IMPERIAL

Department of Mathematics
MATH60026 - MATH70026
Methods for Data Science
Years 3/4/5

Coursework 1 - Deadline: Friday, 21 February 2025 at 1 pm.

General instructions - please read carefully

The goal of this coursework is to analyse a dataset using several tools and algorithms introduced in the lectures, which you have also studied in detail through the weekly Python notebooks containing the computational tasks. You will solve the tasks in this coursework using Python.

You are allowed to use:

- Python code that you have developed in your coding tasks;
- any other basic mathematical functions contained in `numpy`;
- `pandas` to build and clean up data tables;
- `tqdm` as a utility to visually track the progress of for-loops.

You are *not* allowed to use:

- any model-level or automatic differentiation Python packages (*e.g.*, scikit-learn, statsmodels, PyTorch, JAX, Tensorflow/Keras, etc);
- ready-made code found anywhere online;
- conversational AI tools such as ChatGPT, Microsoft Bing, GitHub Copilot, Gemini etc. to generate code and written answers.

Needless to say, your submission must be your own individual work: *You may discuss the analysis with your colleagues but code, written answers, figures and analysis must be written independently on your own. The Department uses code profiling and tools such as Turnitin to check for plagiarism of any sort.* ***Plagiarism is a major form of academic misconduct.***

Marks

This coursework is worth **40% of your total mark for the course.**

Mastery component: The mastery component amounts to 20% of this CW: two tasks (**1.4** and **3.2**) have one version for 3rd year BSc students only and one version for MSc and 4th year MSci students only (Mastery component) - pay attention to this and choose the right option in each of those tasks! You should only answer the version that applies to your case.

For general guidance about writing your solutions and marking scheme, please read the document "Guidelines about writing your solutions" in the folder "Examples of Previous Coursework & Guidelines".

Submission

Files: For the submission of your coursework, you need to save **two documents**:

- a **Jupyter notebook (file format: ipynb)** with all your tasks clearly labelled. You should use the template notebook called *CID_Coursework1.ipynb* provided on Blackboard (folder 'Coursework/Coursework 1'). The notebook should have clear headings to indicate the answers to each question, *e.g.* 'Task 1.1'.

The notebook should contain the cells with your code and their output, plus some brief text explaining your calculations, choices, mathematical reasoning, and discussion of results. (Important: Before submitting you must run the notebook, with all cells executed sequentially and the outputs of the cells must be printed). You can use Google Colab or develop your Jupyter notebook in the local Python environment installed on your computer.

- Once you have executed all cells in your notebook and their outputs are printed, save the notebook as an **html file** (with name *CID_Coursework1.html*). Your ipynb file must produce all the output that appears in your html file, *i.e.*, make sure you have run all cells in the notebook before exporting the html.

The submission of your coursework must consist of **two items,** uploaded **separately**:

1) *A **single zip folder** containing your Jupyter notebook as an **ipynb file** and your notebook exported as an **html file**. Name your zip folder 'CID_Coursework1.zip', where CID is your student CID, e.g. 123456_Coursework1.zip.*
2) *The html file, named CID_Coursework1.html, which will be used for the plagiarism check.*

The submission should consist only of these 2 items - **Do not submit multiple files.**

Do not put your name on the files you submit (only the CID), because the marking must be carried out preserving your anonymity.

Drop Boxes: The submission is done **online via Blackboard,** using the drop boxes inside the folder 'Coursework' on Blackboard.

- If you are a 3rd year BSc student: use the 'Coursework Drop Boxes' inside the folder 'Coursework'. The html file must be uploaded to 'Coursework 1/Coursework 1 Drop Box Spring 25 - HTML', the zip folder must be uploaded to 'Coursework 1/Coursework 1 Drop Box Spring 25 - ZIP File'.

- If you are a 4th year MSci or MSc student: use the 'Mastery Coursework Drop Boxes' inside the folder 'Coursework'. The html file must be uploaded to 'Mastery Coursework 1/Coursework 1 Mastery Drop Box Spring 25 - HTML', the zip file to 'Mastery Coursework 1/Coursework 1 Mastery Drop Box Spring 25 - ZIP File'.

