,:-1]


test_dataset = test_dataset[:,:-1]

# normalizing


xtrainnorm  = (train_dataset-(np.amin(train_dataset))+0)/((np.amax(train_dataset))-
(np.amin(train_dataset)))


xtestnorm = (test_dataset-(np.amin(train_dataset))+0)/((np.amax(train_dataset))-
(np.amin(train_dataset)))






# print(xtrainnorm.shape)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# print(xtestnorm.shape)

# print(train_dataset.shape)

# print(test_dataset.shape)



xtrainnorm  = xtrainnorm.T

xtestnorm = xtestnorm.T


ytrain  = x_labels.reshape(1,xtrainnorm.shape[1])

ytest = y_labels.reshape(1,xtestnorm.shape[1])



# print(xtrainnorm.shape)

# print(ytrain.shape)

# print(xtestnorm.shape)

# print(ytest.shape)


def sig(u):

  return  1/(1+np.exp(-u))







# logistic regression


def logistic(xtrain,ytrain,learningrate,epochs,c,x_test,y_test):
```

Payam Sedighiani
21801298
final report of term project eee-485

```
  m = xtrain.shape[1]

  n = xtrain.shape[0]

 # gaussian or not


 # stochastic gradient decent


  if c == 1:

   b = np.random.normal(0,  1)

   w = np.random.normal(0,  1, n)

   w = np.reshape(w,  (n, 1))

  elif c== 2:

   b = np.random.uniform(0,  n)

   w = np.random.uniform(0,  1, n)

   w = np.reshape(w,  (n, 1))


  else:

   w = np.zeros((n,1))

   b = 0

  costlist= []

  acs = []

  js = []

  for i in range(epochs):

   k = np.dot(w.T,xtrain) + b

   j = sig(k)

   # the cost is observed to see whether the algorithm is working or not

   cost = -(1/m)*np.sum(ytrain*np.log(j)+(1-ytrain)*np.log(1-j))

   dw = (1/m)*np.dot(j-ytrain,xtrain.T)

   db = (1/m)*np.sum(j-ytrain)
```

```
      w = w - learningrate*dw.T

      b = b - learningrate*db

      ac,jp = accuracy(x_test,y_test,w,b)

      costlist.append(cost)

      acs.append(ac)

      js.append(jp)

      if(i%(epochs/10))  == 0:

        print(cost)



    return  w, b, costlist,acs,js


def conf(y_pre,  y_tes):

  con_matrix = np.zeros((2,  2), dtype=int)

  for x in range(len(y_pre)):

    if y_pre[x] == 1 and y_tes[x] == 1:

      con_matrix[1][1] += 1  # tn

    elif y_pre[x] == 1 and y_tes[x] == 0:

      con_matrix[1][0] += 1  # fn

    elif y_pre[x] == 0 and y_tes[x] == 1:

      con_matrix[0][1] += 1  # fp

    else:

      con_matrix[0][0] += 1  # tp


  return  np.asarray(con_matrix)


def accuracy(x,y,w,b):

  k = np.dot(w.T,x) + b
```

Payam Sedighiani
21801298
final report of term project eee-485

```
  j = sig(k)


  j = j>0.5

  j = np.array(j,dtype  = 'int64')


  acuracy = (1-np.sum(np.absolute(j-y))/y.shape[0])*100

  return  acuracy,j




# print(y_labels.shape)

epochs = [2000,5000,7000,10000]

learningrate  = [10**-1,10**-2,10**-3,10**-4]


for e in epochs:

    for l in learningrate:


        print("first  10 cost values  for",e,"epochs and",l,"learning  rate  is:")

        w,b,costlist,acs,jp = logistic(xtrainnorm,x_labels,learningrate  = l,epochs = e,c = 0,

