STA2104 - Statistical Methods for Machine Learning / Homework 2

Luis Alvaro Correia - No. 1006508566

February 16th 2021

Question 1

Multi-Class Logistic Regression Classifier

Item (a)

Solution.

Considering the *Multi-Class Logistic Regression* model, we have the following:

1. $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N]^T \in \mathbb{R}^N \times \{0,1\}^K$ is the matrix containing the *image labels*;
2. $\boldsymbol{X} = [\boldsymbol{x}_1^T, \ldots, \boldsymbol{x}_N^T] \in \mathbb{R}^N \times \mathbb{R}^D$ is the matrix containing the *flattened images*;
3. $\boldsymbol{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K]^T \in \mathbb{R}^D \times \mathbb{R}^K$ is the matrix containing the *weights of each class*;

The parameters of our model are then given by $\boldsymbol{W}$, which has dimension $D \times K$, then *the number of parameters of our model is* 10 *(=* K*) parameters of dimension* 784 *(=* D*) each.* Total: $7,840$ parameters.

Item (b)

Solution.

For our multi-class classification problem, we know from [2] that the posterior probabilities for each class $\mathcal{C}_k$, with $k = 1, \ldots, K$ are given by the softmax transformation of linear functions for a given image $\boldsymbol{x}$, so that:

$$p(\mathcal{C}_k|\boldsymbol{x}) = y_k(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^T \cdot \boldsymbol{x})}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_j^T \cdot \boldsymbol{x})} \tag{1}$$

From (1), we consider the *1-to-K* coding scheme for the target vector $\boldsymbol{t}_n$, for a given image $\boldsymbol{x}_n$ belonging to class $\mathcal{C}_k$.

As consequence, the log-likelihood function of our model can be expressed by:

$$\begin{aligned} p(\boldsymbol{T}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) &= \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\boldsymbol{x}_n)^{t_{nk}} \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} y_k(\boldsymbol{x}_n)^{t_{nk}} \\ &= \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \end{aligned}$$

$$\implies p(\boldsymbol{T}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \tag{2}$$

... where $y_{nk} = y_k(\boldsymbol{x}_n)$ and $\boldsymbol{T}$ is a $N \times K$ matrix of target values t_{nk}.

Taking the negative logarithm and using (2), we have the *cross-entropy* function given by:

$$\begin{aligned} E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) &= -\ln p(\boldsymbol{T}|\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) \\ &= -\ln \Big[\prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}} \Big] \\ &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \ln y_{nk} \end{aligned}$$

$$\implies E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \ln y_{nk} \tag{3}$$

Now taking the gradient w.r.t. each $\boldsymbol{w}_j$, with $j = 1, \ldots, K$ we have that the minimization problem over the cross-entropy function can be solved over the gradient $\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)$ using (3) as follows:

$$
\begin{aligned}
\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) &= \nabla_{\boldsymbol{w}_j} \Big[- \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \ln y_{nk} \Big] \\
&= \nabla_{\boldsymbol{w}_j} \left[- \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \ln \left(\frac{\exp(\boldsymbol{w}_k^T \cdot \boldsymbol{x_n})}{\sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n})} \right) \right] \\
&= \nabla_{\boldsymbol{w}_j} \left[- \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \Big(\boldsymbol{w}_k^T \cdot \boldsymbol{x_n} - \ln \sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n}) \Big) \right] \\
&= - \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \Big\{ \underbrace{\nabla_{\boldsymbol{w}_j} (\boldsymbol{w}_k^T \cdot \boldsymbol{x_n})}_{\mathbf{A}} - \underbrace{\nabla_{\boldsymbol{w}_j} \Big[\ln \sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n}) \Big]}_{\mathbf{B}} \Big\}
\end{aligned}
$$

We have that **A** can be represented as:

$$
\begin{aligned}
\mathbf{A} &= \nabla_{\boldsymbol{w}_j} (\boldsymbol{w}_k^T \cdot \boldsymbol{x_n}) \\
&= \mathbb{1}(j, k) \boldsymbol{x}_n, \text{ where the indicator function is 1 if } j = k, \text{ 0 otherwise.}
\end{aligned}
$$

... and **B** can be reduced to:

$$
\begin{aligned}
\mathbf{B} &= \nabla_{\boldsymbol{w}_j} \Big[\ln \sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n}) \Big] \\
&= \frac{1}{\sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n})} \times \nabla_{\boldsymbol{w}_j} \Big[\sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n}) \Big] \\
&= \frac{1}{\sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n})} \times \exp(\boldsymbol{w}_j^T \cdot \boldsymbol{x_n}) \boldsymbol{x}_n \\
&= \frac{\exp(\boldsymbol{w}_j^T \cdot \boldsymbol{x_n})}{\sum_{i=1}^{K} \exp(\boldsymbol{w}_i^T \cdot \boldsymbol{x_n})} \times \boldsymbol{x}_n \\
&= y_{nj} \boldsymbol{x}_n
\end{aligned}
$$

Plugging **A** and **B** in our gradient equation we have:

$$\begin{aligned}
\nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) &= -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \times \Big[\mathbb{1}(j,k)\boldsymbol{x}_n - y_{nj}\boldsymbol{x}_n\Big] \\
&= -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\mathbb{1}(j,k)\boldsymbol{x}_n + \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}y_{nj}\boldsymbol{x}_n \\
&= -\sum_{n=1}^{N} t_{nj}\boldsymbol{x}_n + \sum_{n=1}^{N} y_{nj}\boldsymbol{x}_n \\
&= \sum_{n=1}^{N} \big(y_{nj} - t_{nj}\big)\boldsymbol{x}_n
\end{aligned}$$

Finally, the gradient of our cross-entropy function is then reduced to:

$$\implies \nabla_{\boldsymbol{w}_j} E(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K) = \sum_{n=1}^{N} \big(y_{nj} - t_{nj}\big)\boldsymbol{x}_n, \text{ for } j = 1, \ldots, K \tag{4}$$

Item (c)

Solution.

For processing the **Multi-class Logistic Regression** it was used the entire `MNIST` database, i.e., $60,000$ *images for training* process and $10,000$ *images for testing* process.

The accuracy of each process and partial[1] listing of classification errors - which includes the *image number* (`Img No.`), *correct images's label* (`Label`) and *model's classification* (`Classif.`) - are as follows:

```
------ Processing Summary (Multi-Class Logistic Regression) -----------

Training accuracy is 89.43%
Test accuracy is 90.05%

------ Listing of Errors from TRAINING procedure (60000 Samples) ------

>>> Total No. of Errors:   6341

>>> Note: Only first 120 will be printed due to limitation on Crowdmark
```

[1]Only first 120 will be printed due to limitation on *Crowdmark* to manage high no. of pages in uploaded PDF format.

to manage high no. of pages in uploaded PDF format.

	Img No.	Label	Classif.
0	24	1	6
1	28	2	7
2	33	9	7
3	48	9	5
4	70	1	2
5	80	9	0
6	109	2	6
7	120	2	7
8	132	5	3
9	134	1	5
10	138	5	0
11	148	7	4
12	158	7	9
13	172	9	4
14	173	5	1
15	178	2	0
16	180	2	1
17	181	3	8
18	182	5	3
19	207	3	7
20	212	7	1
21	224	1	7
22	228	3	5
23	232	0	6
24	240	8	1
25	244	5	8
26	246	0	5
27	254	3	5
28	256	6	1
29	262	2	5
30	264	9	4
31	268	2	7
32	278	5	6
33	304	9	7
34	340	7	9
35	346	9	3
36	370	7	1
37	374	9	4
38	391	2	3
39	404	8	3
40	414	4	6
41	418	8	2