Make sure you submit to the right drop box on Blackboard - any mistake in the submission folder will cause a delay in the release of your mark.

Recommendations about online submissions:

- *There are known issues with particular browsers (or settings with cookies or popup blockers) when submitting to Turnitin. If the submission 'hangs', please try another browser.*
- *You should also **check that your files are not empty or corrupted after submission**.*
- ***To avoid last minute problems** with your online submission, we recommend that you **upload versions of your coursework early on**, before the deadline. You will be able to update your coursework until the deadline, but having this early version provides you with some safety backup. For the same reason, keep **backups of your work**, e.g. save regularly your notebook with its outputs as an .html file, which can be useful if something unpredicted happens just before the deadline.*
- *If you have any issue with the submission, or you realise you have submitted your work to the wrong drop box, please contact directly the UGMathsOffice at maths-student-office@imperial.ac.uk or your MSc programme administrator, in such a way that they can help you solve the issue.*
- *If you need an extension, or happen for any reason to submit your work late, please make a request for mitigating circumstances directly on ZINC.*

  *For these last two points, **do not contact us - we, as lecturers, are not able to grant extensions nor to make changes in the submission folders! We only get to see anonymised submissions.***

Coursework 1

In this coursework, you will work with a dataset on **airfield statistics**, which is made available inside the folder 'Coursework/Coursework 1/Data' on Blackboard. With this dataset you will perform supervised learning tasks of **classification** (Task 1 and 2) as well as **regression** (Task 3).

Dataset: The airfield statistics dataset contains information on weather, aircraft traffic, and runway surface conditions collected over time at the airfield for monitoring and forecasting purposes. Specifically, it contains **6 variables (or "data features") to use as predictors in all the tasks**: `Days of airfrost`, `Precipitation`, `Sunshine hours`, `Humidity (%)`, `Wind Speed`, `Aircraft total movements`.

To guide the design of safe flight operations based on meteorological patterns, a classification system of different conditions into 4 categories was set up. You find the corresponding classification labels $(0, 1, 2, 3)$ in the column `Weather and flight condition category`, and these will be your output targets for the classification tasks. In addition, the minimal and maximal temperatures reached at the runway surface for each condition are reported, respectively, in the columns `Runway surface minimal temperature` and `Runway surface maximal temperature`. These values will be your output targets for the regression task.

The dataset to use as **the training set** consists of $N^{\text{train}} = 571$ samples and is available on Blackboard as: `airfield_statistics_train.csv`.

The dataset to use as **the test set** consists of $N^{\text{test}} = 245$ samples and is available on Blackboard as: `airfield_statistics_test.csv`.

Important:

1. The test set should **not** be used in any learning, specifically it should not be involved in any parameter training or hyperparameter tuning. The test set should be put aside and only be used a posteriori to evaluate the out-of-sample performance of your models. Only `airfield_statistics_train.csv` should be used for the cross-validation tasks.

2. Standardise the dataset **only** when the task explicitly asks for it, using mean and standard deviation of the training set to standardise both the training and test sets, as discussed in the lecture. We will explicitly use the word 'standardised' in front of training and test set when the standardisation is required.

3. For reproducibility, create a random generator with a **fixed seed** at the beginning of your notebook with the command: `rng = np.random.default_rng(0)`.

Task 1: Multi-class classification with Decision Trees and Random Forests

In this task, you will build multi-class classifiers to predict, from the 6 data features, the class label y (i.e., one of the 4 possible values $= 0, 1, 2, 3$ stored in the column `Weather and flight condition category`).

1.1 (10 marks) - Comparison of Decision Trees and k-NN with and without standardisation. Build a multi-class classifier by training a Decision Tree (DT) with `max_depth`$= 10$, `min_samples_leaf`$= 12$, using the Gini index as the loss function for each node split. Evaluate the performance of the DT on the training and test sets, using accuracy as your metric of performance.

Standardise the training and test sets, and repeat the training and testing of the DT (with the same `max_depth` and `min_samples_leaf`) on the standardised training and test sets. Compare the accuracy of the DT on the standardised training and test sets to the one obtained previously without standardisation.

