Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: May 26, 2024
Total: 60 marks

ASSIGNMENT 1

1 What should I submit, where should I submit and by when?

Your submission for this assignment will be one PDF (.pdf) file and one ZIP (.zip) file. Instructions on how to prepare and submit these files are given below.

Assignment Package:
`https://www.cse.iitk.ac.in/users/purushot/courses/ml/2023-24-s/material/assn/assn1.zip`
Deadline for all submissions: June 12, 2024, 9:59PM IST
Code Validation Script: `https://colab.research.google.com/drive/1k0I6Ab_owwZSWv3y2tE4UbhXSUBXYgzi?usp=sharing`
Code Submission: `https://forms.gle/7TbGEHmXrWeMWuC77`
Report Submission: on Gradescope
There is no provision for "late submission" for this assignment

1.1 How to submit the PDF report file

1. The PDF file must be submitted using Gradescope in the *group submission mode*.
2. Note that unregistered auditors cannot make submissions to this assignment.
3. Make only one submission per assignment group on Gradescope, not one submission per student. Gradescope allows you to submit in groups - please use this feature to make a group submission.
4. **Ensure that you validate your submission files on Google Colab before making your submission** (validation details below). Submissions that fail to work with our automatic judge since they were not validated will incur penalties.
5. Link all group members in your group submission. If you miss out on a group member while submitting, Gradescope will think that person never submitted anything.
6. You may overwrite your group's submission as many times as you want before the deadline (submitting again on Gradescope simply overwrites the old submission).
7. Do not submit Microsoft Word or text files. Prepare your report in PDF using the style file we have provided (instructions on formatting given later).

1.2 How to submit the code ZIP file

1. Your ZIP file should contain a single Python (.py) file and nothing else. The reason we are asking you to ZIP that single Python file is so that you can password protect the ZIP file. Doing this safeguards you since even after you upload your ZIP file to your website,

no one can download that ZIP file and see your solution (since you will tell the password only to the instructor). If you upload a naked Python file to your website, someone else may guess the location where you have uploaded your file and steal it and you may get charged with plagiarism later.

2. We do not care what you name your ZIP file but the (single) Python file sitting inside the ZIP file must be named "submit.py". There should be no sub-directories inside the ZIP file – just a single file. We will look for a single Python (.py) file called "submit.py" inside the ZIP file and delete everything else present inside the ZIP file.

3. Do not submit Jupyter notebooks or files in other languages such as C/C++/R/Julia/Matlab/Java. We will use an automated judge to evaluate your code which will not run code in other formats or other languages (submissions in other languages will get a zero score).

4. Password protect your ZIP file using a password with 8-10 characters. Use only alphanumeric characters (a-z A-Z 0-9) in your password. Do not use special characters, punctuation marks, whitespaces etc in your password. Specify the file name properly in the Google form.

5. Remember, your file is not under attack from hackers with access to supercomputers. This is just an added security measure so that even if someone guesses your submission URL, they cannot see your code immediately. A length 10 alphanumeric password (that does not use dictionary phrases and is generated randomly e.g. 2x4kPh02V9) provides you with more than 55 bits of security. It would take more than 1 million years to go through all $> 2^{55}$ combinations at 1K combinations per second.

6. Make sure that the ZIP file does indeed unzip when used with that password (try
   `unzip -P your-password file.zip`
   on Linux platforms).

7. Upload the password protected ZIP file to your IITK (CC or CSE) website (for CC, log on to `webhome.cc.iitk.ac.in`, for CSE, log on to `turing.cse.iitk.ac.in`).

8. Fill in the following Google form to tell us the exact path to the file as well as the password
   `https://forms.gle/7TbGEHmXrWeMWuC77`

9. **Do not host your ZIP submission file on file-sharing services like Dropbox or Google drive. Host it on IITK servers only**. We will autodownload your submissions and GitHub, Dropbox and Google Drive servers often send us an HTML page (instead of your submission) when we try to download your file. Thus, it is best to host your code submission file locally on IITK servers.

10. While filling in the form, you have to provide us with the password to your ZIP file in a designated area. Write just the password in that area. For example, do not write "Password: helloworld" in that area if your password is "helloworld". Instead, simply write "helloworld" (without the quotes) in that area. Remember that your password should contain only alphabets and numerals, no spaces, special or punctuation characters.

11. While filling the form, give the complete URL to the file, not just to the directory that contains that file. The URL should contain the filename as well.

