Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: May 29, 2023
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 is given below.

Assignment Package:
`https://www.cse.iitk.ac.in/users/purushot/courses/ml/2022-23-S/material/assn/assn1.zip`
Deadline for all submissions: 15 June, 2023, 9:59PM IST
Code Validation Script: `https://colab.research.google.com/drive/1hfl34PbGHLAL6boTpU_YxO6fRJj1g_it?usp=sharing`
Code Submission: `https://forms.gle/f7xgJ3fLbzBqbHLV9`
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*. Note that this means that auditors may not make submissions to this assignment.
2. 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.
3. 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 may incur penalties.
4. Link all group members in your group submission. If you miss out on a group member while submitting, that member may end up getting a zero since Gradescope will think that person never submitted anything.
5. You may overwrite your group's submission (submitting again on Gradescope simply overwrites the old submission) as many times as you want before the deadline.
6. 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 (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/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 may simply 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/f7xgJ3fLbzBqbHLV9`

9. Do not host your ZIP submission file on file-sharing services like GitHub or 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 and it is difficult for us to download automatically.

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 that 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 (Sparse PUFs)**.** Melbo wants to design a super-simple PUF that nevertheless offers good security. The idea makes use of a *conditional delay unit* (CDU) described below. A CDU takes in two inputs, a select bit and a signal. If the select bit is 0, then the CDU relays the input signal to its output without any delay i.e. 0 delay. If the select bit is 1, then the CDU relays the input signal to its output after a delay of p microseconds where p is kept secret and given a certain value of p, it is difficult to create another CDU with that exact value of delay.

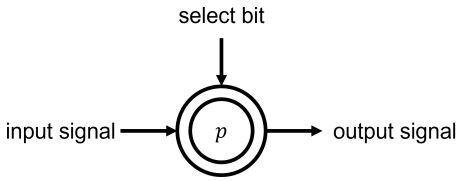

Figure 1: A single conditional delay unit.

CDU PUF: Melbo's idea is to create a good PUF by stringing together several CDUs in sequence. The select bits to these CDUs becomes the challenge and the responses is the total delay incurred, expressed in microseconds. See the figure below explaining delay calculation.

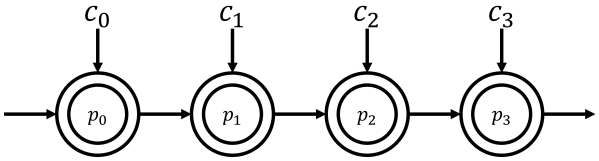

Figure 2: A PUF created by stringing together 4 CDUs. The response is the total delay incurred in the entire chain. For the challenge 0011, the delay would be $p_2 + p_3$ while for the challenge 1010, the delay would be $p_0 + p_2$ and for the challenge 1101, the delay would be $p_0 + p_1 + p_3$.

Sparse CDU PUF: Melbo was happy with this PUF construction till Melbo's sibling Melbi pointed out that a linear model can easily break this PUF as the responses are simply the sum of the delays of the CDUs chosen by the challenge. To make the PUF more challenging, Melbo devised a clever solution of introducing dummy/irrelevant CDUs into the pipeline. See the figure below that explains the trick. Melbo introduces D CDUs but only $S < D$ of them actually participate in the response generation process. The rest $D - S$ CDUs are disconnected from the chain. Thus, the value of the select bit sent to disconnected CDUs has no effect on the response. To make the problem challenging, Melbo refuses to reveal which S CDUs are active.