Train and test a k-Nearest Neighbour (k-NN) classifier with $k = 25$, first on the original training and test sets, next on the standardised training and test sets, using accuracy as the performance metric. Compare: the accuracy of k-NN with and without standardisation; the accuracy of k-NN and the DT.

Discuss your findings, explaining also the reasons for the effect of standardisation on the performance of the DT and k-NN.

1.2 (8 marks) - Decision Trees for soft classification. Train a DT for multi-class soft classification, i.e., a DT that predicts as output, for a given input sample, the vector of probabilities that the sample belongs to each of the 4 classes. [Hint: use the node impurity.] Like in Task **1.1**, set `max_depth`$= 10$, `min_samples_leaf`$= 12$, and use the Gini index as the loss function for each node split.

Evaluate the DT's probabilistic predictions on the test set, and assess the DT's ability to discriminate between pairs of classes by considering only the probabilistic predictions for each pair and by drawing the corresponding Receiver Operating Characteristic (ROC) curve. For example, to compute the ROC curve for the discrimination between class 0 and class 1, you need to use only the probabilistic predictions of belonging to either class 0 or class 1 for the test data from these two classes. Draw the ROC curve and compute the area under the ROC for all the pairwise comparisons between classes. [Hint: you should have a total of 6 plots.] Discuss your findings.

1.3 (15 marks) - Gini importance for Random Forests. Train a Random Forest for multi-class hard classification containing $B = 20$ DTs, where each DT in the forest has `max_depth`$= 10$, `min_samples_leaf`$= 12$ and uses the Gini index as the loss function for each node split (like in Task **1.1**). Perform bagging with bootstrapped samples of size $N' = N^{\text{train}}$. For m, the number of features randomly sampled at each split, use the value recommended for classification (reported in the lecture notes).

To quantify the importance of each predictor to the final prediction, an alternative measure of variable importance is based on the change of the Gini index during the training process and is known as *Gini importance*. In essence, to quantify the importance of the n-th feature x_n, one has to consider, for each tree T_b (with $b = 1, \ldots, B$), all the nodes t that used as splitting feature j_t precisely the n-th feature (i.e., $j_t = n$), and compute the decrease in the Gini index $\Delta GI(s_t, t)$ associated to the split at that node with splitting value s_t. The definition of Gini importance, for a given feature x_n, is then:

$$\text{Gini importance}(x_n) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in T_b : j_t = n} p_t \, \Delta GI(s_t, t)$$

where $p_t = N_t / N'$ denotes the proportion of the N' training samples for tree T_b reaching node t, and the decrease in the Gini index $\Delta GI(s_t, t)$ is given by:

$$\Delta GI(s_t, t) = GI(t) - p_L GI(t_L) - p_R GI(t_R)$$

In the above, $GI(t)$ is the Gini index of node t before the split, $GI(t_L)$ and $GI(t_R)$ are the Gini indices of the left and right daughter nodes generated by the split. $p_L = N_{t_L}/N_t$ and $p_R = N_{t_R}/N_t$ stand for the proportion of the training samples at node t, N_t, that end up, respectively, in the left and right daughter node.

Compute the Gini importances of the 6 predictors, express them as a percentage of the maximally important feature, and plot them. [Hint: modify appropriately the `build_tree` function to output all the quantities that you need.] Discuss your findings.

1.4 (10 marks, BSc students only) - Out-of-Bag (OOB) misclassification error. Optimise B, the number of DTs of the Random Forest of Task **1.3** using the OOB misclassification error, scanning B over an appropriately chosen range that contains at least 10 different values. Re-train the Random Forest with the optimal B and evaluate its accuracy on the test set. Compare it to the accuracy of a single DT (from Task **1.1**) and discuss your findings.

1.4 (8 marks, MSc/4th-year students only) - Weighted Gini index. To improve the performance of a DT at discriminating classes of interest, one can resort to a *weighted* version of the Gini index. One defines a loss matrix $\mathbf{L}$, where each element $L_{qq'}$ (with $q, q' = 1, \ldots, Q$ and $Q = 4$ in our case), represents the loss incurred when an observation of the class c_q is classified as class $c_{q'}$. The weighted version of the Gini index is given by:

$$\sum_{\substack{q,q'=1 \\ q \neq q'}}^{Q} L_{qq'} \pi_q(R_\alpha) \pi_{q'}(R_\alpha)$$

where $\pi_q(R_\alpha)$ denotes, like in the lecture notes, the proportion of class c_q observations in the region (or node) R_α (similarly for $\pi_{q'}(R_\alpha)$). You can read more about this strategy in Section 9.2.4 of Hastie, Tibshirani, Friedman, *The Elements of Statistical Learning*.