    (a) Example of a proper URL:
    `https://web.cse.iitk.ac.in/users/purushot/mlassn1/my_submit.zip`

(b) Example of an improper URL (file name missing):
`https://web.cse.iitk.ac.in/users/purushot/mlassn1/`

(c) Example of an improper URL (incomplete path):
`https://web.cse.iitk.ac.in/users/purushot/`

12. We will use an automated script to download all your files. If your URL is malformed or incomplete, or if you have hosted the file outside IITK in a manner that is difficult to download, then your group may lose marks.

13. Make sure you fill-in the Google form with your file link before the deadline. We will close the form at the deadline.

14. Make sure that your ZIP file is actually available at the link at the time of the deadline. We will run a script to automatically download these files after the deadline is over. If your file is missing, we will treat this as a blank submission.

15. We will entertain no submissions over email, Piazza etc. All submissions must take place before the stipulated deadline over the Gradescope and the Google form. The PDF file must be submitted on Gradescope at or before the deadline and the ZIP file must be available at the link specified on the Google form at or before the deadline.

Problem 1.1 (Cross Connection PUF). While doing some calculations in free time, Melbo realized something funny about arbiter PUFs. Recall that an arbiter PUF is a chain of k multiplexers, each of which either swaps the lines or keeps them intact, depending on what is the challenge bit fed into that multiplexer. The multiplexers each have delays which are hard to replicate but consistent. Let t^u, t^l respectively denote the time for the upper and lower signals to reach the finish line. At the finish line resides an arbiter (usually a flip-flop) which decides which signal reached first, the upper signal or the lower signal. The arbiter then uses this decision to generate the response. Although it is possible for a simple linear model to predict whether the upper signal will reach the finish line first or the lower signal (as we have seen in class), it seems that it is not so simple for a linear model to predict the time taken by the upper signal to reach the finish line, or predict the time taken by the lower signal to reach the finish line. Armed with this realization, Melbo created a new PUF variant and gave it the name **Cross-Connection PUF** or **COCO-PUF** for short.

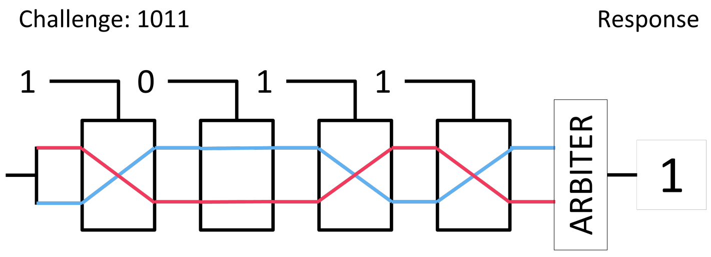

Figure 1: A simple arbiter PUF with 4 multiplexers

A COCO-PUF uses 2 arbiter PUFs, say PUF0 and PUF1 – each PUF has its own set of multiplexers with possibly different delays. Given a challenge, it is fed into both the PUFs. However, the way responses are generated is different. Instead of the lower and upper signals of PUF0 competing with each other, Melbo makes the lower signal from PUF0 compete with the lower signal from PUF1 using an arbiter called Arbiter0 to generate a response called Response0. If the signal from PUF0 reaches first, Response0 is 0 else if the signal from PUF1 reaches first, Response0 is 1. Melbo also makes the upper signal from PUF0 compete with the upper signal from PUF1 using a second arbiter called Arbiter1 to generate a response called Response1. If the signal from PUF0 reaches first, Response1 is 0 else if the signal from PUF1 reaches first, Response1 is 1. Thus, on each challenge, the COCO-PUF generates two responses instead of one response.

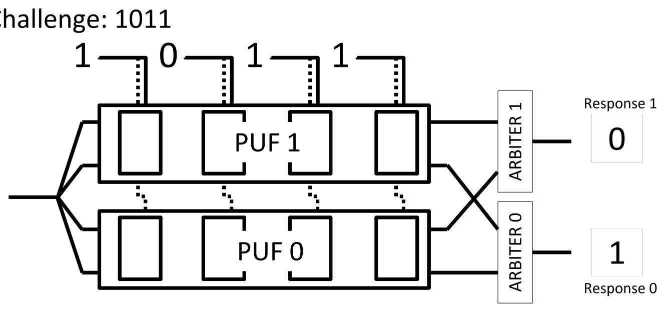

Figure 2: A COCO-PUF with 4-bit challenges and 2-bit responses

Melbo thinks that these cross connections should make it difficult for a linear model to can predict the responses if given a few thousand challenge-response pairs. Your job is to prove

Melbo wrong! You will do this by showing that there do exist linear models that can perfectly predict the responses of a COCO-PUF and these linear model can be estimated fairly accurately if given enough challenge-response pairs (CRPs).