42	420	5	9
43	444	2	1
44	446	7	0
45	459	3	9
46	470	1	8
47	494	6	0
48	497	7	2
49	500	3	7
50	501	9	8
51	502	5	8
52	509	3	9
53	514	5	3
54	528	8	5
55	529	9	4
56	533	1	8
57	544	5	3
58	554	5	8
59	558	2	8
60	584	2	6
61	588	2	1
62	602	8	3
63	610	5	9
64	611	7	1
65	614	5	4
66	626	9	4
67	627	8	6
68	630	6	8
69	631	9	8
70	635	5	2
71	644	7	1
72	646	2	1
73	659	3	9
74	664	2	7
75	665	6	4
76	670	3	2
77	690	5	8
78	720	8	1
79	736	8	3
80	748	5	3
81	749	4	9
82	760	3	5
83	778	5	4
84	784	8	3
85	788	9	6
86	801	2	4

```
87       828      4         2
88       832      5         3
89       836      5         8
90       846      6         8
91       850      4         6
92       854      2         9
93       861      3         5
94       863      2         4
95       864      9         7
96       866      2         3
97       870      1         3
98       879      5         8
99       884      7         2
100      886      5         2
101      892      6         1
102      899      5         8
103      902      9         0
104      903      0         5
105      916      5         6
106      924      5         3
107      933      7         4
108      946      2         8
109      966      3         6
110      968      7         9
111      991      4         5
112      995      7         4
113     1007      3         9
114     1019      7         4
115     1024      5         4
116     1029      0         5
117     1030      4         6
118     1032      5         3
119     1047      2         7

------ Listing of Errors from TESTING procedure (10000 Samples)------

>>> Total No. of Errors:    995

>>> Note: Only first 120 will be printed due to limitation on Crowdmark
          to manage high no. of pages in uploaded PDF format.

     Img No.  Label  Classif.
0          8      5         6
1         33      4         6
2         46      1         3
```

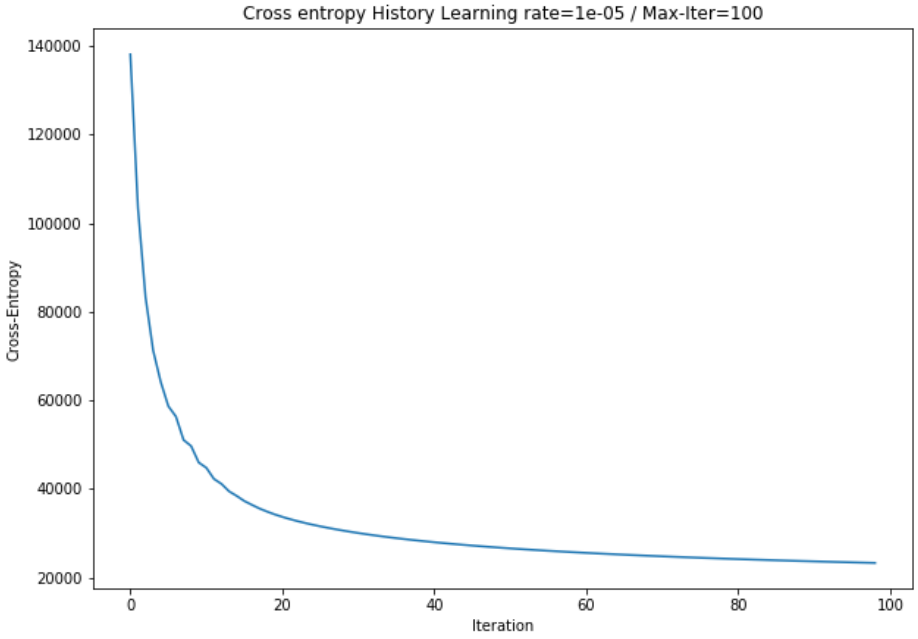
3	59	5	7
4	73	9	7
5	77	2	7
6	111	7	1
7	124	7	4
8	149	2	9
9	151	9	2
10	172	2	3
11	193	9	3
12	195	3	8
13	217	6	5
14	233	8	7
15	241	9	8
16	243	7	4
17	245	3	6
18	247	4	2
19	259	6	0
20	261	5	1
21	290	8	4
22	295	4	9
23	300	4	1
24	307	7	9
25	313	3	5
26	318	2	3
27	320	9	1
28	321	2	7
29	340	5	3
30	341	6	4
31	349	3	7
32	352	5	0
33	362	2	7
34	363	2	1
35	376	4	9
36	381	3	7
37	389	9	4
38	403	8	9
39	435	8	7
40	444	2	8
41	445	6	0
42	448	9	8
43	449	3	5
44	457	6	5
45	464	3	7
46	468	7	2
47	478	5	8

48	479	9	3
49	483	5	8
50	495	8	0
51	502	5	3
52	507	3	5
53	511	4	8
54	515	3	8
55	516	2	4
56	523	1	8
57	524	3	5
58	531	3	6
59	536	2	1
60	543	8	4
61	551	7	1
62	553	8	6
63	565	4	9
64	569	3	5
65	578	3	8
66	582	8	2
67	591	8	3
68	595	3	5
69	605	7	9
70	610	4	6
71	613	2	8
72	619	1	8
73	624	2	1
74	627	9	4
75	628	3	9
76	629	2	6
77	638	5	7
78	658	7	4
79	659	2	7
80	667	7	1
81	684	7	2
82	689	7	9
83	691	8	4
84	692	5	7
85	707	4	9
86	717	0	6
87	720	5	8
88	728	2	8
89	738	2	8
90	740	4	9
91	741	2	8
92	760	4	9

93	791	5	9
94	800	8	5
95	839	8	3
96	844	8	7
97	846	7	9
98	857	5	3
99	881	4	9
100	882	9	7
101	898	7	2
102	900	1	3
103	924	2	7
104	930	7	1
105	938	3	5
106	939	2	8
107	944	3	8
108	947	8	9
109	950	7	2
110	956	1	6
111	959	4	9
112	960	7	1
113	965	6	0
114	975	2	3
115	982	3	8
116	992	9	4
117	994	1	5
118	998	8	9
119	1012	7	9

The training curve represented by the accuracy of the model during the iterative *gradient descent* process is shown below.

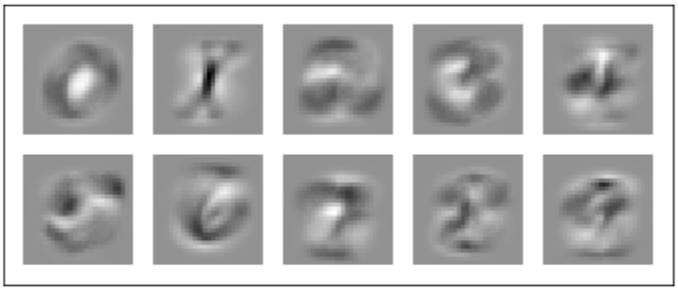
... and the cross-entropy variation shows the convergence of the model.



Item (d)

Solution.

The image containing each class weight $\boldsymbol{w}_j$ for $j = 1, \ldots, K$ is as follows:



Implementation details can be verified in section **Code Listing**, at the end of this question.