Re-train the DT from Task **1.1** using this weighted Gini index with the loss matrix:

$$\mathbf{L} = \begin{vmatrix} 0 & 0.1 & 1 & 1 \\ 0.1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}$$

Evaluate all the pairwise accuracies on the test set, that is, the test accuracies considering only true and predicted labels for two specific classes (similarly to the pairwise ROC in Task **1.2**). Compare them to the pairwise accuracies on the test set that one obtains with the standard Gini index (i.e., with the DT from Task **1.1**). Discuss your findings, explaining the reasons for the effect of the loss matrix on the discrimination performance that you observe.

Task 2: Binary classification with the Huberised Support Vector Machine

In this task, you will build a binary classifier that, given the 6 data features, predicts class 0 vs the rest (i.e., belonging to class 1, 2 or 3). To this end, rename the labels 0 (stored in the column `Weather and flight condition category`) as $y = 1$, and merge all samples with labels $= 1, 2, 3$ into one class with label $y = -1$. Standardise the training and test sets in this task.

2.1 (17 marks) - Modified Huber function vs hinge loss. Consider a linear soft-margin "Huberised" Support Vector Machine (SVM) binary classifier, i.e., a linear soft-margin SVM where the penalty term is modelled via the modified Huber function L_c as follows:

$$L(\mathbf{w}, b) = \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N^{\text{train}}} L_c\left(1 - (\mathbf{x}^{(i)} \cdot \mathbf{w} + b)y^{(i)}\right), \quad \text{with} \quad L_c(z) = \begin{cases} 0 & \text{if } z \leq 0, \\ \frac{1}{2}z^2 & \text{if } 0 < z \leq c, \\ c(z - \frac{c}{2}) & \text{if } z > c. \end{cases}$$

Derive the mathematical expressions for the gradient of the loss $L(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and b. Train this Huberised SVM on the standardised training set using the mini-batch stochastic gradient descent (mini-batch SGD) algorithm with the following parameters: $c = 1$, $\lambda = 100$, a mini-batch size of 32, a learning rate of 10^{-4}, stop criterion of 0.001, and a maximum of 2000 epochs. Produce a plot to show the training convergence.

Train on the standardised training set a second linear soft-margin SVM binary classifier, but this time using the hinge loss as the penalty term with $\lambda = 100$. For the training, use mini-batch SGD with the same parameters as before (mini-batch size 32, learning rate 10^{-4}, stop criterion 0.001, maximum 2000 epochs) and produce a plot to show its convergence.

For each SVM, compute 2 quantities: the accuracy on the standardised test set; the number of training data points that fall outside the margin. Generate a plot illustrating the modified Huber function alongside the hinge loss as a function of $yf(x)$, similarly to figure 6.3 in the lecture notes for the hinge loss alone. Use this plot and the 2 quantities above to discuss and compare the two loss functions, highlighting their differences in terms of their impact on the SVM model structure and its out-of-sample performance.

2.2 (10 marks) - T-fold cross-validation. Optimise the hyperparameters c and λ of the linear soft-margin Huberised SVM from Task **2.1** via T-fold cross-validation ($T = 5$), scanning c and λ over the ranges $\{0.5, 1, 10\}$ and $\{1, 100, 10000\}$, respectively. Use mini-batch SGD with the same parameters as in Task **2.1** for the training and use accuracy as the performance metric for T-fold cross-validation. Produce plots showing the modified Huber function as a function of $yf(x)$ for the different values of c explored during the hyperparameter search.

Retrain the Huberised SVM classifier with the optimal values of c and λ and evaluate its accuracy on the standardised test set, comparing it to the one of the Huberised SVM from Task **2.1**. Evaluate also the *balanced* accuracy on the standardised test set and discuss your results.