Your Data. We have provided you with data from a COCO-PUF with 32-bit challenges. The training set consists of 40000 CRPs and the test set consists of 10000 CRPs. For each CRP, a 32-bit challenge and 2-bit response (corresponding to Response0 and Response1) is provided. Each row in the training and testing files provided to you contain 34 numbers, each number being 0 or 1. The first 32 numbers correspond to the challenge and the last two numbers correspond to the two responses. If you wish, you may create (held out/k-fold) validation sets out of this data in any way you like. Recall that if we properly encode the challenge as a 32-dim vector, then for any arbiter PUF, there exists a linear model that always predicts the correct response for that arbiter PUF. Your job is to create a new feature vector from the 32-bit challenge so that you can learn two linear models – one that can predict Response0 very accurately and the other that can predict Response1 very accurately. However, note that unlike the arbiter PUF case where the new feature vector was also 32-dimensional for 32-dimensional challenges, for a COCO-PUF the new feature vector may need to have a different dimensionality than the challenges.

Your Task. The following enumerates 6 parts to the question. Parts 1,2,3,4,6 need to be answered in the PDF file containing your report. Part 5 needs to be answered in the Python file.

1. By giving a detailed mathematical derivation (as given in the lecture slides), show how for a simple arbiter PUF, a linear model can predict the time it takes for the upper signal to reach the finish line. Specifically, give derivations for a map $\phi : \{0,1\}^{32} \to \mathbb{R}^D$ mapping 32-bit 0/1-valued challenge vectors to D-dimensional feature vectors (for some $D > 0$) so that for any arbiter PUF, there exists a D-dimensional linear model $\mathbf{W} \in \mathbb{R}^D$ and a bias term $b \in \mathbb{R}$ such that for all CRPs $\mathbf{c} \in \{0,1\}^{32}$, we have $\mathbf{W}^\top \phi(\mathbf{c}) + b = t^u(\mathbf{c})$ where $t^u(\mathbf{c})$ is the time it takes for the upper signal to reach the finish line when challenge $\mathbf{c}$ is input. Remember that $t^u(\mathbf{c})$ is, in general, a non-negative real number (say in milliseconds) and need not be a Boolean bit. $\mathbf{W}, b$ may depend on the PUF-specific constants such as delays in the multiplexers. However, the map $\phi(\mathbf{c})$ must depend only on $\mathbf{c}$ (and perhaps universal constants such as $2, \sqrt{2}$ etc). The map ϕ must not use PUF-specific constants such as delays. (8 marks)

2. What dimensionality does the linear model need to have to predict the arrival time of the upper signal for an arbiter PUF? The dimensionality should be stated clearly and separately in your report, and not be implicit or hidden away in some calculations. (2 marks)

3. Use the derivation from part 1 to show how a linear model can predict Response0 for a COCO-PUF. As in part 1, give an explicit map $\tilde{\phi} : \{0,1\}^{32} \to \mathbb{R}^{\tilde{D}}$ and a corresponding linear model $\tilde{\mathbf{W}} \in \mathbb{R}^{\tilde{D}}, \tilde{b} \in \mathbb{R}$ that predicts the responses i.e. for all CRPs $\mathbf{c} \in \{0,1\}^{32}$, we have $\frac{1+\text{sign}(\tilde{\mathbf{W}}^\top \tilde{\phi}(\mathbf{c})+\tilde{b})}{2} = r^0(\mathbf{c})$ where $r^0(\mathbf{c})$ is Response0 on the challenge $\mathbf{c}$. Similarly, show how a linear model can predict Response1 for a COCO-PUF. As before, your linear model may depend on the delay constants in PUF0 and PUF1 but your map must not use PUF-specific constants such as delays. (8 marks)

4. What dimensionality do your need the linear model to have to predict Response0 and Response1 for a COCO-PUF? This may be the same or different from the dimensionality you needed to predict the arrival times for the upper signal in a simple arbiter PUF. The dimensionality should be stated clearly and separately in your report, and not be implicit or hidden away in some calculations. (2 marks)

