Indian Institute of Technology Kanpur
CS771 Introduction to Machine Learning

Instructor: Purushottam Kar
Date: September 20, 2019
Total: 100 marks

ASSIGNMENT 2

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:
`http://web.cse.iitk.ac.in/users/purushot/courses/ml/2019-20-a/material/assn2.zip`
Deadline for all submissions: 26 October, 9:59PM IST
There is no provision for "late submission" for this assignment

1.1 How to submit the PDF 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. Gradescope allows you to submit in groups - please use this feature to make a group submission.

3. To clarify the above, for example, there are, as of today i.e. Sep 20 2019, 49 assignment groups. We wish to have only 49 submissions on Gradescope, i.e. one submission per assignment group, not one submission per student.

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 PDF file using the style file we have provided (instructions on formatting given later).

1.2 How to submit the ZIP file

1. 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. Name your ZIP file `submit.zip`. Specify the file name properly in the Google form.

2. 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.

3. Make sure that the ZIP file does indeed unzip when used with that password (try `unzip -P your-password file.zip` on Linux platforms). If we are unable to unzip your file, you will lose marks.

4. 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`).

5. Fill in the following Google form to tell us the exact path to the file as well as the password `https://forms.gle/7RAbjxLPxVAoPs8Q9`

6. Do not host your ZIP submission file on file-sharing services like Dropbox or Google drive. Host it on IITK servers only. We will be using a script to autodownload your submissions and if not careful, Dropbox or Google Drive URLs may send us an HTML page (instead of your submission) when we try to autodownload your file. Thus, it is best to host your code submission file locally within IITK servers.

7. 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.

8. 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/mlassn2/submit.zip`

   (b) Example of an improper URL (file name missing):
   `https://web.cse.iitk.ac.in/users/purushot/mlassn2/`

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

9. 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, then we will not bother to make any efforts to manually locate your file or else contact you for clarifications. We will simply give your group a zero in these cases.

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

11. 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 simply assign your group zero marks.

12. We will entertain no submissions or mercy appeals 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 Python file must be available on the link specified on the Google form at or before the deadline.

Problem 2.1 (My First Recommendation System)**.** The problem of recommendation requires us to predict, for every user, which products/services would they like. We can obtain a simplified view of this problem by casting it as a *multi-label classification* problem. Simplified as it may be, this view is nevertheless very powerful and indeed widely used in production by actual companies in the e-commerce, social media and internet search sectors.

Users. Each user is represented as a d dimensional feature vector $\mathbf{x}^i \in \mathbb{R}^d$. This feature vector is typically very sparse i.e. only $\hat{d} \ll d$ of these coordinates are non-zero for an average user (some users may of course have more or less coordinates set to non-zero values in their representation). Nothing particularly specific is known about these d features except that they will always take non-negative values. These features were possibly arrived at by looking at normalized bag-of-"words" features where "words" may have been items these users viewed or bought in the past or their profile history, but we are not sure since the engineers who developed the feature extraction routines have left the company.

Items. There are a total of L items in the inventory out of which we wish to recommend items to the users. Each item is called a *label*. For each user i, we are told which all items do they like by giving us a *label-vector* $\mathbf{y}^i \in \{0,1\}^L$ for that user. For any item $j \in [L]$, $\mathbf{y}^i_j = 1$ indicates that user i definitely likes item j. Note that a user may like multiple items in which case all those coordinates would get turned on in the label vector for that user. This illustrates the difference between *multi-class* classification and *multi-label* classification. In multi-class settings, a data point can belong to exactly one of the possible classes but in multi-label settings, a data point may be associated with one or more of the labels. Going by what is mentioned about the items above, the reader may be tempted to think that just as $\mathbf{y}^i_j = 1$ indicates that user i definitely likes item j, then $\mathbf{y}^i_{j'} = 0$ should indicate that user i dislikes item j'. However, this is not true: $\mathbf{y}^i_{j'} = 0$ indicates that either user did not express an interest when shown the item j' or else the user was never shown the item j' at all! In an e-commerce website with 10 million items, it is unreasonable to expect a user to have interacted with, and to have told us, all the items in which they are interested – all recommendation systems have to be careful about this fact.