                                x_test = xtestnorm,y_test = y_labels)

        jpp = np.asarray(jp)[-1][0]


        c = conf(jpp,y_labels)

        print("confusion  matrix is:")

        print(c)

        print("final  accuracy is :",acs[-1])


        print(l,"learning  rate",e," epochs plots:")
```

```
    plt.plot(np.arange(e),costlist)

    plt.xlabel('epochs')

    plt.ylabel('cost')

    plt.figure()

    plt.plot(np.arange(e),acs)

    plt.xlabel('epochs')

    plt.ylabel('accuracy')

    plt.figure()
```

NAÏVE BAYES:

```
# -*- coding: utf-8 -*-

"""

Created on Mon Dec 19 02:21:51 2022


@author: payam

"""


import numpy as np

from matplotlib.image import imread

import cv2


# test dataset labeling()

# import matplotlib.pyplot as plt

# nos = 110

# yess = 95
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# yes_train labeling(1)


list_of_pics_yes = list()

for i in range(0,1500):

    train_image_yes= imread('y'+str(i)+'.jpg')

    train_y = cv2.resize(train_image_yes, (20,20))

    if len(train_y.shape) == 3:

        train_y = cv2.cvtColor(train_y, cv2.COLOR_BGR2GRAY)

    train_y = train_y/255

    train_y = np.reshape(train_y,400)

    train_y = np.append(train_y,1)

    list_of_pics_yes.append(train_y)

list_of_pics_yes = np.asarray(list_of_pics_yes)



# no_train labeling(2)

list_of_pics_no = list()

for i in range(0,1500):

    train_image_no= imread('no'+str(i)+'.jpg')

    train_n = cv2.resize(train_image_no, (20,20))

    if len(train_n.shape) == 3:

        train_n = cv2.cvtColor(train_n, cv2.COLOR_BGR2GRAY)

    train_n = train_n/ 255

    train_n = np.reshape(train_n,400)

    train_n = np.append(train_n,2)

    list_of_pics_no.append(train_n)

list_of_pics_no = np.asarray(list_of_pics_no)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# train dataset and labels dataset

a= [0,300,600,900,1200]

b = [300,600,900,1200,1500]


acs_a = []

for l,p in zip(a,b):

    train_dataset = np.concatenate(((np.concatenate((list_of_pics_no[p:1500],list_of_pics_no[0:l]),

                          axis = 0),np.concatenate((list_of_pics_yes[p:1500],list_of_pics_yes[0:l]),axis  = 0))),axis
= 0)

    train_dataset_withoutlabels  = np.delete(train_dataset,-1,1)

    # print(train_dataset_withoutlabels.shape)

    list_of_pics_ytest = list_of_pics_yes[l:p]


    # # no_test labeling(-1)

    list_of_pics_ntest = list_of_pics_no[l:p]

    test_dataset = np.concatenate((list_of_pics_ntest,list_of_pics_ytest),axis  = 0)

    test_dataset_withoutlabels  = np.delete(test_dataset,-1,1)

    # print(test_dataset)

    # print(test_dataset_withoutlabels)

    # train labes

    train_labels = train_dataset[:,-1]

    train_labels_yes  =np.count_nonzero((train_labels)==1)

    train_labels_no   =np.count_nonzero((train_labels)==2)


    # test labels

    test_labels = test_dataset[:,-1]
```

Payam Sedighiani
21801298
final report of term project eee-485

```
test_labels_yes =np.count_nonzero((test_labels)==1)

test_labels_no  =np.count_nonzero((test_labels)==2)





# probablity of classes


prob_class =[train_labels_yes/3000,train_labels_no/3000]



# print(test_dataset_withoutlabels.shape)


# tetas for class 1

a =1


T1 = 0

TT1 = []

for k in range(400):

  T1 = 0

  for i, j in zip(train_dataset_withoutlabels[:,  [k]], train_labels):

    if j == 1 and i != 0:

      T1+=1

  TT1.append(T1)

TT1 = np.asarray(TT1)

sigT1 = np.array(sum(TT1))