Code-Listing

```
from __future__ import absolute_import
from __future__ import print_function
from future.standard_library import install_aliases
install_aliases()
import numpy as np
import pandas as pd
from scipy.special import logsumexp
import os
import gzip
import struct
import array
import matplotlib.pyplot as plt
import matplotlib.image
from urllib.request import urlretrieve

def download(url, filename):
    if not os.path.exists('data'):
        os.makedirs('data')
    out_file = os.path.join('data', filename)
    if not os.path.isfile(out_file):
        urlretrieve(url, out_file)


def mnist():
    base_url = 'http://yann.lecun.com/exdb/mnist/'

    def parse_labels(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data = struct.unpack(">II", fh.read(8))
            return np.array(array.array("B", fh.read()), dtype=np.uint8)

    def parse_images(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data, rows, cols = struct.unpack(">IIII", fh.read(16))
            return np.array(array.array("B", fh.read()),
                dtype=np.uint8).reshape(num_data, rows, cols)

    for filename in ['train-images-idx3-ubyte.gz',
                     'train-labels-idx1-ubyte.gz',
                     't10k-images-idx3-ubyte.gz',
                     't10k-labels-idx1-ubyte.gz']:
        download(base_url + filename, filename)

    train_images = parse_images('data/train-images-idx3-ubyte.gz')
    train_labels = parse_labels('data/train-labels-idx1-ubyte.gz')
```

```
    test_images = parse_images('data/t10k-images-idx3-ubyte.gz')
    test_labels = parse_labels('data/t10k-labels-idx1-ubyte.gz')

    return train_images, train_labels, test_images[:10000], test_labels[:10000]


def load_mnist(N_data=None):
    partial_flatten = lambda x: np.reshape(x, (x.shape[0], np.prod(x.shape[1:])))
    one_hot = lambda x, k: np.array(x[:, None] == np.arange(k)[None, :],
        dtype=int)
    train_images, train_labels, test_images, test_labels = mnist()
    train_images = (partial_flatten(train_images) / 255.0 > .5).astype(float)
    test_images = (partial_flatten(test_images) / 255.0 > .5).astype(float)
    K_data = 10
    train_labels = one_hot(train_labels, K_data)
    test_labels = one_hot(test_labels, K_data)
    if N_data is not None:
        train_images = train_images[:N_data, :]
        train_labels = train_labels[:N_data, :]

    return train_images, train_labels, test_images, test_labels


def plot_images(images, ax, ims_per_row=5, padding=5, digit_dimensions=(28, 28),
                cmap=matplotlib.cm.binary, vmin=None, vmax=None):
    """Images should be a (N_images x pixels) matrix."""
    N_images = images.shape[0]
    N_rows = np.int32(np.ceil(float(N_images) / ims_per_row))
    pad_value = np.min(images.ravel())
    concat_images = np.full(((digit_dimensions[0] + padding) * N_rows + padding,
                             (digit_dimensions[1] + padding) * ims_per_row +
                                padding), pad_value)
    for i in range(N_images):
        cur_image = np.reshape(images[i, :], digit_dimensions)
        row_ix = i // ims_per_row
        col_ix = i % ims_per_row
        row_start = padding + (padding + digit_dimensions[0]) * row_ix
        col_start = padding + (padding + digit_dimensions[1]) * col_ix
        concat_images[row_start: row_start + digit_dimensions[0],
                      col_start: col_start + digit_dimensions[1]] = cur_image
        cax = ax.matshow(concat_images, cmap=cmap, vmin=vmin, vmax=vmax)
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
    return cax


def save_images(images, filename, **kwargs):
```

```
    fig = plt.figure(1)
    fig.clf()
    ax = fig.add_subplot(111)
    plot_images(images, ax, **kwargs)
    fig.patch.set_visible(False)
    ax.patch.set_visible(False)
    plt.savefig(filename)
    plt.close() # Included

def train_log_regression(images, labels, learning_rate, max_iter):
    """ Used in Q1
        Inputs: train_images, train_labels, learning rate,
        and max num of iterations in gradient descent
        Returns the trained weights (w/o intercept)"""
    N_data, D_data = images.shape
    K_data = labels.shape[1]
    weights = np.zeros((D_data, K_data))

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding 01 - ###
    cross_ent_history = np.zeros(max_iter)
    accuracy_history = np.zeros(max_iter)
    grad_history = np.zeros(max_iter)
    it = 0
    converged = False

    # v.FINAL2 : Here, differently in the previous version i defined the
    # tolerance to check if accuracy stabilizes from previous step.
    tolerance = 1e-5

    while (not converged and (it < max_iter)):
        # Calculate the log-Softmax for the calculated weights
        lsmax = log_softmax(images, weights)

        # Calculate the prediction - This info is also self-contained in lsmax
        # psmax = predict(lsmax) # v.FINAL_3

        # Calculate the Cross Entropy over the prediction
        cent = cross_ent(labels, lsmax)

        # Calculate the Gradient
        grad = (images.T).dot(np.exp(lsmax)-labels)

        # Update Weights with the current gradient descent value
        weights -= grad*learning_rate
```

```
        # Stores Gradient, Cross-Entropy and Accuracy History
        grad_history[it] = np.linalg.norm(grad)
        cross_ent_history[it] = cent
        accuracy_history[it] = accuracy(lsmax, labels)

        # Check for convergence under a tolerance defined
        # converged = (grad_history[it]<=tolerance) - OLD Convergence Criteria

        # v-FINAL2: Initially I was checking the size of gradient
        # to check convergence, but I noticed it takes lots of interations
        # to became small, so I changed to check the convergence of
        if (it > 0):
            converged =
                (abs(accuracy_history[it]-accuracy_history[it-1])<=tolerance)

        # Increment iteration
        it += 1

    # Save Cross-Entrpy for later reference
    plt.figure(figsize=(10,7))
    plt.title('Cross entropy History Learning rate='+
              str(learning_rate)+' / Max-Iter='+str(max_iter), fontsize=12)
    plt.plot(cross_ent_history[:(it-1)]);
    plt.xlabel('Iteration')
    plt.ylabel('Cross-Entropy')
    # plt.show()
    plt.savefig('Cross-Entropy-GD.png')
    plt.close()

    # Save Training Accuracy for later reference
    plt.figure(figsize=(10,7))
    plt.title('Training Accuracy History | Learning rate='+
              str(learning_rate)+' / Max-Iter='+str(max_iter), fontsize=12)
    plt.plot(accuracy_history[:(it-1)]);
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    # plt.show()
    plt.savefig('Trn_Accuracy-GD.png')
    plt.close()
    #### - End Coding 01 - ###

    w0 = None # No intercept for log-reg
    return weights, w0


def train_gda(images, labels):
    """ Used in Q2
```

```
    Inputs: train_images, train_labels
    Returns the trained weights, the intercept, and D x K class means,
    and D x D common covariance matrix."""
N_data, D_data = images.shape
K_data = labels.shape[1]

# YOU NEED TO WRITE THIS PART
### - Start Coding 02 - ###

# Calculating the number of items for each class
N_k = np.sum(labels, axis=0)

# Estimator for the Priors of each class
Pi_k = N_k/N_data

# Estimator for Mu_k
Mu = ((labels.T).dot(images)).T[:,:K_data] / N_k[:K_data]

# Initialize Sigma with non-zeros to avoid singulatity
Sigma_hat = np.identity(D_data)*1e-6

# Estimate Sigma_k and Sigma_hat

for k in range(K_data):
    # Select Images from Class-k
    Idx_k = np.where(labels[:,k])[0]

    # Estimator for Sigma_k
    Sigma_hat_k =
        (1/N_k[k])*(images[Idx_k,:]-Mu[:,k]).T.dot((images[Idx_k,:]-Mu[:,k]))

    Sigma_hat += (N_k[k]/N_data)*Sigma_hat_k

Inv_Sigma_hat = np.linalg.inv(Sigma_hat)

# Initializing Intercept wk0 and weights for all Classes
w0 = np.zeros(K_data)
weights = np.zeros((D_data, K_data))
for k in range(K_data):
    # Calculating Intercept wk0 for each Classe
    w0[k] = -0.5*((Mu[:,k].T).dot(Inv_Sigma_hat)).dot(Mu[:,k])+np.log(Pi_k[k])

    # Calculating the weights for each class
    weights[:,k] = Inv_Sigma_hat.dot(Mu[:,k])

#### - End Coding 02 - ###
```

```
    return weights, w0, Mu, Sigma_hat

def log_softmax(images, weights, w0=None):
    """ Used in Q1 and Q2
        Inputs: images, and weights
        Returns the log_softmax values."""
    if w0 is None: w0 = np.zeros(weights.shape[1])

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    XW = images.dot(weights) + w0
    return XW - np.array([logsumexp(XW,axis=1),]).T
    ### - End Coding - ###


def cross_ent(log_Y, train_labels):
    """ Used in Q1
        Inputs: log of softmax values and training labels
        Returns the cross entropy."""

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    return(- np.sum(train_labels * (log_Y)))
    ### - End Coding - ###

def predict(log_softmax):
    """ Used in Q1 and Q2
        Inputs: matrix of log softmax values
        Returns the predictions"""

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    values = log_softmax.argmax(axis=1)
    return(np.eye(log_softmax.shape[1])[values])
    ### - End Coding - ###

def accuracy(log_softmax, labels):
    """ Used in Q1 and Q2
        Inputs: matrix of log softmax values and 1-of-K labels
        Returns the accuracy based on predictions from log likelihood values"""

    # YOU NEED TO WRITE THIS PART
    ### - Start Coding - ###
    notmtc = np.count_nonzero(labels.argmax(axis=1)-log_softmax.argmax(axis=1))
```

```
    return(1-notmtc/labels.shape[0])
    ### - End Coding - ###

def main():
    N_data = 60000 # Num of samples to be used in training
    # Set this to a small number while experimenting.
    # For log reg, finally use the entire training dataset for training
       (N_data=None).
    # For gda, use as many training samples as your computer can handle.

    print('\nLoading MNIST...\n')

    train_images, train_labels, test_images, test_labels = load_mnist(N_data)

    # Q1: train logistic regression
    learning_rate, max_iter = .00001, 100
    Proc = 'Multi-Class Logistic Regression'
    ProcID = 'MC-LogReg'

    # Q1(c) - Training Multiclass Logistic Regression
    print('\nProcessing %s...\n' % Proc)
    weights, w0 = train_log_regression(train_images, train_labels, learning_rate,
       max_iter)

    # Q1(d) - Saving the Weights as 10 images
    print('\nSaving Images...\n')
    save_images(weights.T, 'weights.png')

    # Q2(b) - Train gaussian discriminant
    # Proc = 'Gaussian Discriminant Analysis'
    # ProcID = 'GDA'
    # print('\Processing %s...\n' % Proc)
    #weights, w0, Mu, Sigma = train_gda(train_images, train_labels)
    #save_images(Mu.T, 'means.png')

    # Q2(e) - Using the Generative model generate 10 samples of digit '0' and '3'

    # Generate images of No. 0
    #np.random.seed(123)
    #new_digit = 0
    #print('\nGenerating & Saving Images...\n')
    #new_images = np.random.multivariate_normal(Mu[:, new_digit], Sigma, 10)
    #save_images((new_images > .5).astype(float), 'new_images_0.png')

    # Generate images of No. 3
    #new_digit = 3
    #new_images = np.random.multivariate_normal(Mu[:, new_digit], Sigma, 10)
```

```
#save_images((new_images > .5).astype(float), 'new_images_3.png')

# Q1-Q2(c) - Generate Predictions no Training and Test dats-sets
#          - Report Errors and Accuracy for both processes

log_softmax_train = log_softmax(train_images, weights, w0)
log_softmax_test = log_softmax(test_images, weights, w0)

train_accuracy = accuracy(log_softmax_train, train_labels)*100.0
test_accuracy = accuracy(log_softmax_test[:10000], test_labels[:10000])*100.0

print('\nProcessing Report...\n')

f=open('Summary_'+ProcID+'.prn','w')
f.write("\n------ Processing Summary (%s) -----------\n" % Proc)
f.write("\nTraining accuracy is %5.2f%%" % train_accuracy)
f.write("\nTest accuracy is %5.2f%%\n" % test_accuracy)

f.write("\n------ Listing of Errors from TRAINING procedure (%d Samples)
    ------\n" % N_data)

ErrTrain =
    np.nonzero(train_labels.argmax(axis=1)-log_softmax_train.argmax(axis=1))[0]
f.write("\n>>> Total No. of Errors: %6d\n" % ErrTrain.shape[0])
f.write("\n>>> Note: Only first 120 will be printed due to limitation on
    Crowdmark\n")
f.write("        to manage high no. of pages in uploaded PDF format.\n\n")
dtrain = {'Img No.':ErrTrain,
          'Label':train_labels[ErrTrain].argmax(axis=1),
          'Classif.':log_softmax_train[ErrTrain].argmax(axis=1)}
dftrain = pd.DataFrame(dtrain)
f.write(dftrain[:120].to_string(header=True, index=True))

f.write("\n\n------ Listing of Errors from TESTING procedure (%d
    Samples)------\n" % 10000)
ErrTest =
    np.nonzero(test_labels.argmax(axis=1)-log_softmax_test[:10000].argmax(axis=1))[0]
f.write("\n>>> Total No. of Errors: %6d\n" % ErrTest.shape[0])
f.write("\n>>> Note: Only first 120 will be printed due to limitation on
    Crowdmark\n")
f.write("        to manage high no. of pages in uploaded PDF format.\n\n")
dtest = {'Img No.':ErrTest,
         'Label':test_labels[ErrTest].argmax(axis=1),
         'Classif.':log_softmax_test[ErrTest].argmax(axis=1)}
dftest = pd.DataFrame(dtest)
f.write(dftest[:120].to_string(header=True, index=True))
```

```
    f.close()

    print('\nProcessing concluded!')

if __name__ == '__main__':
    main()
```