Your Task. Your task is to learn a recommendation system model which, when given a user feature vector, can tell us the top 5 items (labels) in which that user would be most interested. Given a test user, your method should return a vector `pred` of length 5 with each entry between 0 and $L-1$. The first entry of the vector i.e. `pred`[0] (note the zero-based indexing) will be interpreted by us as the item your method thinks is most likely to be liked by the user, the second entry of the vector i.e. `pred`[1] is the next most likely item to be liked by the user, and so on. Note that the first item is indexed as the 0^{th} label and the last item is indexed as the $(L-1)^{\text{th}}$ label (again in zero-based indexing). Thus, if you think the most suitable item for the user is the ninth item, you must set `pred`[0] = 8. Thus, we expect a *ranking* of the 5 most suitable items for that user. Note that you cannot cheat by repeating items in your recommendation vector e.g. by setting `pred`[0] = `pred`[1] = ... Our evaluation code will automatically remove duplicates from your recommendations.

Evaluation Measures. We will evaluate your method on our secret test data (which will have exactly the same value of d and L as the training data i.e. exactly the same set of features and items) using two kinds of performance measures described below

1. Precision at k (prec@k): For this, we will first choose some $k \in [5]$. Then we will ask, for every test user, what fraction of the top k recommendations given by your method

for that user were actually liked by that user. This will be a fractional number $\in [0, 1]$. Taking the average of this number across all test users will give us prec@k of your method.

2. Macro Precision at k (mprec@k): For this, first choose some $k \in [5]$. Then, we will go over each of the items $j \in [L]$ and for each item, look at all the test users that like that item. Then we will calculate the fraction of these users for whom your method did recommend item j in its top k recommendations. This will be a fractional number $\in [0, 1]$. Taking the average of this number across all items will give us mprec@k of your method.

The difference between prec@k and mprec@k largely arise due to the presence of *rare items* in the dataset i.e. items that very few users like. You will see in your data itself that an average item is liked by just $\hat{n} = 40$ users whereas there are a total of $n = 10000$ users! Whereas a method can get very high prec@k by just recommending popular items to everyone (akin to recommending an iPhone to everyone), such a method may do poorly on mprec@k which gives a high score only if that method pays good attention to all items, not just the popular ones.

Your Data. You have been provided in the assignment package, training data for $n = 10000$ users, each user i is represented as a $d = 16385$ dimensional feature vector $\mathbf{x}^i$. The feature vectors are sparse and the average number of non-zero features in a data point is only $\hat{d} \approx 519$. There are a total of $L = 3400$ items and each user is associated with a label vector $\mathbf{y}^i \in \{0, 1\}^L$. The label vectors are also quite sparse and an average user likes only $\hat{L} \approx 13.5$ items on an average. Routines have also been provided to you that read the data as well as show you how we will perform evaluation for your submitted code using prec@k and mprec@k.
Caution: all matrices in this problem (feature and label) are sparse and should be stored in compressed representation (`csr_matrix` from `scipy`). The data loading routine given to you as a part of the assignment package will return compressed matrices. Be careful about densifying these matrices (e.g. using `toarray()` from `scipy`) as you may run into memory issues.

Code and Resource Repository. The following repository offers the state of the art in large scale multi-label learning with papers that describe various methods that do well on such problems, as well as readily available code for some of these methods.
`manikvarma.org/downloads/XC/XMLRepository.html`

Suggested Methods. The following techniques are known to succeed in various forms on such large-scale multi-label problems

1. LwP: in literature this is often used as a *reranking* step (reference [31] in the repository) but can be used as an algorithm in its own right as well. This involves learning one or more prototypes per item that tries to capture prototypical users that like that item.

