APPLIED MACHINE LEARNING COURSEWORK

General instructions

Set: Friday 15th November at 12pm

Due: Friday 6th December at 5pm

Estimated time required: Approximately 12 hours.

Submission: Please write all your Python *codes* in one Jupyter notebook **in the order of questions**. Save the notebook with a name that clearly identifies you as the author, such as `sm007.ipynb` or `smith.ipynb`. Please scan or typeset your *written reports* in Q7 and Q9 as well as the statement on the use of GenAI (see below) such that you can produce a single **PDF file**. Give it a personalised name, such as `smith.pdf`. No marks will be lost if the reports are not typeset, provided they are neatly written.

Upload both the `.ipynb` and `.pdf` files on Moodle at the Coursework Submission point by the due deadline. **It is your responsibility to upload correct files.**

Conditions: This is an individual coursework, so you should **not** discuss it with other students. You may use generic scientific Python packages *numpy*, *matplotlib*, *scipy*, *pandas*, *sklearn* and their official documentation. You may **not** use solutions tailored to the questions, especially if those were written *for* you.

As usual for courseworks, the marking will be **not** anonymous. Please use your name (not the candidate number) in the files.

Value: This coursework carries 40% of the total marks for the unit.

Length: There is no minimum or maximum length for this assignment. Indicative guidance on approximate lengths for reports is provided in Section 2. In marking emphasis will be placed upon correct, well-structured and commented code, and correct, clear and precise argument in the written work.

Support and advice: You can ask **general** questions to the lecturer on the Padlet forum. For example: you may ask "what `ValueError` means in Python in general?". You may **not** ask "what should I do to *my* code to make it go away?".

Feedback: You will receive feedback within a maximum of three semester weeks after the submission deadline. The feedback consists of your *provisional* coursework mark and an overall feedback document commenting on the assessment across the cohort.

Late submission of coursework: If there are valid circumstances preventing you from meeting the deadline, your Director of Studies may grant you an extension to the specified submission date, if it is requested before the deadline. Forms to request an extension are available on SAMIS.

- If you submit a piece of work after the submission deadline and no extension has been granted, the maximum mark possible will be the pass mark.
- If you submit work more than five working days after the submission deadline, you will normally receive a mark of 0 (zero), unless you have been granted an extension.

Academic integrity statement: Academic misconduct is defined by the University as "the use of unfair means in any examination or assessment procedure". This includes (but is not limited to) cheating, collusion, plagiarism, fabrication, or falsification. The University's Quality Assurance Code of Practice, QA53 Examination and Assessment Offences, sets out the consequences of committing an offence and the penalties that might be applied.

Generative AI: Type B: GenAI tools may be used to answer any question but their use is not mandatory in order to complete the assessment. Under the University's Academic Integrity Statement, you 'must not present content created by generative AI tools as though it were your own'. Any text or code produced by GenAI must be checked for correctness and cited.

In addition, you must **include a short statement** (max 250 words) at the beginning of your submission indicating: what tools you used and **how** you used them **OR** that you have not used GenAI. You should be prepared to explain anything in your submission to an examiner if asked to do so.

1. Background

The actual coursework questions are listed in Section 2. However, they refer to algorithms and equations set in this mathematical background, so please study it before attempting the questions.

1.1. **Probabilistic multiclass classification.** Consider a classification problem with K classes ($K \in \mathbb{N}$). For any domain point $\hat{\mathbf{x}} \in \mathbb{R}^n$, we want to output not just the class itself, but probabilities that $\hat{\mathbf{x}}$ belongs to each of the classes. Thus, our prediction rule is a vector function,

$$\mathbf{h}_\Theta(\hat{\mathbf{x}}) = \begin{bmatrix} h_{\boldsymbol{\theta}_0}(\hat{\mathbf{x}}) & \cdots & h_{\boldsymbol{\theta}_{K-1}}(\hat{\mathbf{x}}) \end{bmatrix} : \mathbb{R}^{1\times n} \to \mathbb{R}^{1\times K}, \quad \text{where} \quad h_{\boldsymbol{\theta}_k} : \mathbb{R}^{1\times n} \to \mathbb{R}, \tag{1}$$