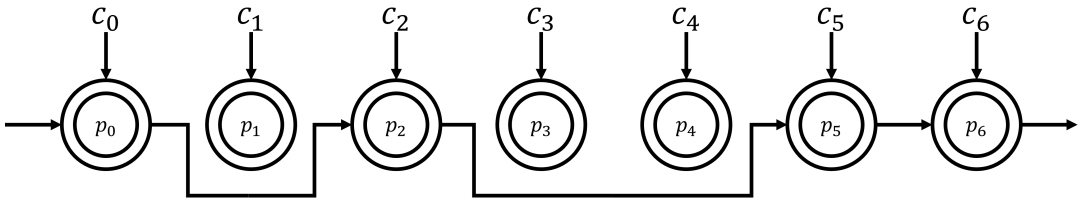

Figure 3: A Sparse CDU PUF with $D = 7$ CDUs out of which only $S = 4$ CDUs are actually participating in the response generation process. Thus, the values of c_1, c_3, c_4 have no effect on the response. For the challenge 1101011, the delay would be $p_0 + p_5 + p_6$ and the same delay would be obtained even for the challenge 1000111. For the challenge 0111010, the delay would be $p_2 + p_5$ and the same delay would be obtained even for the challenge 0110110.

Your Data. We have provided you with data from a Sparse CDU PUF with $D = 2048$ and $S = 512$ i.e. only 512 of the 2048 CDUs are actually active and the rest 1536 are disconnected and play no role in response generation. However, it is not known which 512 CDUs are active. A total of 1600 CRPs have been provided to you. The file `train_challenges.dat` contains 1600 lines each line giving a string of 2048 bits denoting the challenge. The file `train_responses.dat` containing 1600 lines each line giving the response obtained on those challenges in the form of delay in microseconds. Note that numbers provided in `train_challenges.dat` are bits i.e. either 0 or 1 whereas those in `train_responses.dat` are floating point numbers. Note that the number of training CRPs ($N = 1600$ in this case) was deliberately kept small by Melbo (note that $N \leq D$) to make the problem challenging.

If you wish, you may create (held out/k-fold) validation sets out of this data in any way you like. Your job is to learn a linear model that is D-dimensional but S-sparse that can accurately predict the responses on the test set. Thus, your model should be a single D-dimensional vector of which only S coordinates are non-zero and the rest $D - S$ coordinates should be zero. If you generate a model that has more than S non-zero coordinates, then we will forcibly make your model S-sparse using *hard thresholding*. See the Google Colab evaluation file linked to find out exactly how hard thresholding would be performed.

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

1. By giving a detailed mathematical derivation (as given in the lecture slides), show how a sparse CDU PUF can be broken by a single sparse linear model. More specifically, give derivations for a map $\phi : \{0,1\}^D \to \mathbb{R}^D$ mapping D-bit 0/1-valued challenge vectors to D-dimensional feature vectors and show that for any S-sparse CDU PUF, there exists an S-sparse linear model i.e. $\mathbf{w} \in \mathbb{R}^D$ such that $\|\mathbf{w}\|_0 \leq S$ i.e. $\mathbf{w}$ has at most S non-zero coordinates and that for all challenges $\mathbf{c} \in \{0,1\}^D$, the following expression $\mathbf{w}^\top \phi(\mathbf{c})$ gives the correct response. Note that no bias term (not even a hidden one) is allowed in the linear model. (10 marks)

2. Write code to solve this problem by learning a sparse linear model. Note that you may only use the `numpy` library to write your code (e.g. you may use the `lstsq` method to solve least squares – see below for more details). You are not allowed to use any machine learning library e.g. `sklearn, pandas, statsmodels, scipy, tensorflow, keras`, etc. Submit code for your chosen method in `submit.py`. Note that your code will need to implement a method called `my_fit()` that should take CRPs as training data and learn the S-sparse linear model. We will use your provided method to train on the training data we have provided you and then test it on a secret test dataset which consists of CRPs generated from the same Sparse CDU PUF. The same CDUs are active for the secret test set data as were for the train set data. (40 marks)

3. Below are described a few methods you can use to solve this problem. Give an account of which methods you tried (you can try methods other than those mentioned below as well) and why did you like one method over the other. Note that you only need to submit code for the method you found to work the best but in your report you must describe your experiences with at least one other method that you tried. (5 marks)

4. The method you submitted may have hyperparameters such as regularization constants, step lengths etc. Describe in detail how you chose the optimal value of these hyperparameters. For example, if you used grid search, you must show which all values you tried for each hyperparameter and what criterion did you use to choose the best one. (5 marks)

