**Indian Institute of Technology Kanpur**
**CS771 Introduction to Machine Learning**

*Instructor:* Purushottam Kar
*Date:* July 04, 2024
*Total:* 40 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/assn2.zip`
**Deadline for all submissions**: July 17, 2024, 9:59PM IST
**Code Validation Script**: `https://colab.research.google.com/drive/1vm69oldPVeg-VPu1EdR4AQETim7HcSAe?usp=sharing`
**Code Submission**: `https://forms.gle/F1m4s5s8T7uBsAqo6`
**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/F1m4s5s8T7uBsAqo6`

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** (Word Sequencing)**.** In the area of genetic research, a common task is to sequence a genome i.e., find out the exact sequence of base pairs (ATCG) present in the DNA of an organism. However, this is tricky to do directly. The DNA is seldom available in the form of a single long chain that can be read end-to-end. Instead, the DNA is often broken up into short strands each of which is sequenced to obtain the sequence of base pairs in those strands. The whole genome is then using these partial sequences by exploiting the fact that these strands are often overlapping.

Melbo is fascinated with genetics given its exciting applications such as synthetic biology, CRISPR etc that that have massive theraputic and industrial applications. Melbo wishes to gain experience in this field by solving an NLP version of the sequencing problem. For a string, a *character bigram* is simply a sequence of 2 characters that appear adjacent to each other in that string. Consider the English word `area` – it has the following 3 bigrams `( 'ar', 're', 'ea' )`. To take another example, the English word `beside` has the following 5 bigrams `( 'be', 'es', 'si', 'id', 'de' )`

Melbo wants to guess the word given only the bigrams that appear in that word. This problem may appear straightforward if the bigrams are presented in the order in which they appear in the word. However, similar to the DNA sequencing case, the bigrams are not available in such neat order. Instead the following is done before handing the bigrams to Melbo:

1. The set of bigrams are calculated for a word.
2. Duplicates are removed i.e. if a bigram appears twice or more in a word, its duplicate instances are removed.
3. The bigrams are sorted in lexicographic (alphabetically increasing) order.
4. Only the first 5 bigrams are retained and the rest are thrown away.

For example, for the English word `optional`, the entire list of bigrams is first created which happens to be `( 'op', 'pt', 'ti', 'io', 'on', 'na', 'al' )`. These are then sorted in lexicographic order to get `( 'al', 'io', 'na', 'on', 'op', 'pt', 'ti' )`. Then only the first 5 bigrams are retained to get `( 'al', 'io', 'na', 'on', 'op' )`.

We note that it may not be possible to uniquely identify a word given its bigrams processed as above. The process of sorting the bigrams, removing duplicates and retaining only 5 bigrams can cause clashes.

1. Sorting introduces clashes: the above process will give the same set of bigrams `( 'ar', 'ea', 're' )` for the words `area` and `rear` since the bigrams are sorted
2. Neglecting duplicates introduces clashes: the above process will give the same set of bigrams `( 'be', 'de', 'es', 'id', 'si' )` for the words `beside` and `besides` since we discard duplicate bigrams
3. Retaining only 5 bigrams introduces clashes: the above process will give the same set of bigrams `( 'al', 'io', 'na', 'on', 'op' )` for the words `optional` and `proportional` since we report only 5 bigrams

Still, it is assured that each bigram set corresponds to at most 5 words i.e. there is no 6-way clash – the worst that can happen is that the above process generates the same set of bigrams for 5 different words. An example of a 3-way clash is the following: the above process will give the same set of bigrams `( 'ci', 'en', 'ff', 'fi', 'ic' )` for the words `insufficient`, `sufficient`, and `sufficiently`.

Melbo's task is the following: given a list of bigrams as a Python tuple such that the tuple contains 5 or less bigrams and the bigrams are sorted in lexicographic order, guess one or more words that correspond to that list of bigrams. Melbo is allowed to make upto 5 guesses (since there could potentially be a 5-way tie). Melbo returns these guesses in the form of a list of words. If Melbo's list contains more than 5 guesses, only the first 5 guesses will be considered. If the correct word is present in Melbo's guess list, Melbo gets a score of 1 divided by the number of guesses Melbo made (thus, it is beneficial for Melbo to make as few guesses as possible). If the correct word is not in Melbo's guess list, Melbo gets a score of 0. This score is called the precision of the output. The average precision across all test points is the final precision of the model.