Question 2

Gaussian Discriminant Analysis

Item (a)

Solution.

Considering we have a data-set composed by $\{\boldsymbol{x}_n, \boldsymbol{t}_n\}$ from a multi-class classification problem, for a data-point $\boldsymbol{x}_n \in \mathbb{R}^D$ from class $\mathcal{C}_k$, we have its *1-to-K* classification scheme given by $\boldsymbol{t}_n$, where $t_{nk} = 1$, which implies:

$$p(\boldsymbol{x}_n, \mathcal{C}_k) = p(\mathcal{C}_k)p(\boldsymbol{x}_n|\mathcal{C}_k), \text{ for } k = 1, \ldots, K \tag{5}$$

Assuming that the priors are given by $p(\mathcal{C}_k) = \pi_k$ and class means $\boldsymbol{\mu}_k$ for $k = 1, \ldots, K$, from (5) we can write the likelihood function of our model as follows:

$$\begin{aligned} p(\boldsymbol{T}|\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}) &= p(\boldsymbol{X}, \mathcal{C}_1, \ldots, \mathcal{C}_K) \\ &= \prod_{n=1}^{N}\prod_{k=1}^{K}\Big[p(\mathcal{C}_k)p(\boldsymbol{x}_n|\mathcal{C}_k)\Big]^{t_{nk}} \\ &= \prod_{n=1}^{N}\prod_{k=1}^{K}\Big[\pi_k \times \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})\Big]^{t_{nk}} \end{aligned}$$

$$\implies p(\boldsymbol{T}|\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}) = \prod_{n=1}^{N}\prod_{k=1}^{K}\Big[\pi_k \times \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})\Big]^{t_{nk}} \tag{6}$$

... where $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N]^T$ the multi-class *1-to-K* classification matrix, containing the labels for each image $\boldsymbol{x}_n$.

Assuming the data generating distribution is Gaussian, i.e., $p(\boldsymbol{x}|\mathcal{C}_k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, we have:

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\Big\{-\frac{1}{2}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)\Big\} \tag{7}$$

Now, calculating the log-likelihood from (6) and substituting (7) in it, then we have:

$$\ln p(\boldsymbol{T}|\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma}) = \ln \prod_{n=1}^{N}\prod_{k=1}^{K}\Big[\pi_k \times \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k,\boldsymbol{\Sigma})\Big]^{t_{nk}}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \times \ln\Big[\pi_k \times \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\Big\{-\frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)\Big\}\Big]$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \times \Big[\ln\pi_k - \frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)\Big]$$

Then the log-likelihood implied for the model is given by:

$$\implies \ln p(\boldsymbol{T}|\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma}) =$$
$$\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \times \Big[\ln\pi_k - \frac{D}{2}\ln(2\pi) - \frac{1}{2}\ln(|\boldsymbol{\Sigma}|) - \frac{1}{2}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n-\boldsymbol{\mu}_k)\Big] \quad (8)$$

The MLE for the priors are obtained by minimizing (8) subject to the side condition $\sum_{k=1}^{K}\pi_k = 1$.

