MAT4373

Final project

1 PART 1

The first part of the final project is practical. Pick one of the following two or something else that excites you (subject to my oprobation).

What to submit The project should be implemented using Python. Your report should be in PDF format and preferably typeset using LaTeX. If you do, submit your `.tex` file as well. If you don't know how to use LaTeX already I highly suggest you take the opportunity to learn it. It will save you many headaches when laying out text and figures (although it is known to create different kinds of headaches from time to time). Using the site Overleaf is probably the easiest way to get into it with lots of templates.

Reproducibility counts! We should be able to obtain all the graphs and figures in your report by running your code. The only exception is that you may pre-download the images (how you did that, including the code you used to download the images, should be included in your submission.) Submissions that are not reproducible will not receive full marks. If your graphs/reported numbers cannot be reproduced by running the code, you may be docked up to 20%. (Of course, if the code is simply incomplete, you may lose even more.) Suggestion: if you are using randomness anywhere, use a seed to remove randomness. For example in Numpy that would be `numpy.random.seed()`.

Important: Readability counts! If your code isn't readable or your report doesn't make sense, they are not that useful. You will lose marks for those things.

1.1 OPTION A: Face recognition and gender classification with K-nearest neighbors

For this project, you will build a a system for face recognition and gender classification, and test it on a large(-ish) dataset of faces, getting practice with data-science- flavour projects along the way.

The Input You will work with a subset of the FaceScrub dataset. The subset of male actors is here and the subset of female actors is here. The dataset consists of URLs of images with faces, as well as the bounding boxes of the faces. The format of the bounding box is as follows (from the FaceScrub readme.txt file):

The format is x1,y1,x2,y2, where (x1,y1) is the coordinate of the top-left corner of the bounding box and (x2,y2) is that of the bottom-right corner, with (0,0) as the top-left corner of the image.Assuming the image is represented as a Python NumPy array I, a face in I can be obtained as I[y1:y2, x1:x2].

You may find it helpful to use and/or modify the script `get_data.py` for downloading the image data.

At first, you should work with the faces of the following actors: act = ['Gerard Butler', 'Daniel Radcliffe', 'Michael Vartan', 'Lorraine Bracco', 'Peri Gilpin', 'Angie Harmon']

For this project, you should crop out the images of the faces, convert them to grayscale, and resize them to 32x32 before proceeding further. You should use scipy.misc.imresize to scale images (or updated/similar function); and you can use `rgb2gray.py` to convert RGB images to grayscale images.

TIP: It's better to downscale rather than upscale; check visually your images look ok.

Part 1 (10%) Describe the dataset of faces. In particular, provide at least three examples of the images in the dataset, as well as at least three examples of cropped out faces. Comment on the quality of the annotation of the dataset: are the bounding boxes accurate? Can the cropped-out faces be aligned with each other?

Part 2 (5%) Separate the dataset into three non-overlapping parts: the training set (100 face images per actor), the validation set (10 face images per actor), and the test set (10 face images per actor). For the report, describe how you did that (any method is fine). The training set will contain faces whose labels you assume you know and will give to your model. The test set and the validation set will contain faces whose labels you pretend to not know and will attempt to determine using the data in the training set. You will use the performance on the validation set to tune K (the number of neighbors), and you will then report the final performance on the test set.

Part 3 (45%) Write code to recognize faces from the set of faces act using K-nearest neighbors (note that face recognition is a classification task!). Determine the K using the validation set, and the report the performance on the test set. Here and elsewhere in the project, use the L_2 distance between the flattened images of cropped-out faces in order to find the nearest neighbors. Flattening an image means converting it to a vector by listing all entries of one row, then the next etc.

Note: the reason you need to use both a validation set and a test set is that picking the best K by using the validation set and then reporting the performance on the same set makes it so that you report artificially high performance. A better measure is obtained by picking the parameters of the algorithm using a validation set, and then measuring the performance on a separate test set. The performance you obtain will likely be around 50% or a little higher. In addition to the performance on the test set, in your report, display 5 failure cases (where the majority of the K nearest neighbors of a test face are not the same person as the test person). Display both the face to be recognized and its 5 nearest neighbors.

Part 4 (10%) Plot the performance on the test, train, and validation sets of the K-nearest-neighbors algorithm for various K. Make sure the axes of your graphs are correctly labelled. Explain why the plots look the way they do.

Part 5 (20%) Write code to determine, for every face in a set of faces, the gender of the person using K nearest neighbors. This should work in the same way as face recognition, except instead of each face being assigned a name, each face is considered to be simply either male or female. Again, use your validation set to select a K for the best performance (i.e., the proportion of faces whose gender was classified correctly), report the performance for the different K's for the validation set, and then report the results on the test set for the K that works best for the validation set. For this part, you should still use the set of actors act for both the test and the validation set. The performance you obtain will likely be around 80% or a little higher

Part 6 (10%) Now, evaluate the implementation in Part 5 on faces of actors who are not in the training set (i.e., actors other than the people in the set **act**.) Discuss the difference between the performance in Part 5 and the performance in Part 6.

1.2 OPTION B: Handwritten digit recognition using neural networks

For this project, you will build and train two systems for digit classification: one neural network and one linear classifier.

Note: for the classifiers and settings suggested in this project, overfitting is not a significant problem so using a validation set is up to you.

The Input You will work with the famous MNIST dataset. The dataset is available in easy-to-read-in format here, with the digits already seprated into a training and a test set. I recommend you divide the data by 255.0 (note: the .0 is important) so that it's in the range 0..1.

About code There's some code available in the course contents (`mnist_handout.py` and data `snapshot50.pkl`). The code is not meant to be run as is. I am simply providing some functions and code snippets to make your life easier. For beginers I recommend instead using online rescources for code with more explainations (ask me for suggestions). Ultimately must write your own implementation but you can use snippets if you cite the source.

Part 1 (5%) In your report, include 10 images of each of the digits. You may find matplotlib's subplot useful.

Part 2 (5%) Implement a multiclass linear classifier. Specifically, use

$$o_i = \sum_j W_{ij} x_j + b_i$$

and apply a `softmax` to the o_i's. Notice that this is different from the SVM classifiers we have seen. The W is a matrix instead of a vector and we are not just interested in the sign of the output. Instead the output will be a probability distribution on the 10 classes with the chosen class the one with the highest probability.

Part 3 (10%) For this project, we will be using the the sum of the negative log-probabilities of the correct answer for the N training cases under consideration as the loss function. (I.e., the negative log-likelihood of the training set.) Implement a function that computes the gradient of this loss function with respect to the parameters of the network (W and b), for a given subset of training cases. Include a listing of your implementation in your report for this Part. You can, but do not have to, vectorize your code (ie use vector and matrix operations instead of loops and sums. The advantage of doing this is mostly to take advantage of GPU training).

Part 4 (5%) Verify that you are computing the gradient in Part 3 correctly by computing it both using your function and using a finite-difference approximation for several coordinates of the W and b. In your report for this Part, include the listing of the code that you used and the output of the code (which should include both your precise gradient and the approximation).