**Your Data.** We have provided you with a dictionary of $N = 5167$ words. Each word in the dictionary is written using only lower-case Latin characters i.e. `a - z`. At test time, Melbo will need to guess each word from the dictionary given its (deduplicated, sorted and truncated) bigram list.

**Your Task.** You need to develop an ML algorithm (we recommend a decision tree but other solutions are admissible as well) that can play this guessing game on words from the given dictionary. However, note that your algorithm will be tested on a secret dictionary that we have not revealed to you – more on this later. The following enumerates the 2 parts to the question. Part 1 needs to be answered in the PDF file containing your report. Part 2 needs to be answered in the Python file.

1. Give detailed calculations explaining the various design decisions you took to develop your ML algorithm. For instance, if you did use a decision tree algorithm, this includes the criterion to choose the splitting criterion at each internal node (e.g. if a certain bigram is present in the bigram list or not), criterion to decide when to stop expanding the decision tree and make the node a leaf, pruning strategies, hyperparameters etc. (10 marks)

2. Write code implementing your learning algorithm. You are allowed to use libraries such as numpy, scikit-learn, scipy, skopt. You are also allowed to use the code that we used to implement decision trees to play the Hangman game during the discussion hour in week 4 (code available on course website). However, you must ensure that you code does run on our judge system. To ensure this, you must validate your code on Google Colab. Submit code for your method in `submit.py`. Your code must implement a `my_fit()` method that takes a dictionary as a list of words and returns a trained ML model. Your code must also implement a `my_predict()` method that takes your learnt model and a tuple of bigrams sorted in ascending order, and returns a list of guesses. The `my_predict()` must return a list even if it is making only a single guess. If the list contains more than 5 guesses, only the first 5 will be considered. We will evaluate your method on a different dictionary than the one we have given you (see below for details). **Please go over the Google Colab validation code and the dummy submission file** `dummy_submit.py` to clarify doubts about data format, protocol etc. (30 marks)

Part 1 needs to be answered in the PDF file containing your report. Part 2 needs to be answered in the Python file.

**Warnings and Do Nots.** Your `my_fit()` and `my_predict()` methods **must not perform any file IO or attempt to establish internet connections**. Your `my_fit()` is supposed to package the model into a Python object and send it back to the evaluation code. The evaluation code will assess the size of the model and then send the model back to your `my_predict()` method. Your `my_predict()` method must use only the passed model to make its predictions. Attempting to cheat by storing part of the model on disk and trying to get higher marks for smaller model size will be detected by the judge and penalized. Note that you are allowed to package anything you want inside the model. For instance, if you feel you need the entire word list at test time, feel free to package the entire dictionary into the model. However, no file IO or attempt to establish internet connections is allowed whatsoever.

**Evaluation Measures and Marking Scheme.** We have two dictionaries with us, a public one and a secret one. The public one has been given to you in the assignment package but the secret one is safe with us. Both dictionaries are composed of words that are written using only lower-case Latin characters i.e. `a - z`. However, whereas the public dictionary is composed of English words, the secret dictionary may be in another language e.g. Spanish, French, Italian, Hindi, Bangla, etc but there will be no accents or special characters – only lower-case Latin characters will be used. Thus, do not assume that statistics such as character frequencies will be the same in the secret dictionary e.g. the character `e` may not be the most common in the secret dictionary. However, your `my_fit()` function may very well do operations to find out which is the most common character etc. The number of words in the secret dictionary will be between 5000 and 6000 and the length of each word will be between 4 and 15 characters. It is also certified that for every word, at most 4 other words may have the same (deduplicated, sorted and truncated) bigram list – i.e., there will be no 6-way tie, the worst is a 5-way tie which is why your `my_predict()` method is allowed upto 5 guesses as well.

1. How fast is your `my_fit()` method able to finish training (10 marks)
2. What is the on-disk size of your learnt model (after a pickle dump) (5 marks)
3. What is the total testing time for your model (15 marks)

**Mark adjustment** : Your code will first be awarded marks based on the above 3 criterion. Subsequently, your total marks out of 30 will be multiplied with the precision offered by your model. This is to create a disincentive for trivial solutions that return a large list of guesses or use a trivial model to achieve small training time or small model size. The method to calculate precision is outlined above but please refer to the evaluation script on Google Colab (linked below) for the exact method to be used to calculate precision. Thus, the total marks for the code evaluation is 30 marks. We will execute the evaluation script to award marks to your submission.

