sudo code

```
# define: 1) each face image would be 60*70 (X*Y)
          2) each digit image would be 28*28 (X*Y)
          3) we divide each face image into 42*(10*10) small regions (call it o)
          4) labels[i]: 1 or 0, indicates face or not face for the image[i]
          5) data_regions[i][j]: integer, indicates number of '#' in the region j(0-41) for
                 the image[i]
```

perceptron sudo code:

```
perceptron_f(labels, data_region,  type):
# w is the weight for corresponding g value
# initially we assign random number to each w
w = []
for(i in range 42)
        w[i] = uniform(-1, 1)


# g is a list holds number of '#' in a given region of an image
g = []

# f(x), indicator of our prediction. < 0 means our model predict it is not a face while >= 0
# means our model predict it is face
f = 0

# bias
bias = uniform(-1, 1)

# loop
for (i in range (every single image in the percentage of data we want to use) )
        # the range would just be len(labels)

        # the ith image
        g = data_region[i]

        # a loop multiplying each weight with corresponding o value
        for (j in range 42)
                f = f + w[j] * g[j]

        # add bias after the loop
        f = f + bias

        # if we predict it right, move on. Otherwise do the penalty to w
        if f >= 0 and label[i] == 0
                for(k in range 42)
                        w[k] = w[k] - g[k]
```

```
                    bias = bias - 1

        elif f < 0 and label[i] == 1
                for(k in range 42)
                    w[k] = w[k] + g[k]
                    bias = bias + 1

return
```

end of perceptron_f

perceptron_i(labels, data_region, type):

```
# w[i]: the list weight for digit i
# w[i][j]: the jth weight for digit i
for (i = 0; i < 9; i++)
{
        for (j = 0; j < num_of_o; j++)
        {
                w[i][j] = random(-1, 0, 1)
        }
}

# f[]: the list of fs, 0-9
for(i = 0; i < 9; i++)
{
        f[i] = 0
}

# bias: the list of bias, 0-9
for(i = 0; i < 9; i++)
{
        bias[i] = random(-1, 0, 1)
}

# loop
for (i = 0; i < len(labels); i++)
{

        # the ith image
        g = data_region[i]

        # a loop multiplying each weight with corresponding o value
        for (k = 0; k < 9; k++)
        {
                for (j = 0; j < num_of_o; j++)
                {
                        f[k] = f[k] + w[k][j] * g[j]
```

```
                }

                # add bias after the loop
                f[k] = f[k] + bias[k]

                # see which digit our model predicts
                largest = MAX(f[0], …, f[9]);
                for (p = 0; p < 9; p++)
                {
                        if (f[p] == largest)
                                prediction = p
                                break
                }

                if (prediction != labels[i])
                {
                        for (z = 0; z < num_of_o; z++)
                        {
                                w[prediction][z] = w[prediction][z] – g[z]
                                bias[prediction] = bias[prediction] – 1

                                w[labels[i]][z] = w[labels[i]][z] + g[z]
                                bias[labels[i]] = bias[labels[i]] + 1
                        }
                }

        }

}

# what we are going to use for test_file is 'w[][]'
# test code here
return
```

end of perceptron_i

main:

```
/* common code starts */
# gather input indicators
type, percent, algorithm

# gather the wanted data set
labels = training_labels(type, float(percent))
data_regions = training_data(type, float(percent))
/* common code ends */


/* perceptron starts */
if (algorithm == perceptron)
	if (type == f)
		perceptron_f(labels, data_region, type)
	else
		perceptron_d(labels, data_region, type)

# what we are going to use to run the test_file is the 'w[]'
# some code here to run the test_file without changing 'w[]' and report the accuracy
/* perceptron ends */


/* naïve_bayes starts */
if (algorithm == naïve_bayes)
# calculate p(image== face) and p(image == not face)
num_face = 0
for (i in range (len(labels))) # this is correct only if len(labels) give the real length(not n-1)
	if (labels[i] == 1)
	   num_face = num_face + 1

num_Nface = len(labels) – num_face

# go into the dataset and construct the probability table
# we divided each face image into 42*(10*10) small regions that is how we get these below
tabel_face = [ [0 for x in range(100)] for y in range(42)]
tabel_Nface = [ [0 for x in range(100)] for y in range(42)]


# data_region[i][j]. Fill in the data for the table
# outer loop is the jth region, inner loop is the ith image
```

```
# get the 0th region for all the image, then go back and get the 1st and so on
/* below here is java code... I do not have the confidence to do the 2d array stuff in
python*/
int count  = 0
for (j = 0; j < 42; j++)
{

   for (i = 0; i < len(labels), i++)
   {

          count = data_region[i][j]

          if (labels[i] == 1)
          {
                    tabel_face[count][j]++
          }
          else
          {
                    tabel_Nface[count][j]++
          }

   }

}

# traverse both tables, divide each element by number of face/Nface images
# if an index is zero, we replace it with 0.001
for (i = 0; i < 100; i++)
{

   for (j = 0; j < 42; j++)
   {

          If (tabel_face[i][j] == 0)
          {
                    tabel_face[i][j] = 0.001
          }
          else
          {
                    tabel_face[i][j] = tabel_face[i][j] / num_face
          }

          If(tabel_Nface[i][j] == 0)
          {
                    tabel_Nface[i][j] = 0.001
          }
          else
```

```
        {
                tabel_Nface[i][j] = tabel_Nface[i][j] / num_Nface
        }

    }

}


# load the test_file, test and report.etc
/* naïve bayes ends */
```

end of main