Task 3: Regression with the Multi-Layer Perceptron

In this task, you will build a regression model that, based on the 6 data features, jointly predicts two outcome values, the minimal temperature T_{min} (contained in the column `Runway surface minimal temperature`) and the minimal temperature T_{max} (contained in the column `Runway surface maximal temperature`). You will build the regressor by implementing the Multi-Layer Perceptron (MLP). To this end, you should use NumPy alone like in your Week 5 notebook, you should not use TensorFlow/Keras, PyTorch or equivalent libraries. Create `MLP` objects from the `MLP` class provided in the Week 5 notebook, initialising them with `seed`$= 2$ everywhere in this task.

3.1 (20 marks) - MLP with a compound loss function for regression. Set up the MLP architecture for regression as follows.

Architecture of the network: Your network should have an input layer, 2 hidden layers (with 20 neurons each), followed by the output layer with two neurons (the outcome variables to predict). For both hidden layers, apply the ReLU activation function. Train the network on the training set minimising the compound loss function:

$$L = \frac{1}{N^{\text{train}}} \sum_{i=1}^{N^{\text{train}}} \left((T_{\text{min}}^{(i)} - \hat{T}_{\text{min}}^{(i)})^2 + (T_{\text{max}}^{(i)} - \hat{T}_{\text{max}}^{(i)})^2 + \lambda \max(0, \hat{T}_{\text{min}}^{(i)} - \hat{T}_{\text{max}}^{(i)}) \right) \tag{1}$$

where $T_{\text{min}}^{(i)}$ and $T_{\text{max}}^{(i)}$ are the true outcomes for sample i and $\hat{T}_{\text{min}}^{(i)}$ and $\hat{T}_{\text{max}}^{(i)}$ are the corresponding predicted outcomes, while λ is a hyperparameter. [Note: you should set the gradient of $\max(0, x)$ to zero at $x = 0$.]

Use mini-batch SGD as your optimisation method, setting the mini-batch size $= 20$. Train the network for

200 epochs, with a constant learning rate equal to 5×10^{-5}, setting $\lambda = 0, 100, 500$. For each of the 3 values of λ: track the compound loss function on the training set and plot its trend as a function of the number of epochs; compute the R^2 score of the trained MLP on the training and test sets.

Discuss the comparison of the MLPs with the 3 values of λ in terms of convergence of the training (using the plot you have produced) and performance (using the training and test-set R^2 score). Explain the observed differences by interpreting the compound loss function (1).

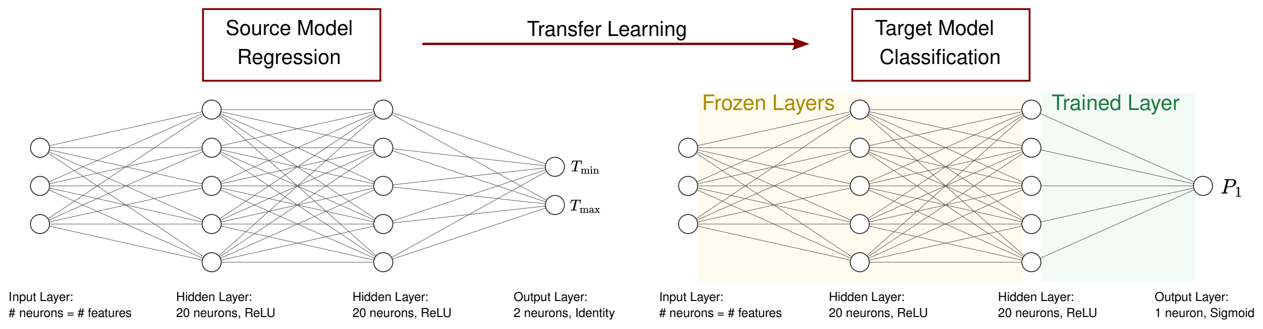
3.2 (10 marks, BSc stdents only) - Deep MLP with dropout. Train an MLP for regression with the loss function (1) (with $\lambda = 0$) and the following deeper network architecture: an input layer, 5 hidden layers (with 100, 200, 300, 100 and 100 neurons each), followed by the output layer with two neurons (again, the outcome variables to predict T_{min} and T_{max}). For all hidden layers apply the ReLU activation function. Apply dropout with rate 0.1 in the 4^{th} hidden layer. Use mini-batch SGD with mini-batch size 16 and train the network for 200 epochs, with a constant learning rate equal to 5×10^{-5}. For comparison, train a second MLP with the same architecture but without dropout.