$$h_{\boldsymbol{\theta}_k}(\hat{\mathbf{x}}) \geq 0 \quad \text{and} \quad \sum_{k=0}^{K-1} h_{\boldsymbol{\theta}_k}(\hat{\mathbf{x}}) = 1 \quad \forall \hat{\mathbf{x}} \in \mathbb{R}^{1\times n}, \quad \text{and} \tag{2}$$

$$\Theta = \begin{bmatrix} \boldsymbol{\theta}_0 & \cdots & \boldsymbol{\theta}_{K-1} \end{bmatrix} \in \mathbb{R}^{(n+1)\times K}, \quad \text{where} \quad \boldsymbol{\theta}_k \in \mathbb{R}^{(n+1)\times 1}, \quad k = 0, \ldots, K-1. \tag{3}$$

Equation (2) ensures that $\mathbf{h}_\Theta(\hat{\mathbf{x}})$ is indeed a vector of probabilities (i.e. nonnegative and sum up to 1). Equation (3) defines the parameter Θ as an $(n+1) \times K$ matrix to introduce the bias in the first row. Specifically, we take a composition of the homogenised affine function and the softmax function to ensure Equation (2) holds for any $\Theta \in \mathbb{R}^{(n+1)\times K}$,

$$h_{\boldsymbol{\theta}_k}(\hat{\mathbf{x}}) = \frac{\exp(\langle \mathbf{x}, \boldsymbol{\theta}_k \rangle)}{\sum_{k'=0}^{K-1} \exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k'} \rangle)}, \quad k = 0, \ldots, K-1, \quad \text{where} \quad \mathbf{x} = \begin{bmatrix} 1 & \hat{\mathbf{x}} \end{bmatrix} \in \mathbb{R}^{1\times(n+1)}. \tag{4}$$

Note that the inner product $\langle \mathbf{x}, \boldsymbol{\theta} \rangle$ is independent of row or column orientation of the vectors $\mathbf{x}$ and $\boldsymbol{\theta}$, but the chosen orientations are more convenient in Python.

A given dataset is also labelled with vector-valued labels: $\mathbf{D} = (\hat{\mathbf{X}}, \mathbf{Y})$, where

$$\hat{\mathbf{X}} = \begin{bmatrix} \hat{\mathbf{x}}_0 \\ \vdots \\ \hat{\mathbf{x}}_{m-1} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_0 \\ \vdots \\ \mathbf{y}_{m-1} \end{bmatrix}, \quad \text{and} \quad \mathbf{y}_i = \begin{bmatrix} 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{1\times K}, \tag{5}$$

for $i = 0, \ldots, m-1$, where the 1 appears in the position corresponding to the class of $\hat{\mathbf{x}}_i$.

In other words, in the *given* data we assume that each domain point is labelled with a unique class with 100% probability.

Since both $\mathbf{h}_\Theta(\hat{\mathbf{x}})$ and $\mathbf{y}$ are probability distributions, a suitable loss between them is the cross-entropy:

$$\tilde{\ell}_{\hat{\mathbf{x}},\mathbf{y}}(\Theta) = \ell(\mathbf{h}_\Theta(\hat{\mathbf{x}}), \mathbf{y}) = -\sum_{k=0}^{K-1} y_k \log h_{\boldsymbol{\theta}_k}(\hat{\mathbf{x}}). \tag{6}$$

This gives the standard empirical risk of the entire dataset:

$$L_{\mathbf{D}}(\Theta) = -\frac{1}{m} \sum_{i=0}^{m-1} \sum_{k=0}^{K-1} y_{i,k} \log h_{\boldsymbol{\theta}_k}(\hat{\mathbf{x}}_i). \tag{7}$$