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

Evaluation Measures and Marking Scheme. We created a sparse CDU PUF with $D = 2048, S = 512$. It is not known which 512 CDUs are active in the PUF and which 1536 are disconnected. A few thousand CRPs were generated from this PUF out of which 1600 are provided to you as training data and the rest are kept secret and would be used for testing. We will use your code to train on the training data we have provided you to generate a linear model. Note that no bias terms (even hidden ones) are allowed in the model. If your model is not 512-sparse, it will be made 512-sparse using hard thresholding before further evaluation (see the Google Colab code to see how exactly would this be done). We will then test that model on the secret test CRPs. Since the test CRPs were generated from the same PUF, hyperparameter choices that seem good to you during validation (e.g. step length, regularization constant, stopping criterion etc), should work decently on test data as well. We will repeat the evaluation process described below 5 times and use the average performance parameters so as to avoid any unluckiness due to random choices inside your code in one particular trial. We will award marks based on four performance parameters

1. How fast is your `my_fit` method able to finish training (15 marks)
2. What is the Euclidean distance between the true model and your learnt model (5 marks)
3. What mean absolute error does your learnt model offer on test CRPs (10 marks)
4. How accurately was your model able to discover the 512 CDUs that were active (10 marks)

Thus, the total marks for the code evaluation is 40 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/1hfl34PbGHLAL6boTpU_YxO6fRJj1g_it?usp=sharing`

Validation ensures that your `submit.py` does work with the automatic judge and does not give errors due to array formats, wrong dimensionality etc. Please use the IPYNB file at the above link on Google Colab, the dummy test data and the dummy gold model 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 etc 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 test and dummy gold model file in the assignment package itself (see the directory called `dummy`). However, note that the dummy secret test data is just a subset of the training data and the

dummy gold model is a generic model with the first 512 coordinates non-zero and the rest zero. These are not the actual test data and gold model on which your submission will be evaluated. The reason for providing the dummy secret dataset and dummy gold model is to allow you to check whether the evaluation script is working properly on Google Colab or not. Be warned that the secret test data on which we actually evaluate your submission has not been released to you. We have also included a dummy submission file `dummy_submit.py` to show you how your code must be written to return a sparse linear model with `my_fit()`. Note that the model used in `dummy_submit.py` is a bad model which will give poor accuracies. However, this is okay since its purpose is to show you the code format.

2 A few Methods to Solve this Problem

The problem we have here is an instance of the *sparse recovery* problem which is a form of regression. Specifically, given data $\mathbf{x}_i\mathbb{R}^D, y_i \in \mathbb{R}$ for $i \in [n]$ we wish to learn an S-sparse linear model i.e. $\mathbf{w} \in \mathbb{R}^D$ such that $\|\mathbf{w}\|_0 \leq S$ i.e. $\mathbf{w}$ has at most S non-zero coordinates such that $\mathbf{w}^\top \mathbf{x}_i \approx y_i$ for all $i \in [n]$. This can be cast as an optimization problem

$$\begin{aligned} \arg\min_{\mathbf{w}\in\mathbb{R}^D} \frac{1}{2n}\sum_{i=1}^{n}(\mathbf{w}^\top\mathbf{x}_i - y_i)^2 \\ \text{s.t. } \|\mathbf{w}\|_0 \leq S \end{aligned}$$

Although this problem is NP-hard in general, there exist a few nice techniques to solve it described below. However, before that, let us define the hard-thresholding operation.

Definition 1.1 (Hard Thresholding)**.** Given a vector $\mathbf{z} \in \mathbb{R}^D$ and a sparsity level $S < D$, the hard thresholding operator returns the S-sparse vector that is closest to $\mathbf{z}$. Specifically,

$$\mathbf{v} = \mathrm{HT}(\mathbf{z}, S) = \arg\min_{\mathbf{u}\in\mathbb{R}^D:\|\mathbf{u}\|_0\leq S} \|\mathbf{u}-\mathbf{z}\|_2^2 .$$

Implementing this operator is super-simple – we simply sort the coordinates of $\mathbf{z}$ in decreasing order of their magnitude, retain the top S ones and set the other coordinates to 0 (see the Google Colab evaluation file for an implementation).