TETA1 = np.log(TT1/sigT1)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
TETA1a = np.log((TT1+a)/(sigT1  +a*400))

# print(TT1)

# print(sigT1)

# print(TETA1)


# print('ssssssssss')

# tetas for class 2

T2 = 0

TT2 = []

for k in range(400):

  T2 = 0

  for i, j in zip(train_dataset_withoutlabels[:,  [k]], train_labels):

    if j == 2 and  i != 0:

      T2+=1

  TT2.append(T2)

TT2 = np.asarray(TT2)

sigT2 = np.array(sum(TT2))

TETA2 = np.log(TT2/sigT2)



TETA2a = np.log((TT2+a)/(sigT2  +a*400))

# print(TT2)

# print(sigT2)

# print(TETA2.shape)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# finding predictions


y1 = (prob_class[0]) + np.dot(test_dataset_withoutlabels , TETA1a)

y1 = np.exp(y1)

y2 = (prob_class[1]) + np.dot(test_dataset_withoutlabels , TETA2a)

y2 = np.exp(y2)



# labeling predictions

nos =0

yes =0

y_pre = []

for o in range(600):

    if y2[o] > y1[o]:

        nos+=1

        y_pre.append(2)

    else:

        yes+=1

        y_pre.append(1)



def conf(y_pre, y_tes):

  con_matrix = np.zeros((2, 2), dtype=int)

  for x in range(len(y_pre)):

    if y_pre[x] == 2 and y_tes[x] == 2:

      con_matrix[1][1] += 1  # tn

    elif y_pre[x] == 2 and y_tes[x] == 1:

      con_matrix[1][0] += 1  # fn
```

```
      elif y_pre[x] == 1 and y_tes[x] == 2:

        con_matrix[0][1] += 1  # fp

      else:

        con_matrix[0][0] += 1  # tp


    return np.asarray(con_matrix)

  # accuracy


  def accuracy(con_matrix):

    return ((con_matrix[0][0] + con_matrix[1][1]) * 100) / (np.sum(con_matrix))


  a =conf(y_pre,test_labels)

  print('confusion matrix =','\n', a)

  print('accuracy = ','\n',accuracy(a),'%')

  acs_a.append(accuracy(a))


print("the  average accuracy for 5-fold cross validation is",sum(acs_a)/5)
```

CNN:

```
# -*- coding: utf-8 -*-

"""

Created on Sun Dec 18 13:09:57 2022


@author: payam

"""

import numpy as np

from matplotlib.image import imread

import cv2
```

Payam Sedighiani
21801298
final report of term project eee-485

```
import matplotlib.pyplot as plt



list_of_pics_yes = []

yes_labels = []


for i in range(0,1500):

    train_image_yes= cv2.imread('y'+str(i)+'.jpg',0)

    train_y = cv2.resize(train_image_yes, (30,30))

    list_of_pics_yes.append(train_y)

    yes_labels.append(1)

list_of_pics_yes = np.asarray(list_of_pics_yes)

yes_labels = np.asarray(yes_labels)


# no dataset labeling(0)

list_of_pics_no = []

no_labels = []

for i in range(0,1500):

    train_image_no= cv2.imread('no'+str(i)+'.jpg',0)

    train_n = cv2.resize(train_image_no, (30,30))

    list_of_pics_no.append(train_n)

    no_labels.append(0)

list_of_pics_no = np.asarray(list_of_pics_no)

no_labels = np.asarray(no_labels)
```

Payam Sedighiani
21801298
final report of term project eee-485

```
# k-fold = 5-fold





# one hot encoding


# x_labels = train_dataset[:,-1]

# y_labels = test_dataset[:,-1]





class conv:

    def __init__(self,n_filters,  s_filter):

        self.n_filters = n_filters

        self.s_filter = s_filter

        self.c_filter = np.random.randn(n_filters,s_filter,s_filter  )/(s_filter  * s_filter)