From [1] we will use *Lagrange Multipliers* to obtain the solution for this maximization problem:

$$\nabla_{\pi_j}\ln p(\boldsymbol{T}|\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma}) = 0 \text{ for } j=1,\ldots,K \text{ and subject to } \sum_{k=1}^{K}\pi_k = 1$$

In other words, we need to find the solution for the following *Lagrangian*:

$$\nabla_{\pi_j}\Big[\ln p(\boldsymbol{T}|\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma})\Big] + \lambda\nabla_{\pi_j}\Big(1-\sum_{k=1}^{K}\pi_k\Big) = 0 \text{ for } j=1,\ldots,K \quad (9)$$

From (9), after removing all terms not dependent on the priors from the first part of the Lagrangian, we have:

$$\nabla_{\pi_j}\Big[\ln p(\boldsymbol{T}|\pi_1,\ldots,\pi_K,\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_K,\boldsymbol{\Sigma})\Big]+\lambda\nabla_{\pi_j}\Big(1-\sum_{k=1}^{K}\pi_k\Big)=0$$

$$\nabla_{\pi_j}\Big(\sum_{n=1}^{N}\sum_{k=1}^{K}t_{nk}\times\ln\pi_k\Big)+\lambda(-1)=0$$

$$\nabla_{\pi_j}\Big(\sum_{n=1}^{N}\sum_{k=1}^{K}t_{nk}\times\ln\pi_k\Big)-\lambda=0$$

$$\sum_{n=1}^{N}\sum_{k=1}^{K}t_{nk}\nabla_{\pi_j}\big(\ln\pi_k\big)-\lambda=0$$

$$\underbrace{\sum_{n=1}^{N}t_{nj}}_{N_j}\times\frac{1}{\pi_j}-\lambda=0$$

$$\implies \lambda=\frac{N_j}{\pi_j} \tag{10}$$

We know that each N_k is cardinality of $\mathcal{C}_k$ then we have $\sum_{k=1}^{K}N_k=N$. Using (10) and the side-condition we have:

$$\sum_{j=1}^{K}N_j=N$$

$$\sum_{j=1}^{K}(\lambda\pi_j)=N$$

$$\lambda\underbrace{\sum_{j=1}^{K}\pi_j}_{=1}=N$$

$$\implies \lambda=N \tag{11}$$

Now, substituting (11) in (10) we have the MLE estimator for the priors given by:

$$\implies \hat{\pi}_k=\frac{N_k}{N}\text{ with }k=1,\ldots,K \tag{12}$$

Now, calculating the MLE estimator for the class means we need to maximize the gradient of (8) w.r.t. to $\boldsymbol{\mu}_k$:

$$\nabla_{\boldsymbol{\mu}_j} \ln p(\boldsymbol{T}|\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}) = 0 \text{ for } j = 1, \ldots, K$$

Removing all terms not dependent from the class means, the maximization problem is reduced to calculate the solutions for the following gradient and setting it equals to 0:

$$\begin{aligned}
&\nabla_{\boldsymbol{\mu}_j} \ln p(\boldsymbol{T}|\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}) \\
&\quad = \nabla_{\boldsymbol{\mu}_j} \Big\{ \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \Big[-\frac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \Big] \Big\} \\
&\quad = \nabla_{\boldsymbol{\mu}_j} \Big\{ \sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \times \Big[-\frac{1}{2} \big(\boldsymbol{x}_n^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x}_n - 2\boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{x}_n + \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k \big) \Big] \Big\} \\
&\quad = \sum_{n=1}^{N} t_{nj} \times \Big[-\frac{1}{2} \big(-2\boldsymbol{\Sigma}^{-1} \boldsymbol{x}_n + 2\boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j \big) \Big] \\
&\quad = \sum_{n=1}^{N} t_{nj} \times \Big(\boldsymbol{\Sigma}^{-1} \boldsymbol{x}_n - \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_j \Big) \\
&\quad = \boldsymbol{\Sigma}^{-1} \Big(\sum_{n=1}^{N} t_{nj} \times \boldsymbol{x}_n - \boldsymbol{\mu}_j \times \underbrace{\sum_{n=1}^{N} t_{nj}}_{N_j} \Big) = 0
\end{aligned}$$

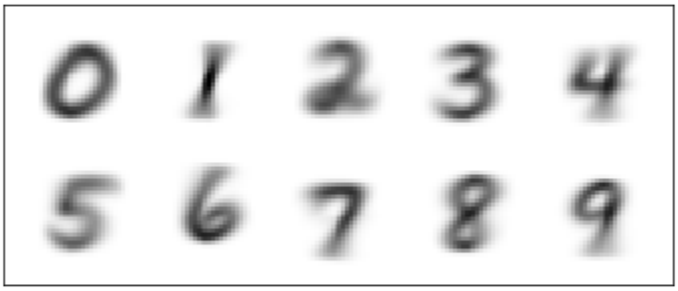
Then, solving this equation for $\hat{\boldsymbol{\mu}}_j$ and $j = 1, \ldots, K$ we have that the MLE for each class mean is given by:

$$\implies \hat{\boldsymbol{\mu}}_j = \sum_{n=1}^{N} \frac{t_{nj}}{N_j} \boldsymbol{x}_n \text{ for } j = 1, \ldots, K \tag{13}$$

Item (b)

Solution.

The image containing each class mean $\hat{\boldsymbol{\mu}}_j$ for $j = 1, \ldots, K$ is as follows:



Implementation details can be verified in section **Code Listing (I)**, at the end of this question.

Item (c)

Solution.

For processing the **Gaussian Discriminant Analysis** it was used the entire `MNIST` database, i.e., $60,000$ *images for training* process and $10,000$ *images for testing* process.

The accuracy of each process and partial[2] listing of classification errors - which includes the *image number* (`Img No.`), *correct images's label* (`Label`) and *model's classification* (`Classif.`) - are as follows:

```
------ Processing Summary (Gaussian Discriminant Analysis) -----------

Training accuracy is 86.75%
Test accuracy is 86.65%

------ Listing of Errors from TRAINING procedure (60000 Samples) ------

>>> Total No. of Errors:   7949

>>> Note: Only first 120 will be printed due to limitation on Crowdmark
```

[2]Only first 120 will be printed due to limitation on *Crowdmark* to manage high no. of pages in uploaded PDF format.

to manage high no. of pages in uploaded PDF format.

	Img No.	Label	Classif.
0	19	9	7
1	24	1	2
2	26	4	9
3	28	2	7
4	29	7	8
5	33	9	7
6	34	0	8
7	38	7	4
8	48	9	5
9	55	8	9
10	80	9	3
11	100	5	9
12	101	7	9
13	120	2	3
14	122	2	6
15	132	5	3
16	133	9	7
17	134	1	5
18	140	7	3
19	143	2	8
20	148	7	4
21	158	7	9
22	168	7	1
23	173	5	8
24	178	2	6
25	182	5	3
26	183	9	7
27	190	2	6
28	202	8	5
29	207	3	8
30	212	7	1
31	224	1	9
32	228	3	5
33	232	0	6
34	240	8	1
35	250	3	4
36	254	3	5
37	262	2	0
38	264	9	4
39	268	2	7
40	278	5	6
41	282	9	7