2.1 Naive Technique

In this technique, we first solve the least squares problem without worry about the sparsity constraint and then sparsify the solution using hard thresholding. Thus, we first find out

$$\mathbf{v} \stackrel{\text{def}}{=} \arg\min_{\mathbf{u}\in\mathbb{R}^D} \frac{1}{2n}\sum_{i=1}^{n}(\mathbf{u}^\top\mathbf{x}_i - y_i)^2$$

and then find a sparse solution by taking $\mathbf{w} = \mathrm{HT}(\mathbf{v}, S)$. You need not implement a least squares solver yourself. Instead, you may use the `lstsq` solver offered by the `numpy` model. Given features $X \in \mathbb{R}^{n\times D}$ and labels $\mathbf{y} \in \mathbb{R}^n$ for n data points for a regression problem, the following command

```
np.linalg.lstsq( X, y, rcond = None )[0]
```

solves the least squares problem

$$\arg\min_{\mathbf{w}\in\mathbb{R}^D} \frac{1}{2n}\|X\mathbf{w}-\mathbf{y}\|_2^2$$

Note that the `lstsq` solver returns a tuple of 4 outputs of which the first is the linear model which is why we used [0] to index out the first element of the tuple. Not using the `rcond = None` will generate a deprecation warning but the method should still work fine.

2.2 Relaxation Technique

The naive technique attempts to learn a dense model which performs poorly when sparsified. The LASSO (least absolute shrinkage and selection operator) technique encourages learning of a sparse model by using L_1 regularization by modifying the problem as follows.

$$\mathbf{v} \stackrel{\text{def}}{=} \arg\min_{\mathbf{u}\in\mathbb{R}^D} \ \lambda \cdot \|\mathbf{u}\|_1 + \frac{1}{2n}\sum_{i=1}^{n}(\mathbf{u}^\top \mathbf{x}_i - y_i)^2,$$

where $\lambda \geq 0$ is a *regularization constant*. Setting a very small value of λ e.g. $\lambda = 10^{-6}$ will be akin to just least squares and thus, not offer any benefits. Setting a very large value of λ e.g. $\lambda = 10^{+6}$ will likely yeild an all-zero model which is super sparse but not very useful either. However, for moderate values of λ, the LASSO solution, when sparsified i.e. $\mathbf{w} = \text{HT}(\mathbf{v}, S)$, yields good results. λ must be tuned as a hyperparameter but for the problem we have given you, the optimal value of lambda is expected to be somewhere in the range $[0.1, 5]$. Note that LASSO solutions are not automatically sparse and need to be sparsified as a last step.

There are several techniques one can use to solve the LASSO problem (there is no LASSO solver in `numpy`):

1. Sub-gradient descent: since the L_1 function is non-differentiable, the subgradient must be used. Stochastic and mini-batch stochastic variants can be used to improve speed as well. Step lengths must be carefully tuned as a hyperparameter.

2. Coordinate descent: In this technique, only one coordinate of $\mathbf{v}$ (say the j^{th} coordinate v_j) is updated in each iteration. x_{ij} denotes the j^{th} coordinate of the feature vector of the i^{th} data point.

$$v_j \stackrel{\text{def}}{=} \arg\min_{t\in\mathbb{R}^D} \ \lambda \cdot |t| + \frac{1}{2n}\sum_{i=1}^{n}(t \cdot x_{ij} + \sum_{k\neq j} v_k x_{ik} - y_i)^2,$$

   See the YouTube videos for discussion on how to use bookmarking techniques to speed-up coordinate descent techniques.

3. Proximal gradient descent: this is a more advanced (but also speedier) technique that you should attempt by consulting online resources or contacting the instructor. For the LASSO problem, the proximal descent method yields a technique called Iterative soft-thresholding algorithm (ISTA).