    # generating buffer for saving patches

    def patch(self,img):


        h,w = img.shape

        self.img = img

        for i in range(h-(self.s_filter)  +1):

            for j in range(w-(self.s_filter)  +1):

                img_p = img[i: (i+self.s_filter),  j : (j + self.s_filter)]

                yield img_p, i, j
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    def f_propagation(self,img):
        h,w = img.shape
        output_conv = np.zeros((h - self.s_filter+1,w-self.s_filter+1,self.n_filters))
        for img_pch,i,j in self.patch(img):
            output_conv[i,j] = np.sum(img_pch *self.c_filter, axis = (1,2))

        return output_conv



    def b_propagation(self,out_dl, alpha):

        params_df = np.zeros(self.c_filter.shape)

        for img_pch,i,j in self.patch(self.img):
            for l in range(self.n_filters):
                params_df[l] += img_pch*out_dl[i,j,l]

        self.c_filter -= alpha * params_df
        return params_df




class max_pool:

    def __init__(self,s_filter):
        self.s_filter =s_filter
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    def patch(self,img):

        new_h = img.shape[0]//self.s_filter

        new_w = img.shape[1]//self.s_filter

        self.img = img


        # extraction of image pacthes

        for i in range(new_h):

            for j in range(new_w):

                img_p = img[(i*self.s_filter): (i *self.s_filter + self.s_filter),

                        (j*self.s_filter): (j *self.s_filter + self.s_filter)]

                yield img_p,i,j


    def f_propagation(self,img):

        h,w, n_filters = img.shape

        output_fp = np.zeros((h//self.s_filter,w  // self.s_filter, n_filters))


        for img_pch,i,j in self.patch(img):

            output_fp[i,j] = np.amax(img_pch,axis  = (0,1))


        return  output_fp


    def b_propagation(self,out_dl):

        pool_dl = np.zeros(self.img.shape)

        for img_pch,i,j in self.patch(self.img):

            h,w,n_filters = img_pch.shape

            val_max = np.amax(img_pch,axis  = (0,1))


            for l in range(h):
```

```
            for a in range(w):

                for b in range(n_filters):

                    if img_pch[l,a,b] == val_max[b]:

                        pool_dl[i *self.s_filter + l, j *self.s_filter+a,b] = out_dl[i,j,b]

        return pool_dl




class soft_max:

    def __init__(self,x_n,s_n):

        self.w = np.random.randn(x_n,s_n)/  x_n

        self.b = np.zeros(s_n)


    def f_propagation(self,img):


        self.img_first = img.shape

        # flatteining the cubes

        img_changed = img.flatten()

        self.changed_x = img_changed

        val_out = np.dot(img_changed,self.w)+self.b

        self.y = val_out

        out_ex = np.exp(val_out)

        # probability output

        return out_ex/np.sum(out_ex,axis=0)


    def b_propagation(self,out_dl,alpha):


        for i,g in enumerate(out_dl):
```

```
            if g == 0:

                continue

            trequ = np.exp(self.y)

            tot = np.sum(trequ)


            # output(z) gradient


            dydz = - trequ[i] * trequ/(tot**2)

            dydz[i] = trequ[i]*(tot-trequ[i])/(tot**2)


            # weight and bias and input gradients with respect to tot


            dzdw = self.changed_x

            dzdb = 1

            dzdx = self.w


            # loss gradient

            dldz = g *dydz


            # loss gradient with respect to bias,weights,input


            dldw = dzdw[np.newaxis].T @ dldz[np.newaxis]