42	304	9	7
43	318	2	1
44	334	9	4
45	340	7	9
46	346	9	8
47	370	7	1
48	376	5	8
49	384	7	4
50	404	8	3
51	418	8	5
52	420	5	9
53	444	2	1
54	459	3	9
55	469	8	5
56	470	1	8
57	472	2	1
58	480	5	8
59	488	6	1
60	494	6	0
61	497	7	9
62	500	3	7
63	502	5	8
64	528	8	5
65	529	9	4
66	533	1	8
67	540	3	9
68	544	5	3
69	547	6	4
70	558	2	5
71	559	8	5
72	574	3	1
73	584	2	6
74	588	2	1
75	589	9	4
76	602	8	3
77	608	7	9
78	610	5	9
79	611	7	1
80	626	9	4
81	629	3	7
82	630	6	8
83	635	5	2
84	644	7	1
85	646	2	1
86	647	4	5

```
87       665      6         4
88       670      3         2
89       674      2         1
90       688      2         3
91       704      9         3
92       706      4         9
93       712      0         8
94       720      8         1
95       736      8         3
96       749      4         9
97       760      3         5
98       769      8         5
99       770      6         4
100      778      5         4
101      784      8         3
102      788      9         8
103      801      2         4
104      820      7         9
105      828      4         2
106      832      5         3
107      836      5         8
108      843      3         4
109      846      6         8
110      850      4         6
111      854      2         9
112      863      2         4
113      866      2         3
114      870      1         3
115      884      7         1
116      886      5         2
117      889      0         5
118      892      6         1
119      896      4         5

------ Listing of Errors from TESTING procedure (10000 Samples)------

>>> Total No. of Errors:   1335

>>> Note: Only first 120 will be printed due to limitation on Crowdmark
          to manage high no. of pages in uploaded PDF format.

     Img No.  Label  Classif.
0         11      6         8
1         18      3         2
2         33      4         5
```

3	38	2	3
4	43	2	1
5	46	1	3
6	47	2	6
7	63	3	2
8	73	9	7
9	80	7	9
10	92	9	4
11	97	7	1
12	111	7	1
13	124	7	9
14	149	2	5
15	150	9	4
16	167	5	3
17	175	7	9
18	185	9	4
19	195	3	9
20	211	5	9
21	217	6	5
22	218	5	7
23	233	8	7
24	241	9	8
25	243	7	4
26	244	2	3
27	245	3	6
28	247	4	6
29	256	2	1
30	259	6	0
31	261	5	1
32	264	9	8
33	268	8	3
34	282	7	2
35	290	8	4
36	299	8	3
37	300	4	1
38	307	7	9
39	313	3	5
40	318	2	3
41	320	9	1
42	321	2	7
43	333	5	8
44	336	9	4
45	340	5	3
46	341	6	4
47	349	3	9

48	352	5	0
49	358	7	9
50	359	9	4
51	362	2	8
52	363	2	1
53	366	6	1
54	376	4	9
55	380	0	5
56	381	3	7
57	389	9	7
58	398	4	9
59	403	8	4
60	404	2	8
61	406	5	0
62	412	5	3
63	443	0	5
64	444	2	8
65	445	6	0
66	448	9	8
67	449	3	5
68	456	2	5
69	457	6	5
70	460	5	7
71	464	3	7
72	478	5	8
73	479	9	3
74	488	9	7
75	495	8	2
76	502	5	3
77	507	3	5
78	508	6	5
79	511	4	8
80	514	6	4
81	515	3	8
82	516	2	6
83	523	1	8
84	530	9	4
85	531	3	6
86	543	8	4
87	550	7	9
88	551	7	1
89	552	0	6
90	565	4	9
91	569	3	5
92	578	3	8

93	582	8	2
94	583	2	8
95	591	8	3
96	593	9	7
97	605	7	9
98	606	8	3
99	613	2	8
100	617	7	4
101	619	1	8
102	624	2	1
103	627	9	4
104	628	3	9
105	629	2	6
106	658	7	4
107	659	2	1
108	661	0	8
109	667	7	1
110	674	5	3
111	684	7	3
112	688	6	8
113	689	7	9
114	691	8	4
115	692	5	7
116	707	4	9
117	710	5	3
118	714	8	5
119	717	0	6

Implementation details can be verified in section **Code Listing (I)**, at the end of this question.

Item (d)

Solution.

The *end-game* prediction performance of both models are similar, with a slight advantage to *Multi-Class Logistic Regression* (accuracy of 90.05%) in relation to *Gaussian Discriminant Analysis* (accuracy of 86.65%).

In order to analyze the behaviour of both processes at different levels of number of training samples, we run the training and test cycles 12 times, with a $5,000$-step of training samples up to the maximum number of images in `MNIST`, which is $60,000$.

We can see from the graphs below that the *testing curves* have similar behavior, i.e., both started approximately from the same level of 84% and grew continuously as the number of

training samples increased. We observe a higher increase in Multi-Class Logistic Regression, up to a level of 90% accuracy when compared to Gaussian Discriminant Analysis, which achieved its maximum at 86% level.

On the other hand, the performance of *training curves* are quite different: Multi-Class Logistic Regression increases its performance as the number of sample increases, and Gaussian Discriminant Analysis has the opposite behaviour: its performance decreases as we increase the number of samples.

Some sort of combined situations might affect the performances differently depending of the process we use.

- With increasing number of training images, the distribution of the samples might differ significantly, i.e., we can see different concentrations of each class which influences the estimation of the covariance matrices and, by consequence, the estimation of overall $\mathbf{\Sigma}$ included in training. On the other side, with fewer samples the distribution of samples might be *more sparse* and GDA is expected to perform well in these cases;
- Another possible reason might be also due to, when increasing the number of images, the mass of images is less separable and in these situations GDA's estimates becomes worst and, on the other hand, logistic regression benefits on less separable situations,

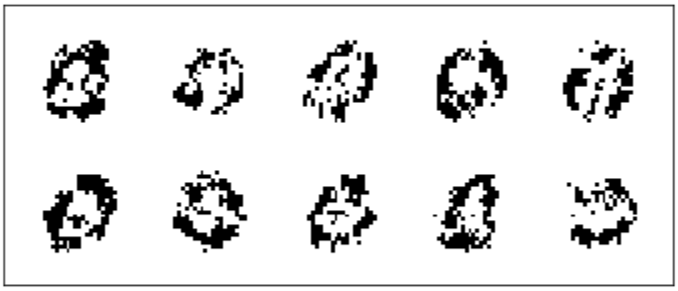
Another intrinsic factor that might influence the different behaviors we observed is that, with increasing number of images, GDA is more penalized with an also increasing covariance structure which influences the MLE estimation of the parameters. On the other hand, Multi-Class Logistic Regression makes use of numerical methods (*gradient descent*) to estimate the parameters, which benefits with increasing number of training images.

Implementation details can be verified in section **Code Listing (II)**, at the end of this question.

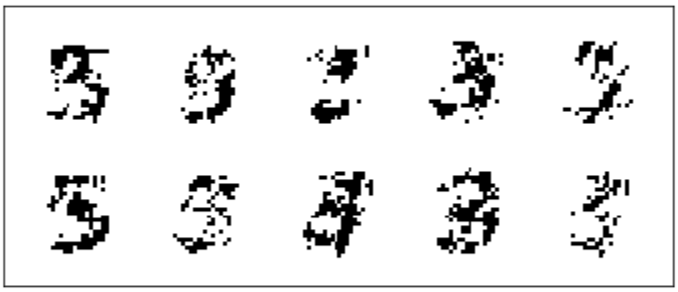
Item (e)

Solution.

The image containing the generated images of digit 0 is as follows:



... and for digit 3:



Implementation details can be verified in section **Code Listing (I)**, at the end of this question.