2.3 Projected Gradient Descent

This technique realizes that the hard thresholding operator is simply a projection onto the set of sparse vectors and applies gradient descent to the original problem, using hard thresholding to keep the iterates sparse. Let us denote

$$\ell(\mathbf{w}) \stackrel{\text{def}}{=} \frac{1}{2n}\sum_{i=1}^{n}(\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

1. Initialize $\mathbf{w}^0, t \leftarrow 0$

2. Let $\mathbf{z}^t \leftarrow \mathbf{w}^t - \eta \cdot \nabla \ell(\mathbf{w}^t)$

3. Let $\mathbf{w}^{t+1} \leftarrow \mathrm{HT}(\mathbf{z}^t, S)$

4. Update $t \leftarrow t + 1$ and repeat until convergence

Some variants:

1. Initialization: there are plenty of ways to initialize the model vector e.g. the all-zero vector i.e. $\mathbf{w}^0 = \mathbf{0}$, the solution of the least squares problem i.e. $\mathbf{w}^0 = \arg\min_{\mathbf{w} \in \mathbb{R}^D} \ell(\mathbf{w})$ or something else. Try out different initializations – they make a big difference. Note that you may use the `lstsq` method within `numpy` to solve least squares problems.

2. Correction: project gradient can be slow and take up a lot of iterations to give a good solution. There are well-known tricks to accelerate convergence, the simplest of which is *correction*. This technique uses PGD updates to estimate which coordinates of $\mathbf{w}$ are non-zero and then solves a least squares problem over those coordinates to make a large amount of progress. Thus, we compute $\mathrm{HT}(\mathbf{z}^t, S)$ as shown above, but instead of using it to update $\mathbf{w}^{t+1}$, we find out the non-zero indices in $\mathrm{HT}(\mathbf{z}^t, S)$ (lets call this index set $I \subset [D]$, note that $|I| \leq S$ by design). Next, we solve a least squares problem using the data $X_I, \mathbf{y}$ where $X_I \in \mathbb{R}^{n \times D}$ is a matrix prepared by zeroing out all columns of X that are not in the index set I. Let the solution of this S-dimensional least squares problem be $\mathbf{u} \in \mathbb{R}^D$. Then we update $\mathbf{w}^{t+1}$ by setting $\mathbf{w}_i^{t+1} = \mathbf{u}_i$ if $i \in I$ and $\mathbf{w}_j^{t+1} = 0$ if $j \notin I$. Note that you may use the `lstsq` method within `numpy` to solve least squares problems.

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. coordinate descent to solve the LASSO problem. However, if you do use any such resource, cite it in your PDF file. There is no penalty for using external resources so long as they are acknowledged 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 (e.g. you may use the `lstsq` method to solve least squares – see above for more details). However, **the use of any other library, including machine learning libraries, is prohibited** e.g. **sklearn, pandas, statsmodels, scipy, libsvm, keras, tensorflow**. 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. 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)

3 How to Prepare the PDF File

Use the following style file to prepare your report.
`https://neurips.cc/Conferences/2023/PaperInformation/StyleFiles`

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 - no hand-drawn plots would be accepted. All plots must have axes titles and a legend indicating what the plotted quantities are. Insert the plot into the PDF file using proper LaTeX `\includegraphics` commands.

4 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 training data in the files `train_challenges.dat` and `train_responses.dat` in the assignment package that has 1600 data points. You may use this as training data in any way to tune your hyperparameters (e.g. step size, regularization constant, stopping criterion 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 quickly learn a sparse model to predict the responses in the test set accurately.

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 inside that one Python file itself using variables/functions.

4. You are allowed to define new methods, new variables, new classes in inside your submission Python file while not changing the non-editable code.

5. The use of any library other than numpy e.g. machine learning libraries is prohibited e.g. sklearn, pandas, statsmodels, scipy, libsvm, keras, tensorflow.

6. 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.

7. Before submitting your code, make sure you validate on Google Colab to confirm that there are no errors etc.

8. 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.