For both MLPs: plot the loss function on the train and test set against the number of epochs and produce a histogram of the post-activations of the 4^{th} hidden layer. Discuss the comparison between the deep MLP with and without dropout, explaining the reasons for the differences that you observe in these plots.

3.2 (12 marks, MSc/4th year students only) - Transfer learning: from regression to classification. In this subtask, we come back to a classification task, but we assume that we have classification labels only for classes $= 1, 3$ (where the class labels are again the ones in the column `Weather and flight condition category`). First, produce the training and test data for this subtask by selecting the data points with class labels $= 1, 3$ and replace the class labels with $y = 1, 0$ to perform binary classification.

In this way, the size of the labelled training data is reduced from 571 to 288 samples. When we have a small sample, it can be convenient to apply strategies of *transfer learning*, which allow one to leverage pre-trained models typically for a different prediction task (called "source" models). You can read more about transfer learning in this post.

In this subtask, we apply a transfer learning strategy whereby we transfer the MLP regressor from Task **3.1** with $\lambda = 0$ (a "source" model trained on a larger training dataset for a regression task) to the new task of binary soft classification that is performed by a new MLP (the "target" model). The target MLP model is composed



of: the input layer and the 2 hidden layers (with 20 neurons each and ReLU activation) of the MLP from Task **3.1**, with their trained weights and biases at $\lambda = 0$; an output layer with one neuron (see figure). This neuron stands for P_1, the predicted probability of belonging to class 1, obtained by applying a sigmoid function in this output layer (see below).

During training, the layers inherited from the MLP from Task **3.1** stay frozen, i.e., their weights and biases are not trained, while only the weights and biases of the output layer are updated (see figure). Train the target MLP model by minimising the logistic loss for probabilistic binary classification:

$$L(\{y^{(i)}, P_1^{(i)}(\mathbf{a}^{(N_L+1)})\}) = -\frac{1}{N^{\text{train}}} \sum_{i=1}^{N^{\text{train}}} \left(y^{(i)} \log\left(P_1^{(i)}(\mathbf{a}^{(N_L+1)})\right) + (1 - y^{(i)}) \log\left(1 - P_1^{(i)}(\mathbf{a}^{(N_L+1)})\right)\right) \qquad (2)$$

where N^{train} is here meant as the size of the reduced training set of this subtask. $y^{(i)}$ is the true class label for sample i and $P_1^{(i)}$ is the probabilistic prediction for sample i given by the sigmoid activation function applied to the pre-activation $\mathbf{a}^{(N_L+1)}$ of the output unit:

$$P_1(\mathbf{a}^{(N_L+1)}) = \sigma_{\text{Sigmoid}}(\mathbf{a}^{(N_L+1)}) \qquad \text{with} \qquad N_L : \text{ Number of hidden layers}$$

[Hint: to avoid evaluating the logarithm of zero probabilities in the loss (2), you can limit the values of P_1 between ϵ and $1-\epsilon$, with $\epsilon = 10^{-7}$, using `np.clip`.] Train the network for 50 epochs, using mini-batch SGD with mini-batches of 8 data points and a constant learning rate equal to 0.001. Evaluate the classifier's probabilistic predictions on the test set and produce a histogram to visualise their distribution. Convert them to hard class assignments taking $P_1 = 0.5$ as the threshold above which data points are assigned to class 1, and evaluate the test-set accuracy.

Compare this test-set accuracy to the one of an MLP classifier with the same architecture but that does not implement transfer learning, i.e., where one trains all the weights and biases in the network. Use the same training data and training protocol (i.e., mini-batch size $= 8$, learning rate $= 0.001$, 50 epochs) as before. Discuss your findings, explaining the reasons for the observed performance of the MLP classifier with and without transfer learning.