            dldb = dldz * dzdb

            dldx = dzdx @ dldz

        self.w -= alpha *dldw

        self.b -= alpha *dldb


        return dldx.reshape(self.img_first)
```

Payam Sedighiani
21801298
final report of term project eee-485

y_test dataset labeling(-1)

list_of_pics_ytest = list_of_pics_yes[300:600]

no_test labeling(-1)

list_of_pics_ntest = list_of_pics_no[300:600]

test_dataset = np.concatenate((list_of_pics_ntest,list_of_pics_ytest),axis = 0)

x_test = test_dataset

y_test = np.concatenate((yes_labels[300:600],no_labels[300:600]),axis = 0)

train dataset and labels dataset

train_dataset =
np.concatenate(((np.concatenate((list_of_pics_no[600:1500],list_of_pics_no[0:300]),axis =
0),np.concatenate((list_of_pics_yes[600:1500],list_of_pics_yes[0:300]),axis = 0))),axis = 0)

x_train = train_dataset

y_train = train_dataset = np.concatenate(((np.concatenate((no_labels[600:1500],no_labels[0:300]),axis
= 0),np.concatenate((yes_labels[600:1500],yes_labels[0:300]),axis = 0))),axis = 0)

Payam Sedighiani
21801298
final report of term project eee-485

```
convolution_ob = conv(10,3)

maxpool_ob = max_pool(2)

softmax_ob = soft_max(14*14*10,10)
```

```
def forward_cnn(img,label):

    m = 0.5

    out_f = convolution_ob.f_propagation((img/255)  - m)

    out_f = maxpool_ob.f_propagation(out_f)

    out_f = softmax_ob.f_propagation(out_f)


    # accuracy and entropy_loss


    ent = -np.log(out_f[label])

    accuracy_preds = 1 if np.argmax(out_f) == label else 0

    return out_f,ent,accuracy_preds
```

```
def train_cnn(img,label,alpha  = 0.05):

    # forward propagation
```

```
    output_f, ent_loss, acs_preds = forward_cnn(img,label)

    grad = np.zeros(10)

    grad[label] = -1/output_f[label]


    # back propagation


    grad_b =softmax_ob.b_propagation(grad,  alpha)

    grad_b =maxpool_ob.b_propagation(grad_b)

    grad_b = convolution_ob.b_propagation(grad_b,  alpha)


    return ent_loss,acs_preds





epochs = 2







for i in range(epochs):

    print("epoch :",i)


    shuffle = np.random.permutation(len(x_train))

    x_train = x_train[shuffle]

    y_train = y_train[shuffle]


    # start training
```

Payam Sedighiani
21801298
final report of term project eee-485

```
    loss = 0

    pred_corrects = 0

    epc = 0

    accs = []


    los = []

    for i,(j,l) in enumerate(zip(x_train,y_train)):

        if i % 100 == 0 :

            epc+= 1

            accs.append(pred_corrects)

            los.append(loss/100)

            print("step",i)

            print("loss average:", loss/100)

            print("accuracy:",pred_corrects,"%")

            loss =0

            pred_corrects = 0




        ent_l, acs_pred = train_cnn(j,l)

        loss += ent_l

        pred_corrects+=acs_pred


test_los = []

test_acs = []

test_epochs = 6

for i in range(test_epochs):


    for j,l in zip(x_test,y_test):
```

```
        output_f,l_1,acs = forward_cnn(j,  l)

        loss += l_1

        pred_corrects  += acs


        n_tests  = len(x_test)

        test_acs.append(pred_corrects/n_tests)

        test_los.append(1-(loss/(n_tests*100)))

    print('test  loss:',1-(loss/(n_tests*100)))

    print('test  accuracy:', (pred_corrects/n_tests)*100,'%')


c= 3000

plt.plot(np.arange(epc),accs)

plt.xlabel('epochs')

plt.ylabel('training_accuracy')

plt.figure()

plt.plot(np.arange(epc),los)

plt.xlabel('epochs')

plt.ylabel('training_loss')

plt.figure()

plt.plot(test_acs)

plt.xlabel('epochs')

plt.ylabel('test_  accuracy')

plt.figure()

plt.plot(test_los)

plt.xlabel('epochs')

plt.ylabel('test_loss')

plt.figure()
```