1.2. **Calculation of the loss gradient.** Firstly, we compute

$$\log h_{\boldsymbol{\theta}_k}(\widehat{\mathbf{x}}) = \langle \mathbf{x}, \boldsymbol{\theta}_k \rangle - \log \left[\sum_{k'=0}^{K-1} \exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k'} \rangle) \right]$$

from Equation (4). The partial derivative in the parameter component θ_{j_*,k_*} is given by

$$\begin{aligned} \frac{\partial \log h_{\boldsymbol{\theta}_k}(\widehat{\mathbf{x}})}{\partial \theta_{j_*,k_*}} &= x_{j_*}\delta_{k,k_*} - \frac{1}{\sum_{k'=0}^{K-1}\exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k'} \rangle)} \sum_{k'=0}^{K-1} \exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k'} \rangle) x_{j_*}\delta_{k',k_*} \\ &= x_{j_*}\delta_{k,k_*} - \frac{\exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k_*} \rangle)}{\sum_{k'=0}^{K-1}\exp(\langle \mathbf{x}, \boldsymbol{\theta}_{k'} \rangle)} x_{j_*} \\ &= x_{j_*}\left(\delta_{k,k_*} - h_{\boldsymbol{\theta}_{k_*}}(\widehat{\mathbf{x}})\right), \end{aligned}$$

where

$$\delta_{k,k'} = \begin{cases} 1, & k = k', \\ 0, & \text{otherwise.} \end{cases}$$

The partial derivative of the total empirical risk now becomes

$$(\nabla L_{\mathbf{D}}(\Theta))_{j_*,k_*} = \frac{\partial L_{\mathbf{D}}(\Theta)}{\partial \theta_{j_*,k_*}} = \frac{1}{m}\sum_{i=0}^{m-1}\left[x_{i,j_*}\left(\left(\sum_{k=0}^{K-1} y_{i,k}\right) h_{\boldsymbol{\theta}_{k_*}}(\widehat{\mathbf{x}}_i) - y_{i,k_*}\right)\right],$$

for $j_* = 0, \ldots, n$ and $k_* = 0, \ldots, K-1$.

Note that since $\mathbf{y}_i$ is a vector of probabilities, $\sum_{k=0}^{K-1} y_{i,k} = 1$ can be omitted. Therefore,

$$(\nabla L_{\mathbf{D}}(\Theta))_{j_*,k_*} = \frac{1}{m}\sum_{i=0}^{m-1}\left[x_{i,j_*}\left(h_{\boldsymbol{\theta}_{k_*}}(\widehat{\mathbf{x}}_i) - y_{i,k_*}\right)\right]. \tag{8}$$

1.3. **Gradient Descent (GD) with data splitting and early stopping.** Since the minimal cross-entropy loss may not be exactly zero (especially for non-separable data), we cannot stop GD by thresholding the loss itself or its gradient. Instead, we split the entire data into training and test sets, and employ the early stopping in GD, as shown in Algorithm 1.

Algorithm 1 Gradient Descent (GD) method with early stopping

Require: Training dataset $\mathbf{D}_{train} = (\widehat{\mathbf{X}}_{train}, \mathbf{Y}_{train})$, test dataset $\mathbf{D}_{test} = (\widehat{\mathbf{X}}_{test}, \mathbf{Y}_{test})$, initial guess Θ_0, constant learning rate $t > 0$, iteration gap $p \in \mathbb{N}$, loss reduction threshold $q > 0$, maximal number of iterations $S \in \mathbb{N}$.