5. Write code to solve this problem by learning two linear models $\mathbf{W}_0, b_0, \mathbf{W}_1, b_1$ using the training data that predict Response0 and Response1 for the COCO-PUF. You are allowed to use any linear classifier formulation available in the sklearn library to learn the linear model (i.e. you need not write a solver yourself). For instance, you may use LinearSVC, LogisticRegression, RidgeClassifier etc. However, the use of non-linear models is not allowed. Submit code for your chosen method in `submit.py`. Note that your code will need to implement at least 2 methods namely

   (a) `my_map()` that should take test challenges and map each one of them to a D-dimensional feature vector.

   (b) `my_fit()` that should take train CRPs and learn the linear model by invoking an sklearn routine. It is likely that `my_fit()` will internally call `my_map()`. The method must return two linear models namely $\mathbf{W}_0, b_0, \mathbf{W}_1, b_1$ (see validation code on Google Colab for clarification)

   Note that your learnt model must be two vectors and two bias terms (if you do not wish to use a bias term, set it to 0). The use of non-linear models such as decision trees, random forests, nearest neighbors, neural networks, kernel SVMs etc is not allowed. To implement the map ϕ, you may find the Khatri-Rao product useful (`https://en.wikipedia.org/wiki/Khatri%E2%80%93Rao_product#Column-wise_Kronecker_product`). The KR-product is implemented in the Scipy package as `scipy.linalg.khatri_rao` which you can freely use without penalty (use of any other Scipy routine will incur penalties). However, use of the KR product is not compulsory and you may choose to not use it. Although the KR-product is define in a column-wise manner, you may find the row-wise version more useful by transposing the matrix before applying the KR-product. (35 marks)

6. Report outcomes of experiments with both the sklearn.svm.LinearSVC and sklearn.linear_model.LogisticRegression methods when used to learn the linear model. In particular, report how various hyperparameters affected training time and test accuracy using tables and/or charts. Report these experiments with both LinearSVC and LogisticRegression methods even if your own submission uses just one of these methods or some totally different linear model learning method (e.g. RidgeClassifier) In particular, you must report how at least 2 of the following affect training time and test accuracy:

   (a) changing the `loss` hyperparameter in LinearSVC (hinge vs squared hinge)

   (b) setting `C` in LinearSVC and LogisticRegression to high/low/medium values

   (c) changing `tol` in LinearSVC and LogisticRegression to high/low/medium values

   (d) changing the `penalty` (regularization) hyperparameter in LinearSVC and LogisticRegression (l2 vs l1)

   You may of course perform and report all the above experiments and/or additional experiments not mentioned above (e.g. changing the `solver`, `max_iter` etc) but reporting at least 2 of the above experiments is mandatory. You do not need to submit code for these experiments – just report your findings in the PDF file. Your submitted code should only include your final method (e.g. learning the linear model using LinearSVC) with hyperparameter settings that you found to work the best. (5 marks)

Parts 1,2,3,4,5 need to be answered in the PDF file containing your report. Part 5 needs to be answered in the Python file.

Evaluation Measures and Marking Scheme. We created two COCO-PUFs – a public one and a secret one. Both used 32-bit challenges but the arbiter PUFs in the secret COCO-PUF are different than those in the public COCO-PUF. Thus, a model that is able to predict the public COC-PUF responses is expected to do very poorly at predicting the responses for the secret COCO-PUF and vice versa. The train/test sets we have provided you with the assignment package were created using CRPs from the public COCO-PUF. We similarly created a secret train and secret test set using CRPs from the secret COCO-PUF.

We will use your `my_fit()` method to train on our secret train set and use the learnt models $\mathbf{W}_0, b_0, \mathbf{W}_1, b_1$ to make predictions on our secret test set. Given a test challenge $\tilde{\mathbf{x}}_t$, we will use your `my_map()` method to map it to $\tilde{D}$ dimensions and then apply the linear model as $\frac{1+\text{sign}(\mathbf{W}_0^\top \tilde{\phi}(\tilde{\mathbf{x}}_t)+b_0)}{2}$ to predict the COCO-PUF Response0 and similarly use $\frac{1+\text{sign}(\mathbf{W}_1^\top \tilde{\phi}(\tilde{\mathbf{x}}_t)+b_1)}{2}$ to predict Response1. Note that the secret train/test set may not have the same number of train/test points as the public train/test set that we have provided you. However, we assure you that the public and secret COCO-PUFs are similar in that hyperparameter choices that seem good to you during validation (e.g. C, tol, solver etc), should work decently on our secret dataset as well. We will repeat the evaluation process described below 5 times and use the average performance so as to avoid any unluckiness due to random choices inside your code in a particular trial. We will award marks based on four performance parameters

