CSE 512: Project instructions

1 General instructions

- **General overview**: pick one of the following "questions" to answer, backing up your answer with analysis (theory or experiments).
- You work in teams of 2-4, unless you are registered for CSE 522, in which you work individually.
- You will submit a 2 page formal proposal + a poster presentation. Rubric below.
- Code that is more than 2 MB should be uploaded on a github with link provided. We will not run your code to grade it, but you are welcome to advertise it and win pride points.
- Please feel free to use the full range of Python-available code, e.g. `sklearn`, `pandas`, or any of the tools built in past homework assignments.
- **Literature review.** You must cite at least 3 papers involved in either inspiring you or competing against you in your experiments. Your report should include at least 1 paragraph per citation, discussing what they wrote and why it pertains to your project. Your literature review should include work that is *current* and *novel*–a tutorial on how logistic regression works does not count.
- Any additional references are welcome, and you must say at least something about all references in the body of your writeup, but how much you write is up to you.
- You **must** cite any references you borrow ideas from. Any suspicion of plagiarism will result in a 0 + reporting.
- It is ok to **borrow data and data preprocessing code** from your own ongoing experiments or past experiments, but you should be investigating a new question for this project.
- **Scope of project.** Pick a scope that fits your team's coding ability. For those of you who are strong in coding but weaker in theory, this is your chance to shine–go to town! (But don't forget to write a great report so that your work doesn't go unnoticed!) If you are relatively weaker, focus on picking an interesting problem and justifying your experimentation scheme. You can still get a significant amount of partial credit if what you claim you will do is logical and interesting, even if your code doesn't work perfectly in the end.
- **Teamwork.** You must include a brief description of each team member's contributions. In order for all members to get full credit, the work allocation should be as balanced as possible. (This does not need to be in the 5 page limit.)

2 Timeline

- Monday Sept 12: **(1 pt)** Project team members must be chosen. You must submit an abstract with a rough idea of what you want to do. This is not ironclad; you may change it around later, but you must submit something to start with. This will get a 0/1 grading, with +1 if a reasonable effort is demonstrated.
- Wednesday October 5: **(3 pts)** 2 page proposal due. This must include a thorough introduction and related works section, as well as a clearly described project outline. Feedback will be given. While I will give you a score, you may improve that score in the rebuttal period.

  Grading rubric:

  - 1.5 points: Clearly describing the motivation and presenting relevant literature.
  - 1.5 points: Setting up an experimental methodology that is logical and well-defended.

- Monday Oct 17 - Monday Oct 24: Period of rebuttal and resubmission of proposal. You get up to 2 resubmits. Feedback is guaranteed only if submitted before Friday Oct 21.
- Monday Nov 28 - In class poster presentation. You must submit a PDF of the final poster for grading, but you will also get a chance to present your project to teammates and external visitors.
- Wednesday Nov 30 - submit all files. **(4 pts)** You may include a 1 page description to accompany your poster, and any accompanying code (which most likely won't be read, but a github is always good to have.)

  Rubric

  - 2 points: Properly displaying results and connecting them to answer your motivating questions.
  - 1 point: Whether everything worked by the time of deadline.
  - 1 point: Subjective awesome-factor / scientific professionalism.

3 Questions

3.1 Compare and contrast ML tools.

In this class we focus on many different machine learning tools, of which work in different scenarios for different reasons. In practice, you will want to pick the tool that best fits your data structure and task needs. Explore this idea by picking two different "data modalities" and experimenting with at least 3 different classifiers or regressors, to see how they perform in these regimes.

Examples.

- giving movie recommendations, which may include social network features or IMDB movie descriptions as side information; vs sentiment analysis over twitter data, vs image classification tasks.
- physics or chemistry measurements, spectrograms, time signals, etc (usually objective but noisy) vs survey data for psychology (can also be subjective)
- short term financial data prediction (for microtrading) vs long term financial data prediction (e.g. real estate, annual GDP, etc)

Questions to consider. Are there certain tools that are better for certain data modalities than others? Are there certain tools that are better for solving the kind of question you are really after?

Extensions. Feel free to also count things like "boosted vs unboosted" or "ensembled vs standalone", as well as experimenting with different data preprocessing techniques (such as random hashing, learning embeddings, etc) to count as the different methods.

Metrics. Comment on some of the metrics we've discussed throughout the class: runtime, memory usage of different tools, hyperparameter tuning, overfitting tendency, etc. Make some recommendations to the practitioner about how to choose the right tool for each of the two tasks.

Do something surprising. Don't compare a CNN and a decision stump on image classification. We all know that isn't a fair comparison. Push a bit further, find some interesting and niche question to answer. I'm not looking for something really complex, but it should be a somewhat surprising discovery.

3.2 Make your own solver.

(Recommended for strong programmers.) A lot of tools these days have become standardized, which makes sense for a general audience, but sometimes if you have a very specific task or dataset, you can really get performance speedup by writing your own code.
If you pick this option, you can either choose to write code from scratch or rewrite certain parts of standardized code. For example, in PyTorch and Tensorflow you can rewrite the optimization module and keep everything else constant.

Possible things you can mess with.

- storing a compressed version of the model parameters (via quantization, matrix factorization, or forced random sparsificsaton, etc)
- replacing SGD/ADAM (and other standard library offers) with a completely different method, such as ADMM, greedy coordinate descent, or anything else
- holding certain parts of the model fixed and only training certain parameters at certain times, or modifying the step size so that certain parameters train faster than others

Models. Do not feel limited to deep learning models. You can also experiment with any of the other models we learn in the class. For example, maybe you try SVM with Nystrom column sampling. Or, you impose your own accelerated or memory reduced tree boosting scheme. The sky is the limit.