1: **for** $s = 0, 1, \ldots, S-1$ **do**
2: **if** $s \geq p$ **and** $L_{\mathbf{D}_{test}}(\Theta_s) > q \cdot L_{\mathbf{D}_{test}}(\Theta_{s-p})$ **then**
3: **break**
4: **end if**
5: Compute $G = \nabla L_{\mathbf{D}_{train}}(\Theta_s) \in \mathbb{R}^{(n+1)\times K}$ as shown in (8).
6: Update the iterate $\Theta_{s+1} = \Theta_s - tG$.
7: **end for**
8: **return** Θ_{s_*} where $s_* = \arg\min_{\ell=0,\ldots,s} L_{\mathbf{D}_{test}}(\Theta_\ell)$.

1.4. **Principal Component Analysis (PCA) for large data.** Some data may have a rather large n, e.g. thousand. Direct classification of such $\widehat{\mathbf{x}}$ is slow and prone to overfitting. Instead, we can first try to compress the data with PCA.

Note though that $\widehat{\mathbf{x}}$ here is a row vector, in contrast to column vectors in the lecture notes. So we need to transpose the vectors in Theorem 3.11 of the lectures and solve

$$\underset{U\in\mathbb{R}^{n\times r}, U^\top U = I}{\arg\max} \operatorname{trace}\left(U^\top A U\right), \quad \text{where} \quad A = \frac{1}{m}\sum_{i=0}^{m-1} \widehat{\mathbf{x}}_i^\top \widehat{\mathbf{x}}_i \in \mathbb{R}^{n\times n}. \tag{9}$$

Having found $U \in \mathbb{R}^{n\times r}$ as the r eigenvectors corresponding to r largest eigenvalues of A, we compute the compressed vectors also in the transposed form,

$$\widehat{\mathbf{z}} = \widehat{\mathbf{x}}U, \qquad \forall\, \widehat{\mathbf{x}} \in \mathbb{R}^{1\times n}.$$

Finally, we construct the prediction rule of $\widehat{\mathbf{z}}$:

$$h_{\boldsymbol{\theta}_k}(\widehat{\mathbf{z}}) = \frac{\exp(\langle \mathbf{z}, \boldsymbol{\theta}_k\rangle)}{\sum_{k'=0}^{K-1}\exp(\langle \mathbf{z}, \boldsymbol{\theta}_{k'}\rangle)}, \quad \text{where} \quad \mathbf{z} = \begin{bmatrix}1 & \widehat{\mathbf{z}}\end{bmatrix} \in \mathbb{R}^{1\times(r+1)}, \tag{10}$$

and $\boldsymbol{\theta}_k \in \mathbb{R}^{(r+1)\times 1}$ for $k = 0, \ldots, K-1$.

2. Coursework

Preparation of files.

- First click Noteable server link on Moodle. When on Noteable, click "Git → Clone a Repository", enter `https://github.com/james-m-foster/MA50290_24.git` and click "Clone". This will create a folder `MA50290_24`.

  Unfortunately the underscore _ does not copy correctly from LaTex pdfs, so you may have to type the "`MA50290_24.git`" part of the link.
- You can update the `MA50290_24` folder by clicking "Git → Pull from Remote".
- In any case, you should now have a folder `MA50290_24/Coursework` on the Noteable server with the skeleton notebook `CW.ipynb` and the data file `CW.npz`.

  Currently the notebook contains:
  - a function `synthetic_data(m=300)` which creates a random synthetic classification dataset `Xh`$= \widehat{\mathbf{X}} \in \mathbb{R}^{m\times 2}$, for $n = 2$, and `Y`$= \mathbf{Y} \in \mathbb{R}^{m\times 3}$, for $K = 3$.
  - A function `extend_X(Xh)` which augments domain points as per Equation (4),

$$\mathbf{X} = \begin{bmatrix}1 & \widehat{\mathbf{x}}_0 \\ \vdots & \vdots \\ 1 & \widehat{\mathbf{x}}_{m-1}\end{bmatrix}.$$