1. What dimensionality does your feature map use to predict the COCO-PUF responses i.e. how large is D? (5 marks)
2. How much time does your `my_fit` method take to finish training (10 marks)
3. How much time does your `my_map` method take to generate test features (10 marks)
4. What is the test misclassification rate offered by your model for Response0 and Response1? (5+5=10 marks)

For all performance parameters, lower is better (e.g. smaller D will get higher marks). Thus, the total marks for the code evaluation is 35 marks. For more details, please check the evaluation script on Google Colab (linked below). Once we receive your code, we will execute the evaluation script to award marks to your submission.

Validation on Google Colab. Before making a submission, you must validate your submission on Google Colab using the script linked below.

Link: `https://colab.research.google.com/drive/1kOI6Ab_owwZSWv3y2tE4UbhXSUBXYgzi?usp=sharing`

Validation ensures that your submitted file `submit.py` does work with the automatic judge and does not give errors. Please use the IPYNB file at the above link on Google Colab and the dummy secret train and dummy secret test set (details below) to validate your submission.

Please make sure you do this validation on Google Colab itself. **Do not download the IPYNB file and execute it on your machine – instead, execute it on Google Colab itself.** This is because most errors we encounter are due to non-standard library versions on students personal machines. Thus, running the IPYNB file on your personal machine defeats the whole purpose of validation. You must ensure that your submission runs on Google Colab to detect any library conflict. **Please note that there will be penalties for submissions which were not validated on Google Colab and which subsequently give errors with our automated judge.**

Dummy Submission File and Dummy Secret Files. In order to help you understand how we will evaluate your submission using the evaluation script, we have included a dummy secret train and secret test set in in the assignment package itself (see the directory called `dummy`). However, note that these are just copies of the train and test dataset we provided you. The reason for providing the dummy secret dataset is to allow you to check whether the evaluation script is working properly on Google Colab or not. Be warned that the secret dataset on which we actually evaluate your submission will be a different one that is generated using a completely different COCO-PUF. We have also included a dummy submission file `dummy_submit.py` to show you how your code must be written to return a model with `my_fit()` and features with `my_map()`. Note that the model used in `dummy_submit.py` is a very bad model which will give poor accuracies. However, this is okay since its purpose is only to show you the code format.

Using Internet Resources. You are allowed to refer to textbooks, internet sources, research papers to find out more about this problem and for specific derivations e.g. the arbiter-PUF problem. However, if you do use any such resource, cite it in your PDF file. There is no penalty for using external resources but claiming someone else's work (e.g. a book or a research paper) as one's own work without crediting the original author will attract penalties.

Restrictions on Code Usage. You are allowed to use the numpy module in its entirety and any sklearn submodule that learns linear models. To do so, you may include submodules of sklearn e.g. `import sklearn.svm` or `import sklearn.linearmodel` etc. However, **the use of any non-linear model is prohibited** e.g. decision trees, random forests, nearest neighbors, neural networks, kernel SVMs etc. The use of other machine learning libraries such as **libsvm, keras, tensorflow is also prohibited**. The only exception to this rule is the use of the `scipy.linalg.khatri_rao` routine from SciPy (all other SciPy routines are prohibited too). Use of prohibited modules and libraries for whatever reason will result in penalties. For this assignment, you should also not download any code available online or use code written by persons outside your assignment group (we will relax this restriction for future assignments). Direct copying of code from online sources or amongst assignment groups will be considered and act of plagiarism for this assignment and penalized according to pre-announced policies.

(60 marks)

2 How to Prepare the PDF File

Use the following style file to prepare your report.
`https://media.neurips.cc/Conferences/NeurIPS2023/Styles/neurips_2023.sty`

For an example file and instructions, please refer to the following files
`https://media.neurips.cc/Conferences/NeurIPS2023/Styles/neurips_2023.tex`
`https://media.neurips.cc/Conferences/NeurIPS2023/Styles/neurips_2023.pdf`

You must use the following command in the preamble

```
\usepackage[preprint]{neurips_2023}
```

instead of `\usepackage{neurips_2023}` as the example file currently uses. Use proper LaTeX commands to neatly typeset your responses to the various parts of the problem. Use neat math expressions to typeset your derivations. Remember that all parts of the question need to be answered in the PDF file. All plots must be generated electronically - hand-drawn plots are unacceptable. All plots must have axes titles and a legend indicating what are the plotted quantities. Insert the plot into the PDF file using LaTeX `\includegraphics` commands.