2. OvA: in literature this is also known as *binary relevance* (reference [32, 33, 34] in the repository) and this involves learning a binary classifier to predict for every item, whether a user will like that item or not. One may alternatively, use logistic regression to predict a probability score per item and rank items using that score. Note that this method sort of contradicts our earlier observation that $\mathbf{y}^i_j = 0$ does not mean that user i dislikes item j. Nevertheless, this method is quite successful.

3. DT: two main types of decision trees are popular in literature for multi-label problems

   (a) The first kind of approaches (examples are references [02, 31, 41] in the repository) use the decision tree to take a user to a leaf where the labels of training users who also reached that leaf, are used to perform prediction

(b) The second kind of approaches (examples are references [04, 36] in the repository) use the decision tree to merely eliminate which items would not be of interest to that user. An OvA-style classifier or ranker is then used to decide among the remaining items, which should be recommended

Apart from this there are a few other methods such as output codes (just as we saw for multiclassification), dimensionality reduction (aka embedding) methods, and deep learning methods, are also popular. Several references are available on the repository for all such techniques (although code may not be readily available for all of them).

1. Take one reference from the following list (references [2, 5, 31, 32, 33, 34, 36, 40, 41] in the repository), give a citation to that paper and briefly describe the method in your own words. You do not have to give all minute algorithmic details (since we can always just read the paper itself) but do explain the salient points which are crucial to the algorithm.

2. Discuss a few advantages and disadvantages of the method you just described above.

3. Download an implementation of the method you chose (or if you wish, implement yourself). Caution: the implementations may be in C/C++. Train the method you chose on the train data we have provided you and test it on the train data itself to give us the following statistics: prec@k and mprec@k for k = 1, 3, 5.

4. Suggest a method that you would use to solve the problem. The can either be

   (a) the method you chose to present in the previous parts with some modifications: in this case describe all modifications you made and relate them to the advantages and disadvantages you described in the previous part

   (b) the method you chose to present in the previous parts without any modifications: in this case you must justify your choice to not perform any modifications by enumerating what all modifications you tried but which failed to improve performance

   (c) some method other than the one you chose in the previous parts (with or without modifications) or else a totally new method: in this case give all details of the new method including all details for any modifications you propose to an existing method

5. Train your chosen method on the train data we have provided (using any validation technique you feel is good). Store the model you obtained in the form of any number of binary/text/pickled/compressed files as is convenient and write a prediction method in Python in the file `predict.py` that can take a new data point and use the model files you have stored to make predictions on that data point. Include all the model files, the `predict.py` file, as well as any library files that may be required to run the prediction code, in your ZIP submission. You are allowed to have subdirectories within the archive.

(10+10+20+10+50 marks)

Some Observations. You may use these to guide your exploration. DT methods usually offer fast prediction as well as good precision. However, their model size is usually large too. OvA methods usually offer very high precision values. However, they are usually slow at prediction. LwP methods usually offer high mprec@k since they do well on rare labels. However, they are usually slow at prediction as well. It is notable however, that these are broad observations and need not apply to every single method in these families. For instance, DT techniques of the second kind (that prepare a shortlist of labels) can use LwP methods in every leaf within the shortlisted labels to obtain reasonable prediction speed as well as good accuracies on rare labels.

Code Usage from the Internet. Irrespective of which option you choose in part 5, as well as part 4, you have full freedom to download any code available online to perform training. This is allowed (without any penalty) even if you choose option 2 in part 5. Thus, you are being given full freedom to use code written by others and there will be absolutely no penalty even if you (are able to) directly use someone's code without making any changes to it. However **you must cite all code that you take from others** by giving names of authors and name of the URLs from where you obtained said code. There is no penalty for using code for which you have cited the original author. However, there will be heavy penalty for using someone's work and not giving them credit for it.