Code-Listing (I)

```
from __future__ import absolute_import
from __future__ import print_function
from future.standard_library import install_aliases
install_aliases()
import numpy as np
import pandas as pd
from scipy.special import logsumexp
import os
import gzip
import struct
import array
import matplotlib.pyplot as plt
import matplotlib.image
from urllib.request import urlretrieve

def download(url, filename):
    if not os.path.exists('data'):
        os.makedirs('data')
    out_file = os.path.join('data', filename)
    if not os.path.isfile(out_file):
        urlretrieve(url, out_file)


def mnist():
    base_url = 'http://yann.lecun.com/exdb/mnist/'

    def parse_labels(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data = struct.unpack(">II", fh.read(8))
            return np.array(array.array("B", fh.read()), dtype=np.uint8)

    def parse_images(filename):
        with gzip.open(filename, 'rb') as fh:
            magic, num_data, rows, cols = struct.unpack(">IIII", fh.read(16))
            return np.array(array.array("B", fh.read()),
                dtype=np.uint8).reshape(num_data, rows, cols)

    for filename in ['train-images-idx3-ubyte.gz',
                     'train-labels-idx1-ubyte.gz',
                     't10k-images-idx3-ubyte.gz',
                     't10k-labels-idx1-ubyte.gz']:
        download(base_url + filename, filename)

    train_images = parse_images('data/train-images-idx3-ubyte.gz')
    train_labels = parse_labels('data/train-labels-idx1-ubyte.gz')
```

```
    test_images = parse_images('data/t10k-images-idx3-ubyte.gz')
    test_labels = parse_labels('data/t10k-labels-idx1-ubyte.gz')

    return train_images, train_labels, test_images[:10000], test_labels[:10000]


def load_mnist(N_data=None):
    partial_flatten = lambda x: np.reshape(x, (x.shape[0], np.prod(x.shape[1:])))
    one_hot = lambda x, k: np.array(x[:, None] == np.arange(k)[None, :],
        dtype=int)
    train_images, train_labels, test_images, test_labels = mnist()
    train_images = (partial_flatten(train_images) / 255.0 > .5).astype(float)
    test_images = (partial_flatten(test_images) / 255.0 > .5).astype(float)
    K_data = 10
    train_labels = one_hot(train_labels, K_data)
    test_labels = one_hot(test_labels, K_data)
    if N_data is not None:
        train_images = train_images[:N_data, :]
        train_labels = train_labels[:N_data, :]

    return train_images, train_labels, test_images, test_labels


def plot_images(images, ax, ims_per_row=5, padding=5, digit_dimensions=(28, 28),
                cmap=matplotlib.cm.binary, vmin=None, vmax=None):
    """Images should be a (N_images x pixels) matrix."""
    N_images = images.shape[0]
    N_rows = np.int32(np.ceil(float(N_images) / ims_per_row))
    pad_value = np.min(images.ravel())
    concat_images = np.full(((digit_dimensions[0] + padding) * N_rows + padding,
                             (digit_dimensions[1] + padding) * ims_per_row +
                                padding), pad_value)
    for i in range(N_images):
        cur_image = np.reshape(images[i, :], digit_dimensions)
        row_ix = i // ims_per_row
        col_ix = i % ims_per_row
        row_start = padding + (padding + digit_dimensions[0]) * row_ix
        col_start = padding + (padding + digit_dimensions[1]) * col_ix
        concat_images[row_start: row_start + digit_dimensions[0],
                      col_start: col_start + digit_dimensions[1]] = cur_image
        cax = ax.matshow(concat_images, cmap=cmap, vmin=vmin, vmax=vmax)
        plt.xticks(np.array([]))
        plt.yticks(np.array([]))
    return cax


def save_images(images, filename, **kwargs):
```

```
    fig = plt.figure(1)
    fig.clf()
    ax = fig.add_subplot(111)
    plot_images(images, ax, **kwargs)
    fig.patch.set_visible(False)
    ax.patch.set_visible(False)
    plt.savefig(filename)
    plt.close() # Included

def train_log_regression(images, labels, learning_rate, max_iter):
    """ Used in Q1
        Inputs: train_images, train_labels, learning rate,
        and max num of iterations in gradient descent
        Returns the trained weights (w/o intercept)"""
    N_data, D_data = images.shape
    K_data = labels.shape[1]
    weights = np.zeros((D_data, K_data))

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding 01 - ###
    cross_ent_history = np.zeros(max_iter)
    accuracy_history = np.zeros(max_iter)
    grad_history = np.zeros(max_iter)
    it = 0
    converged = False
    tolerance = 1e-4
    while (not converged and (it < max_iter)):
        # Calculate the log-Softmax for the calculated weights
        lsmax = log_softmax(images, weights)

        # Calculate the prediction
        psmax = predict(lsmax)

        # Calculate the Cross Entropy over the prediction
        cent = cross_ent(psmax, labels)

        # Calculate the Gradient
        grad = -(images.T).dot(labels-np.exp(lsmax))

        # Update Weights with the current gradient descent value
        weights = weights - grad*learning_rate

        # Stores Gradient, Cross-Entropy and Accuracy History
        grad_history[it] = np.linalg.norm(grad)
        cross_ent_history[it] = cent
        accuracy_history[it] = accuracy(lsmax, labels)
```

```
        # Check for convergence under a tolerance defined
        converged = (grad_history[it]<=tolerance)

        # Increment iteration
        it += 1

    print('\nFinished Iterations - Cross entropy %.2f\n' % cent)

    # Save Cross-Entrpy for later reference
    plt.figure(figsize=(10,7))
    plt.title('Cross entropy History Learning rate='+
              str(learning_rate)+' / Max-Iter='+str(max_iter), fontsize=12)
    plt.plot(cross_ent_history);
    plt.xlabel('Iteration')
    plt.ylabel('Cross-Entropy')
    # plt.show()
    plt.savefig('Cross-Entropy-GD.png')
    plt.close()

    # Save Training Accuracy for later reference
    plt.figure(figsize=(10,7))
    plt.title('Training Accuracy History | Learning rate='+
              str(learning_rate)+' / Max-Iter='+str(max_iter), fontsize=12)
    plt.plot(accuracy_history[:it]);
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    # plt.show()
    plt.savefig('Trn_Accuracy-GD.png')
    plt.close()

    #### - End Coding 01 - ###

    w0 = None # No intercept for log-reg
    return weights, w0


def train_gda(images, labels):
    """ Used in Q2
        Inputs: train_images, train_labels
        Returns the trained weights, the intercept, and D x K class means,
        and D x D common covariance matrix."""
    N_data, D_data = images.shape
    K_data = labels.shape[1]

    # YOU NEED TO WRITE THIS PART
    ### - Start Coding 02 - ###
```

```
    # Calculating the number of items for each class
    N_k = np.sum(labels, axis=0)

    # Estimator for the Priors of each class
    Pi_k = N_k/N_data

    # Estimator for Mu_k
    Mu = ((labels.T).dot(images)).T[:,:K_data] / N_k[:K_data]

    # Initialize Sigma with non-zeros to avoid singulatity
    Sigma_hat = np.identity(D_data)*1e-6

    # Estimate Sigma_k and Sigma_hat

    for k in range(K_data):
        # Select Images from Class-k
        Idx_k = np.where(labels[:,k])[0]

        # Estimator for Sigma_k
        Sigma_hat_k =
            (1/N_k[k])*(images[Idx_k,:]-Mu[:,k]).T.dot((images[Idx_k,:]-Mu[:,k]))

        Sigma_hat += (N_k[k]/N_data)*Sigma_hat_k

    Inv_Sigma_hat = np.linalg.inv(Sigma_hat)

    # Initializing Intercept wk0 and weights for all Classes
    w0 = np.zeros(K_data)
    weights = np.zeros((D_data, K_data))
    for k in range(K_data):
        # Calculating Intercept wk0 for each Classe
        w0[k] = -0.5*((Mu[:,k].T).dot(Inv_Sigma_hat)).dot(Mu[:,k])+np.log(Pi_k[k])

        # Calculating the weights for each class
        weights[:,k] = Inv_Sigma_hat.dot(Mu[:,k])

    #### - End Coding 02 - ###

    return weights, w0, Mu, Sigma_hat

def log_softmax(images, weights, w0=None):
    """ Used in Q1 and Q2
        Inputs: images, and weights
        Returns the log_softmax values."""
    if w0 is None: w0 = np.zeros(weights.shape[1])
```

```
    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    XW = images.dot(weights) + w0
    return XW - np.array([logsumexp(XW,axis=1),]).T
    ### - End Coding - ###


def cross_ent(log_Y, train_labels):
    """ Used in Q1
        Inputs: log of softmax values and training labels
        Returns the cross entropy."""

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    return(- np.sum(train_labels * (log_Y)))
    ### - End Coding - ###

def predict(log_softmax):
    """ Used in Q1 and Q2
        Inputs: matrix of log softmax values
        Returns the predictions"""

    # YOU NEED TO WRITE THIS PART

    ### - Start Coding - ###
    values = log_softmax.argmax(axis=1)
    return(np.eye(log_softmax.shape[1])[values])
    ### - End Coding - ###

def accuracy(log_softmax, labels):
    """ Used in Q1 and Q2
        Inputs: matrix of log softmax values and 1-of-K labels
        Returns the accuracy based on predictions from log likelihood values"""

    # YOU NEED TO WRITE THIS PART
    ### - Start Coding - ###
    notmtc = np.count_nonzero(labels.argmax(axis=1)-log_softmax.argmax(axis=1))
    return(1-notmtc/labels.shape[0])
    ### - End Coding - ###

def main():
    N_data = 60000 # Num of samples to be used in training
    # Set this to a small number while experimenting.
    # For log reg, finally use the entire training dataset for training
        (N_data=None).
```

```
# For gda, use as many training samples as your computer can handle.

print('\nLoading MNIST...\n')

train_images, train_labels, test_images, test_labels = load_mnist(N_data)

# Q1: train logistic regression
#learning_rate, max_iter = .00001, 100
#Proc = 'Multi-Class Logistic Regression'
#ProcID = 'MC-LogReg'

# Q1(c) - Training Multiclass Logistic Regression
# print('\nProcessing %s...\n' % Proc)
#weights, w0 = train_log_regression(train_images, train_labels,
    learning_rate, max_iter)

# Q1(d) - Saving the Weights as 10 images
#print('\nSaving Images...\n')
#save_images(weights.T, 'weights.png')

# Q2(b) - Train gaussian discriminant
Proc = 'Gaussian Discriminant Analysis'
ProcID = 'GDA'
print('\Processing %s...\n' % Proc)
weights, w0, Mu, Sigma = train_gda(train_images, train_labels)
save_images(Mu.T, 'means.png')

# Q2(e) - Using the Generative model generate 10 samples of digit '0' and '3'

# Generate images of No. 0
np.random.seed(123)
new_digit = 0
print('\nGenerating & Saving Images...\n')
new_images = np.random.multivariate_normal(Mu[:, new_digit], Sigma, 10)
save_images((new_images > .5).astype(float), 'new_images_0.png')

# Generate images of No. 3
new_digit = 3
new_images = np.random.multivariate_normal(Mu[:, new_digit], Sigma, 10)
save_images((new_images > .5).astype(float), 'new_images_3.png')

# Q1-Q2(c) - Generate Predictions no Training and Test dats-sets
#          - Report Errors and Accuracy for both processes

log_softmax_train = log_softmax(train_images, weights, w0)
log_softmax_test = log_softmax(test_images, weights, w0)
```

```
    train_accuracy = accuracy(log_softmax_train, train_labels)*100.0
    test_accuracy = accuracy(log_softmax_test[:10000], test_labels[:10000])*100.0

    print('\nProcessing Report...\n')

    f=open('Summary_'+ProcID+'.prn','w')
    f.write("\n------ Processing Summary (%s) -----------\n" % Proc)
    f.write("\nTraining accuracy is %5.2f%%" % train_accuracy)
    f.write("\nTest accuracy is %5.2f%%\n" % test_accuracy)

    f.write("\n------ Listing of Errors from TRAINING procedure (%d Samples)
        ------\n" % N_data)

    ErrTrain =
        np.nonzero(train_labels.argmax(axis=1)-log_softmax_train.argmax(axis=1))[0]
    f.write("\n>>> Total No. of Errors: %6d\n" % ErrTrain.shape[0])
    f.write("\n>>> Note: Only first 120 will be printed due to limitation on
        Crowdmark\n")
    f.write("        to manage high no. of pages in uploaded PDF format.\n\n")
    dtrain = {'Img No.':ErrTrain,
              'Label':train_labels[ErrTrain].argmax(axis=1),
              'Classif.':log_softmax_train[ErrTrain].argmax(axis=1)}
    dftrain = pd.DataFrame(dtrain)
    f.write(dftrain[:120].to_string(header=True, index=True))

    f.write("\n\n------ Listing of Errors from TESTING procedure (%d
        Samples)------\n" % 10000)
    ErrTest =
        np.nonzero(test_labels.argmax(axis=1)-log_softmax_test[:10000].argmax(axis=1))[0]
    f.write("\n>>> Total No. of Errors: %6d\n" % ErrTest.shape[0])
    f.write("\n>>> Note: Only first 120 will be printed due to limitation on
        Crowdmark\n")
    f.write("        to manage high no. of pages in uploaded PDF format.\n\n")
    dtest = {'Img No.':ErrTest,
             'Label':test_labels[ErrTest].argmax(axis=1),
             'Classif.':log_softmax_test[ErrTest].argmax(axis=1)}
    dftest = pd.DataFrame(dtest)
    f.write(dftest[:120].to_string(header=True, index=True))

    f.close()

    print('\nProcessing concluded!')

if __name__ == '__main__':
    main()
```