3 How to Prepare the Python File

The assignment package contains a skeleton file `submit.py` which you should fill in with the code of the method you think works best among the methods you tried. You must use this skeleton file to prepare your Python file submission (i.e. do not start writing code from scratch). This is because we will autograde your submitted code and so your code must have its input output behavior in a fixed format. Be careful not to change the way the skeleton file accepts input and returns output.

1. The skeleton code has comments placed to indicate non-editable regions. **Do not remove those comments**. We know they look ugly but we need them to remain in the code to keep demarcating non-editable regions.

2. We have provided you with data points in the file `public_trn.txt` in the assignment package that has 40000 data points, having a 32-bit challenge and a 2-bit response. You may use this as training data in any way to tune your hyperparameters (e.g. `C`, `tol` etc) by splitting into validation sets in any fashion (e.g. held out, k-fold). You are also free to use any fraction of the training data for validation, etc. Your job is to do really well in terms of coming up with an algorithm that can learn a model to predict the responses in the test set accurately and speedily, produce a model that is not too large, and also perform learning as fast as possible.

3. The code file you submit should be self contained and should not rely on any other files (e.g. other .py files or else pickled files etc) to work properly. Remember, your ZIP archive should contain only one Python (.py) file. This means that you should store any hyperparameters that you learn (e.g. step length etc) inside that one Python file itself using variables/functions.

4. We created two COCO-PUFs – a public one and a secret one. Using the public COCO-PUF, we created CRPs that were split into the train and test set we have provided you with the assignment package. We similarly created CRPs using the secret COCO-PUF and created a secret train and secret test set with it. We will use your code to train on our secret train set and use the learnt model to test on our secret test set. Both the public and secret COCO-PUFs have 32-bit challenges and use two PUFs each. However, the delays in the PUFs are not the same so a linear model that is able to predict the public COCO-PUF responses will do poorly at predicting the secret COCO-PUF delays and vice versa. Moreover, note that the secret train set is not guaranteed to contain 40000 points and the secret test set is not guaranteed to contain 10000 points (for example, our secret test set may have 19000 points or 21000 points etc). However, we assure you that the public and secret PUFs look similar otherwise so that hyperparameter choices that seem good to you during validation (e.g. step length, stopping criterion etc), should work decently on our secret dataset as well.

5. Certain portions of the skeleton code have been marked as non-editable. Please do not change these lines of code. Insert your own code within the designated areas only. If you tamper with non-editable code (for example, to make your code seem better or faster than it really is), the auto-grader may refuse to run your code and give you a zero instead (we will inspect each code file manually).
6. You are allowed to freely define new functions, new variables, new classes in inside your submission Python file while not changing the non-editable code.
7. The use of any non-linear model is prohibited e.g. decision trees, random forests, nearest neighbors, neural networks, kernel SVMs etc. The use of other machine learning libraries such as scipy, libsvm, keras, tensorflow is also prohibited.
8. You are allowed to use basic Python libraries such as numpy, time and random which have already been included for you. You are also allowed to use linear model learning methods from the sklearn library (e.g. sklearn.svm or sklearn.linearmodel) and the Khatri-Rao product method from the scipy library. Please note that you are not allowed to use any other routine from the scipy library.
9. Do take care to use broadcasted and array operations as much as possible and not rely on loops to do simple things like take dot products, calculate norms etc otherwise your solution will be slow and you may get less marks.
10. Please do not perform file operations in your code or access the disk – you should not be using libraries such as sys, pickle in your code.
11. The use of any non-linear model is prohibited e.g. decision trees, random forests, nearest neighbors, neural networks, kernel SVMs etc. The use of other machine learning libraries such as libsvm, keras, tensorflow is also prohibited.
12. Before submitting your code, make sure you validate on Google Colab to confirm that there are no errors etc.
13. You do not have to submit the evaluation script to us – we already have it with us. We have given you access to the Google Colab evaluation script just to show you how we would be evaluating your code and to also allow you to validate your code.