**Possible Solution Strategies.** You may of course follow standard information-theoretic entropy reduction algorithms to learn a decision tree such as we have discussed in class. You are free to use non-decision tree solutions as well. However, there is much more you can experiment with:

1. A slightly expensive but solid strategy is to implement *lookahead*. Notice that instead of choosing the split that gives the maximum entropy reduction right now (as we saw in class), a better strategy is too choose a split that will result in maximum entropy reduction two levels or three levels from now. The figure above shows an example where a greedy split may end up giving poorer result. However, implementing large lookahead can quickly

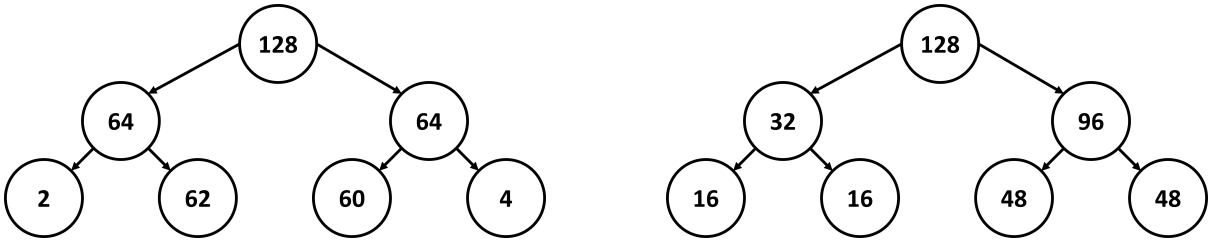


Figure 1: Greedy splitting can be suboptimal. In the example on the left, the split at the root was very nice (entropy 6.0) but it eventually led to a bad set of grandchildren (entropy 5.73). In the example on the left, the split at the root was not that nice (entropy $6.19 > 6.0$) but the set of grandchildren it produced had lower entropy (entropy 5.19).

blow up training time. If you at all wish to experiment with lookahead, do so once the number of words in the nodes has reduced to smaller numbers i.e. don't do lookahead at the root – do so a few levels down the root.

2. You may ditch entropy altogether and instead directly aim for minimizing the number of queries it takes to correctly guess the word. Suppose you have 100 words in a node at depth 2. Find out all subtrees that can be built from this node onward by trying out all possible queries at this node, then all possible queries at this node's children and so on for 3 levels (we have to stop at 3 levels since we cannot make more than 5 queries as the number of bigrams in the list is atmost 5 and the node at which we started was already at level 2). Of all these subtrees, choose the one that has the highest value of average precision across the 100 words or lowest average number of queries asked for those 100 words before success, or some combination thereof. Again, do not experiment with this technique close to the root. Try this only when the node sizes have become manageable and sufficiently small.

3. You may experiment with splits that are not just based on entropy reduction or expected time to success but that also offer a balanced split. Think of the balance of a split being a regularization term that prevents you from choosing an imbalanced split that simply reduces entropy a lot.

**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/1vm69oldPVeg-VPu1EdR4AQETim7HcSAe?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 Dictionary.** In order to help you understand how we will evaluate your submission using the evaluation script, we have included a dummy secret dictionary and a dummy submission file in in the assignment package itself (see the directory called `dummy`). However, note that the dummy secret dictionary is just a copy of the public dictionary. However, the dummy submission file may prove useful to you since it contains code to implement a fully functional decision tree for the problem. However, please note that the code may have errors and bugs. It also makes suboptimal decisions.

The reason for providing the dummy secret dictionary is to allow you to check whether the evaluation script is working properly on Google Colab or not. Be warned that the secret dictionary on which we actually evaluate your submission will be a different one. 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 `my_predict()`. Note that the model used in `dummy_submit.py` is a bad model which may give poor accuracies. However, this is okay since its purpose is 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.** Use of prohibited modules such as those that do file IO or make internet connections during training and/or testing for whatever reason will result in penalties. Direct copying of code from online sources or amongst assignment groups will be considered an act of plagiarism for this assignment and penalized according to pre-announced policies.

(40 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. 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.

3. 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.

4. You are not allowed to do file IO or establish internet connections using your training or testing code – you should not be using libraries such as sys, pickle in your code.

5. 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.

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

7. 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.