  - A function `plot_2D_data(Xh, Y, Theta=0)` to plot $(\widehat{\mathbf{X}}, \mathbf{Y})$ and $\mathbf{h}_\Theta(\widehat{\mathbf{x}})$ from `synthetic_data` on a plane, colouring points from $\widehat{\mathbf{X}}$ with their class labels from $\mathbf{Y}$, and the probability values from $\mathbf{h}_\Theta(\widehat{\mathbf{x}})$ with gray scale (white=1, black=0). When `Theta=0` or $\mathbf{h}_\Theta$ is not implemented, all of the probabilities are set to $1/2$.
  - Unit test functions `test_h1`, ..., `test_grad3` for the prediction rule, loss function and its gradient. Experiments with these unit tests will **not** be marked, but you may find them helpful to debug your code.
- Rename `CW.ipynb` to a name that clearly identifies you as the author, such as `smith.ipynb`. This will be the notebook where all questions should be implemented.

Q1: Softmax feature function. Implement a Python function `softmax(W)` which takes as input a numpy array `W`$= W \in \mathbb{R}^{M\times K}$, and returns a numpy array `P`$= P \in \mathbb{R}^{M\times K}$ with elements,

$$P_{i,k} = \frac{\exp(W_{i,k})}{\sum_{k'=0}^{K-1}\exp(W_{i,k'})}, \qquad i = 0, \ldots, M-1, \quad k = 0, \ldots, K-1.$$

[1 point]

Q2: Prediction rule. Implement a Python function `h(Xh,Theta)` which takes as input numpy arrays `Xh`$= \widehat{\mathbf{X}} \in \mathbb{R}^{m\times n}$ and `Theta`$= \Theta \in \mathbb{R}^{(n+1)\times K}$, and returns a numpy array `H`$= H \in \mathbb{R}^{m\times K}$ with elements

$$H_{i,k} = \mathbf{h}_{\boldsymbol{\theta}_k}(\widehat{\mathbf{x}}_i), \qquad i = 0, \ldots, m-1, \quad k = 0, \ldots, K-1,$$

where Θ, $\mathbf{h}_{\boldsymbol{\theta}_k}(\widehat{\mathbf{x}})$ and $\widehat{\mathbf{X}}$ are defined by Equations (3), (4) and (5) respectively. You should use the functions `softmax` from Q1 and `extend_X` provided.

Test your function `h` for correctness by running the unit test functions `test_h1`, `test_h2`, and `test_h3` and making sure that none of these functions produces an error. [2 points]

Q3: Loss function. Implement a Python function `loss(Xh, Theta, Y)` which takes as input numpy arrays `Xh` $= \widehat{\mathbf{X}} \in \mathbb{R}^{m\times n}$, `Theta` $= \Theta \in \mathbb{R}^{(n+1)\times K}$, and `Y` $= \mathbf{Y} \in \mathbb{R}^{m\times K}$ and returns the value (single real number) of the empirical risk defined in (7). The matrix $\mathbf{Y}$ is defined in (5). Use the function `h` implemented in Q2.

Test your function `loss` for correctness by running the unit test functions `test_loss1`, `test_loss2`, and `test_loss3` and making sure that none of these functions gives an error.

[2 points]

Q4: Gradient of the loss function. Implement a Python function `grad_loss(Xh, Theta, Y)` which takes as input the same arguments as the `loss` function in Q3. The function `grad_loss` should return a numpy array `G` $= G \in \mathbb{R}^{(n+1)\times K}$ with elements of the gradient

$$G_{j_*,k_*} = \frac{\partial L_{\mathbf{D}}(\Theta)}{\partial \theta_{j_*,k_*}} = \frac{1}{m}\sum_{i=0}^{m-1}\left[x_{i,j_*}\left(h_{\boldsymbol{\theta}_{k_*}}(\widehat{\mathbf{x}}_i) - y_{i,k_*}\right)\right], \qquad j_* = 0,\ldots,n, \quad k_* = 0,\ldots,K-1,$$

as defined in (8).