Code-Listing (II)

This section lists the `main()` function modified to generate the training and testing curves when the number of images used to train the model increases. All other functions called by this function remains unchanged and can be referred in section **Code Listing (I)**.

```
def main():

    learning_rate, max_iter = .00001, 100
    Start_N = 5000
    End_N = 65000
    Step_N = 5000

    Size = int((End_N-Start_N)/Step_N)

    lreg_hist_Trn = np.zeros(Size)
    lreg_hist_Tst = np.zeros(Size)
    gda_hist_Trn = np.zeros(Size)
    gda_hist_Tst = np.zeros(Size)
    Lst_N_data = np.zeros(Size)

    it = 0

    for N_data in range(Start_N, End_N, Step_N):

        Lst_N_data[it] = N_data

        train_images, train_labels, test_images, test_labels = load_mnist(N_data)

        print("\n---- Processing Iteration No. ",it+1, '| No. of Samples=',
            N_data)
        print("\n\nLogistic Regression Started... ")

        # Logistic Regression
        weights, w0 = train_log_regression(train_images, train_labels,
            learning_rate, max_iter)

        log_softmax_train = log_softmax(train_images, weights, w0)
        log_softmax_test = log_softmax(test_images, weights, w0)

        lreg_hist_Trn[it] = accuracy(log_softmax_train, train_labels)
        lreg_hist_Tst[it] = accuracy(log_softmax_test[:10000],
            test_labels[:10000])

        print("\n\nLogistic Regression Finished... ")

        # Gaussian Discriminant Analysis
```

```
    print("\n\nGaussian Discriminant Analysis Started... ")
    weights, w0, Mu, Sigma = train_gda(train_images, train_labels)

    log_softmax_train = log_softmax(train_images, weights, w0)
    log_softmax_test = log_softmax(test_images, weights, w0)

    gda_hist_Trn[it] = accuracy(log_softmax_train, train_labels)
    gda_hist_Tst[it] = accuracy(log_softmax_test[:10000], test_labels[:10000])
    print("\n\nGaussian Discriminant Analysis Finished... ")

    it += 1

# Plot Performance Logistic Regression
plt.figure(figsize=(8,6))
plt.title('LogReg-Compare Training vs. Test Performance')
plt.plot(Lst_N_data, lreg_hist_Trn, label = 'Training');
plt.plot(Lst_N_data, lreg_hist_Tst, label = 'Test-10,000');
plt.xlabel('No. of Training Samples')
plt.ylabel('Accuracy')
plt.legend()
#plt.show()
plt.savefig('LogReg-Acc.png')
plt.close()

# Plot Performance Logistic Regression
plt.figure(figsize=(8,6))
plt.title('GDA-Compare Training vs. Test Performance')
plt.plot(Lst_N_data, gda_hist_Trn, label = 'Training');
plt.plot(Lst_N_data, gda_hist_Tst, label = 'Test-10,000');
plt.xlabel('No. of Training Samples')
plt.ylabel('Accuracy')
plt.legend()
#plt.show()
plt.savefig('GDA-Acc.png')
plt.close()

print("\n---- End Processing Processing - ", it, ' iterations\n')
```

References

[1] Kaplan W. *Advanced Calculus*. 5th Edition - Addison Wesley, 2002.

[2] Bishop C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.