Goal. To clarify, the goal of this question is not so much to increase the performance of a task as it is to improve a way of training the model, maintaining the same (or almost the same) performance score. While of course we won't turn down a method that gives better scores, the goal here is *not* to, for example, compare two models of vastly different sizes and say that the first is better because it trains faster, or the second is better because the score is higher.

3.3 Who has the best pretrained model?

These days, many heavy duty tasks are made simpler with the availability of pretrained neural networks, which can then either be fine tuned, or just attached to a "final task layer" which is then trained for a specific task. This is often the case with HuggingFace models in NLP and CNNs for image processing.

- In this task, the goal is to try and figure out who has the best embedding for either the widest array of tasks, or for one or two very specific tasks that you yourself cook up.
- For example, maybe I want to use pretrained BERT to go through my own phone messages and do sentiment analysis on my conversations with all my friends, or go through movie scripts / books on Gutenberg to do plot embeddings, etc.
- While here the goal is not to go through every possible pretrained model you can download, you want to **identify at least 3 models** that behave sufficiently differently enough to tell an interesting story. Maybe one is very easy to train and super generalizable, but doesn't quite get state-of-the-art performance on a specific task compared to another. And so on.
- Make sure here you're doing the experiments yourself, rather than repeating experiences already out there, by picking truly unique tasks, or doing very creative analysis (not just reporting performance scores, but e.g. TSNE embeddings, changepoint / trajectory analysis, etc.) Your work does not have to be perfectly novel, but at the very least you should not simply repeat the ideas and methodologies of another paper. (If you are not sure about this please see me and we can discuss.)

3.4 Generalization.

Are there methods and models that have great optimization properties, but do not generalize well? Are there datasets of which overfitting is inevitable? Are some more tasks more problematic than others?

- In this project, explore two datasets and two ML tools (Again, these can be three different models, or one model with three different preprocessing techniques.) The goal is to see if, during training, you observe overfitting or not (and be sure to differentiate between the case where test loss suffers, vs test loss stays the same). Try to find examples that differ, and see if you can generate some hypotheses over what data structures / model structures are more amenable to overfitting than others.
- Then, evaluate three different overfitting prevention strategies on these four combinations. They can be: early stopping, ensemble methods, regularization, or something else you read in literature (dropout, noise injection, etc.) Discuss their effectiveness on each of the scenarios. Make some recommendations to a practitioner, based on your findings.

3.5 Verify a theory

I often find that the best way to learn something complicated and super mathematical is to just do some simulations and make some visualizations. If you choose this option, pick one of the big principles we discussed in class and try to code it up. For example:

- asymptotic bounds of the 1NN classifier vs the Bayes classifier
- Bayesian view of ridge regression, with reasonable priors
- gradient descent and stochastic gradient descent convergence rates, on problems with different conditioning
- generalization properties of overparametrized models (in particular, multilayer MLP vs kernel SVM vs margin maximizing methods)
- a deep dive into duality and SVMs

There should be two components to your experimentation:

- **Simulation data** Try to simulate the right data / problem regime where the thing you want to prove is forced to the limit. For example, find the case where 1NN reaches its asymptotic bound, or gradient descent exactly hits the upper bound on the convergence rate. Describe the data / problem model used to achieive this, and justify the presented behavior.
- **Real world data** Try to find some real world data that hits these limits, or argue that the limits in the theory are not realistic in practice, leading to some theory/practice performance gap. Do this convincingly, e.g. include some experiments on real data.

This problem is pretty hard to do right, so I strongly advise you setting up an appointment to chat with me if you pick this option. (However, it is not required.)

3.6 You and your adviser

Pick something you discussed with your research adviser. For this option, you must send me a (no more than 1 page) proposal by email at least 3 weeks before the due date, which I must evaluate and approve. [1]

3.7 Your team and your imagination.

Pick your own pet project. For this option, you must send me a (no more than 1 page) proposal by email at least 3 weeks before the due date, **and** schedule a personal meeting with me to discuss your idea, which I must evaluate and approve. [2]

4 Data resources

- For small datasets, the UCI repository `https://archive.ics.uci.edu/ml/index.php` has many great ones. If this is your first time exploring data science datasets, it's a great place to start.
- Many great data science challenge sets can be found on `https://www.kaggle.com/`.
- Natural language processing datasets and models can be found at `https://huggingface.co/`. If you go down this route, please take the time to understand the tokenization code, the models, and the training methodology.
- For images, standard datasets include ImageNet `https://www.image-net.org/` and CIFAR `https://www.cs.toronto.edu/~kriz/cifar.html`. MNIST is a standard digit classification dataset: `http://yann.lecun.com/exdb/mnist/` but SVHN is slightly more challenging `http://ufldl.stanford.edu/housenumbers/`.

[1] So, really, as early as possible.
[2] Again, as early as possible.

- There are many great data sources for recommendation systems, such as `https://movielens.org/`, as well as past RecSys challenges `http://www.recsyschallenge.com/2021/`. (Getting the second set of data may be challenging, but it should be out there somewhere.)
- A lot of interesting graphs can be found in `http://snap.stanford.edu/`. While we do not really talk about graph-based problems in this class, you're certainly welcome to use it in whatever creative way you think of.
- I do not recommend going out there to scrape datasets, as it takes quite a bit of time and will not win you extra project points. However, if you do feel ambitious, or if it is a part of a larger project you are doing for another course or for research, feel free! Please describe your data acquiring process clearly in your report.
- Please include links to any cool data repositories you find as well to the piazza to share with the class, and I will add them to next year's "datasets list".