Test your function `grad_loss` for correctness by running the unit test functions `test_grad1`, `test_grad2`, and `test_grad3` and making sure that none of these functions gives an error.

[3 points]

Q5: Data splitting. Implement a Python function `data_split(X,Y, Mtest=50)` that takes as input numpy arrays `X` $\in \mathbb{R}^{M\times N}$ and `Y` $\in \mathbb{R}^{M\times K}$, and an integer number `Mtest` with the default value of 50, and returns numpy arrays `Xtrain`, `Ytrain`, `Xtest` and `Ytest` that are random non-overlapping subsets of `X` and `Y`, created as follows:

- Assemble an index array `I`$= [0, 1, \ldots, M-1]$.
- Shuffle `I` randomly.
- Split `I` into `Itest` $= \left[\mathtt{I}_0, \ldots, \mathtt{I}_{\mathtt{Mtest}-1}\right]$ and `Itrain` $= \left[\mathtt{I}_{\mathtt{Mtest}}, \ldots, \mathtt{I}_{M-1}\right]$.
- Let `Xtrain` $=$ `X`[`Itrain`], `Ytrain` $=$ `Y`[`Itrain`], `Xtest` $=$ `X`[`Itest`], `Ytest` $=$ `Y`[`Itest`].

[2 points]

Q6: Gradient descent. Implement a Python function `gd(Xtrain, Ytrain, Xtest, Ytest, Theta0, t=1, p=100, q=0.99, S=1000)` with the following inputs:

- `Xtrain`, `Ytrain`, `Xtest`, `Ytest`: numpy arrays as in the output of Q5.
- `Theta0`: initial guess Θ_0 as a numpy array of shape `(n+1, K)`.
- `t`: learning rate $t > 0$ with the default value 1.
- `p`: iteration gap $p \in \mathbb{N}$ with the default value 100.
- `q`: loss reduction threshold $q > 0$ with the default value 0.99.
- `S`: maximal number of iterations $S > 0$ with the default value 1000.

The function `gd` should implement Algorithm 1 and return Θ_{s_*} as a numpy array of shape `(n+1, K)`, the corresponding iteration number s_*, and a numpy array of test losses at all iterations carried out until the algorithm has stopped, $[L_{\mathbf{D}_{test}}(\Theta_0), \ldots, L_{\mathbf{D}_{test}}(\Theta_s)]$.

Note: the minimisation $s_* = \arg\min_{\ell=0,\ldots,s} L_{\mathbf{D}_{test}}(\Theta_\ell)$ can be realised in the course of gradient descent iterations without storing all $\Theta_1, \ldots, \Theta_s$. Full points will be awarded for correct code that stores only three instances of Θ: the initial guess Θ_0, the current iterate Θ_s , and the best candidate Θ_{s_*}. 1 point will be deducted for storing all $\Theta_0, \ldots, \Theta_s$.

[3 points]

Q7: Testing GD on synthetic data. Create `Xh` and `Y` using the `synthetic_data` function (with the default $m = 300$). **Split** the data `Xh`, `Y` into the training dataset `Xtrain`, `Ytrain` and the test dataset `Xtest`, `Ytest` using the function `data_split` with the default `Mtest`=50.

Vary t in the range $1/4, 1/2, 1, 2, 4$, and for each t **run** the `gd` function with the `Xtrain`, `Ytrain`, `Xtest`,`Ytest` you have just created, initial guess $\Theta_0 = 0 \in \mathbb{R}^{3\times 3}$, default p and q, and the maximal number of iterations $S = 10^4$.

For each value of t, **plot** the test loss computed by `gd` as a function of the iteration number in the logarithmic scale for the loss. Make sure you label your plot.

Run your code for this question (starting from creating `Xh` and `Y`) a few times (e.g. 5-10) for each t, and **find** the largest t in the given range for which the test loss looks monotonically decreasing for all runs. Record also the iteration number s when the GD algorithm has stopped, and the iteration number s_* of the smallest test loss.