Marking Scheme. Parts 1-4 need to be answered in the PDF file itself and part 5 needs to be answered in the ZIP file submission. The 50 marks for part 5 will be awarded as follows:
Total size of the submission, once unzipped (smaller is better): 10 marks
Total time taken to return top-5 recommendations for all test users (smaller is better): 10 marks
prec@1,3,5 (larger is better): 3 x 5 = 15 marks
mprec@1,3,5 (larger is better): 3 x 5 = 15 marks

2 How to Prepare the PDF File

Use the following style file to prepare your report.
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.sty`

For an example file and instructions, please refer to the following files
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.tex`
`https://media.neurips.cc/Conferences/NeurIPS2019/Styles/neurips_2019.pdf`

You must use the following command in the preamble

```
\usepackage[preprint]{neurips_2019}
```

instead of `\usepackage[final]{neurips_2019}` 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 any mathematical derivations. 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.

3 How to Prepare the ZIP File

Your submission ZIP archive must contain a file called `predict.py` which we will call to get recommendations for test data points. The assignment package contains a sample submission file `sample_submit.zip` which shows you how you must structure your submission as well as showing you the way in which your prediction code `predict.py` must accept input and return recommendations. Do not change this input output behavior otherwise our autograder will not be able to process your recommendations and award you zero marks.

1. Your ZIP archive itself must contain a python file named `predict.py` which should not be contained inside any other directory within the archive. Apart from this file your ZIP archive may contain any number of files or even directories and subdirectories.

2. Do not change the name `predict.py` to anything else. The file must contain a method called `getReco` which must take in the test features as a `csr_matrix` and a value $k > 0$ and return k recommendations for each test data point.

3. There are no "non-editable regions" in any of the files for this assignment. So long as you preserve the input output behavior, feel free to change the structure of the file as you wish. However, do not change the name of the file `predict.py` to anything else.

4. Code you download from the internet may be in C/C++. You are free to perform training in C/C++ etc. However, your prediction code must be a python file called `predict.py`. This file may internally call C/C++ code but it is your job to manage that using extension modules etc. Do not expect us to call anything other than the single file `predict.py`

5. The assignment package also contains a file called `eval.py` which is an example of the kind of file we will be using to evaluate the code that you submit. Before submitting your code, make sure to run `eval.py` and confirm that there are no errors etc.

6. Do not submit `eval.py` to us – we already have it with us. We have given you access to the file `eval.py` just to show you how we would be evaluating your code.

7. Make sure that running `predict.py` does not require us to install any Python libraries which are not available using `pip`. Libraries that are installable using `pip` are fine. However, if your prediction routine requires using external code not available using `pip`, you must supply all that external code to us within your ZIP archive – we should not have to install it. To be sure, we will not install or download any libraries from places like GitHub, personal homepages, or other repositories (even the repository mentioned in the question itself) even if you want us to. Your archive itself should contain all such files.

8. Other than the above restrictions, you are free to import libraries as you wish,e.g. sklearn, scipy, pandas. Files for libraries not available on `pip` must be provided by you.

9. Once we have installed `pip` libraries required by your code, we should just be able to call `predict.py` and get predictions i.e. your prediction code must be self contained. We should not have to create any new directories, move files around or rename them. All of that must be done by you while creating the archive (remember, you are allowed to have as many files and (sub)directories apart from the file `predict.py` as you want).

10. We do not need your training code for part 5. We do not need any code (training or prediction) that you used to solve parts 1-3. We simply need your prediction code for part 5. Do not increase your submission file size (remember there are marks for submission size too) by including unnecessary files.

11. The system on which we will run your prediction code will **not have a GPU**. Make sure your code can run simply on a CPU. Thus, use the CPU version of deep learning libraries if you are using them and not the GPU version of those libraries e.g. keras.

12. You may use a GPU (if you can find one) to train your method but your prediction method must still work on a CPU. To be sure, we will not offer your prediction code a GPU even if you ask us to. Do not use any libraries that require GPU access during prediction. There are no restrictions on using GPU for training but we will not provide your code GPU access during prediction.

13. We will test your submitted code on a secret dataset which would have the same features per user as well as same number of labels.