Now for the chosen value of t and the corresponding value of Θ_{s_*} from one of the runs at this t, **plot** the data `Xh`, `Y` and the predicted class probabilities $\mathbf{h}_{\Theta_{s_*}}(\widehat{\mathbf{x}})$ using the `plot_2D_data` function.

Write a report (half to one page should suffice) summarising your results:

- What is the largest t you have chosen that still gives a monotone test loss?
- How close is the best iteration number s_* to the last iteration number s on average for this t? Is this to be expected? Why?
- What are the smallest and largest last iteration numbers s you observed for this t over different runs? Can s be less than 100 with the given parameters?
- How accurate is the prediction for this t? Discuss both the visual perception of the prediction plot and some further quantitative indicator that you should suggest and compute.
- What is the smallest test loss (over all t and runs) you observed? Why is it convenient to use the stopping criterion as in Algorithm 1?

[6 points]

Q8: Loading and PCA of spectromicroscopy data. The file `CW.npz` contains:

- `Xh`: a 300×1000 matrix of possibly noisy and contaminated X-ray absorption spectra of several chromium compounds.
- `Y`: a 300×3 matrix of labels for these spectra.
- `Xhnew`: a $33^2\times1000$ matrix of spectra taken at 33×33 pixels of an unknown specimen.

Write a Python code to read `Xh`, `Y` and `Xhnew` from the file `CW.npz`.

Compute the PCA of `Xh` with $r = 3$ according to Equation (9). Store the corresponding compressed vectors $\widehat{\mathbf{z}}_i = \widehat{\mathbf{x}}_i U$, $i = 0, \dots, 299$, into a 300×3 matrix `Zh`.

Compress the specimen data `Xhnew` into a $33^2 \times 3$ matrix `Zhnew` using the same matrix U computed from `Xh`.

[3 points]

Q9: Classification of spectromicroscopy data. Write Python code to split `Zh` and `Y` into `Ztrain`, `Ytrain`, `Ztest` and `Ytest` using the `data_split` function with `Mtest`=50.

Vary t in the range $1/4, 1/2, 1, 2, 4$, and for each t **run** the `gd` function with data `Ztrain`, `Ytrain`, `Ztest` and `Ytest`, initial guess $\Theta_0 = 0 \in \mathbb{R}^{4\times3}$, default p and q, and $S = 10^5$.

For each t, **plot** the test loss as a function of the iteration number. **Find** the largest t in the given range for which the test loss looks decreasing monotonically.

For the value of t and Θ_{s_*} that you have just found, **compute** a $33^2 \times 3$ matrix of predicted probabilities

$$\texttt{Hnew} = \begin{bmatrix} \mathbf{h}_{\Theta_{s_*}}(\hat{\mathbf{z}}_0) \\ \vdots \\ \mathbf{h}_{\Theta_{s_*}}(\hat{\mathbf{z}}_{33^2-1}) \end{bmatrix}, \quad \text{where} \quad \begin{bmatrix} \hat{\mathbf{z}}_0 \\ \vdots \\ \hat{\mathbf{z}}_{33^2-1} \end{bmatrix} = \texttt{Zhnew}.$$

Let

$$\mathbf{K}_{i_1,i_2} = \arg\max_{k=0,1,2} \texttt{Hnew}_{i,k}, \quad \text{where} \quad i = 33 \cdot i_1 + i_2, \quad i_1, i_2 = 0, \dots, 32,$$

be the class labels for each pixel of the specimen image. **Plot** the matrix $\mathbf{K}$ as an image with the default colormap. Make sure you label your plots.

Write a report (a few sentences should suffice) summarising your results:

- What is the maximal t you have chosen?
- How reproducible is the specimen classification image $\mathbf{K}$ if you run this code a few times? (e.g. 5-10 times)
- How does the behaviour of GD compare overall to that in Q7?

[3 points]