Final Project Guideline

Instructor: Bo Liu

October 15, 2019

General Guideline

In the final project, you will work in groups (the maximum number of team members per group is 3) to apply the techniques that you've learned in this class to a new setting that you're interested in. Specifically, you will build a system to solve a well-defined task. Which task you choose is completely open-ended. Have fun and don't wait until the last minute!

The final project should consist of the following parts:

- Task definition: What does your system do (what is its input and output)? What real-world problem does this system try to solve? Make sure that the scope of the project is not too narrow or broad. This is probably the most important part of the project, but is also the part that you will not get practice doing from homeworks.

  The first task you might come up with is to apply binary classification on some standard dataset. This is probably not enough. If you are thinking in terms of binary classification, you are probably thinking too narrowly about the task. For example, when recommending news articles, you might not want to make predictions for individual articles, but might benefit from choosing a diverse set.

  An important part of defining the task is the **evaluation**. In other words, how will you measure success of your system? For this, you need to obtain a reasonably sized dataset of example input-output pairs, either from existing sources, or collecting one from scratch. A natural evaluation metric is accuracy, but it could be memory or running time. How big the dataset is depends on your task at hand.

- Infrastructure: In order to do something interesting, you have to set up the infrastructure. For machine learning tasks, this involves collecting data (either by scraping, using crowdsourcing, or hand labeling). For game-based tasks, this involves building the game engine/simulator. While infrastructure is necessary, try not to spend too much time on it. You can sometimes take existing datasets or modify existing simulators to save time, but if you want to solve a task you care about, this is not always an option. Note that if you download existing datasets which are already preprocessed (e.g., Kaggle), then you will be expected to do more with the project.

- Approach: Identify the challenges of building the system and the phenomena in the data that you're trying to capture. How should you model the task (e.g., using search, machine learning, planning, etc.)? There will be many ways to do this, but you should pick one or two and explain how the methods address the challenges as well as any pros and cons. What algorithms are appropriate for handling the models that you came up with, and what are the tradeoffs between accuracy and efficiency? Are there any implementation choices specific to your problem? Besides your primary approach(es), you should have two types of other ones, **baselines and oracles**. These are really important as they tell you how significant the problem you're solving is. Baselines are simple algorithms, which might include

predicting the majority label, using a small set of hand-crafted rules, training a simple classifier, etc. Baselines are meant to be extremely simple, but you might be surprised at how effective they are. Oracles are algorithms that "cheat" and look at the correct answer or involve humans. If your problem has two components (identifying entities and classifying them), an oracle uses the correct answer for one of the components, and sees how well the other can do. Baselines give lower bounds and oracles give upper bounds on your performance. If this gap is too small, then you probably don't have an interesting enough task.

- Literature review: Have there been other attempts to build such a system? Compare and contrast your approach with existing work, citing the relevant papers. The comparison should be more than just high-level descriptions. You should try to fit your work and other work into the same framework. Are the two approaches complementary, orthogonal, or contradictory?

- Error analysis: Design a few experiments to show the properties (both pros and cons) of your system. For example, if your system is supposed to deal with graphs with lots of cycles, then construct both examples with lots of cycles and ones without to test your hypothesis. Each experiment should ask a concise question, such as: Do we need to model the interactions between the ghosts in Pac-Man? or How well does the system scale up to large datasets? Analyze the data and show either graphs or tables to illustrate your point. What's the take-away message? Were there any surprises?

Milestones

Throughout the rest of the semester, there will be several milestones so that you can get adequate feedback on the project. You have **TWO presentations milestones** (proposal, progress report and final report) and **Two submission milestones** (proposal and final report).

- Proposal (**2 pages max report**): Define the input-output behavior of the system and the scope of the project. What is your evaluation metric for success? Collect some preliminary data, and give concrete examples of inputs and outputs. Implement a baseline and an oracle and discuss the gap. What are the challenges? Which topics (e.g., search, MDPs, etc.) might be able to address those challenges (at a high-level, since we haven't covered any techniques in detail at this point)? Search the Internet for similar projects and mention the related work. You should basically have all the infrastructure (e.g., building a simulator, cleaning data) completed to do something interesting by now. **The content of the proposal can be directly used in the final report** (with minor adaptations). In summary, it should include:
  1. problem formulation, clearly identifying the input/output and the challenges
  2. identifying data source
  3. literature review with a reference list
  4. a toy example with a baseline algorithm implementation (optional)

- Progress report (**only slides presentation**): Propose a model and an algorithm for tackling your task. You should describe the model and algorithm in detail and use a concrete example to demonstrate how the model and algorithm work. Don't describe methods in general; describe precisely how they apply to your problem (what are variables, factors, states, etc.)? You should also have finished implementing a preliminary version of your algorithm (maybe it's not fully optimized yet and it doesn't have all the features you want). Report your initial experimental results.

- Final report (**suggested length: around 10 pages report with slides presentation**): You should have completed everything (task definition, infrastructure, approach, literature review, error analysis).

Note:

(1) You can have an appendix beyond the maximum number of allowed pages with any figures/plots/examples if you need.

(2) **Everyone should be present (or have asked leaves justified beforehand) during any of the presentations. If not, your individual's corresponding milestone score is 0, no matter if you submit the progress report or not.**

Submission

Each of the team should submit the submission milestones **individually.** All milestones are due at 11pm (23:00, not 23:59). Late days (up to 2) can be used except for the final report.

- For each milestone, you should submit: proposal.pdf, or final.pdf containing a PDF of your writeup. group.txt containing the names and Banner IDs of the entire group, one per line. title.txt containing the title of your project on one line. keywords.txt containing one keyword (e.g., Bayesian networks) per line.

For final, you should also submit supplementary material, mainly the code. **There is no late extension for the final report!**

- Code: You can just package it up: code.zip containing the code (any language is fine; does not have to run on corn out-of-the-box) and a README file documenting what everything is and what commands you ran.
- Data: you don't need to submit the data, but should give the link to the data.

Grading

- Task definition: is the task precisely defined and does the formulation make sense?
- Approach: was a baseline, an oracle, and an advanced method described clearly, well justified, and tested?
- Data and experiments: have you explained the data clearly, performed systematic experiments, and reported concrete results?
- Analysis: did you interpret the results and try to explain why things worked (or didn't work) the way they did? Do you show concrete examples?
- Extra credit: does the project present interesting and novel ideas (i.e., would this be publishable at a good conference)?

Regardless of the group size, all groups must do the same basic amount of work (e.g., oracles, baselines, error analysis) as described in each milestone. Of course, the experiments may not always be successful, so we will cut the smaller groups more slack, while larger groups are expected to be more thorough in their experiments.

An example strategy

This is a suggestion of how to approach the final project with an example.

- Pick a topic that you're passionate about (e.g., food, language, energy, politics, sports, card games, robotics). As a running example, say we're interested in how people read the news to get their information.
- Brainstorm to find some tasks on that topic: ask "wouldn't it be nice to have a system that does X?" or "wouldn't it be nice to understand X?" A good task should not be too easy (sorting a list of numbers) and not too hard (building a system that can automatically solve the homeworks). Please come to office hours for feedback on finding the right balance. Let's focus on recommending news to people.
- Define the task you're trying to solve clearly and convince yourself (and a few friends) that it's important/interesting. Also state your evaluation metric – how will you know if you have succeeded or not? Concentrate on a small set of popular news sites: nytimes.com, slashdot.org, sfgate.com, onion.com, etc. For each user and each day, assume we have acquired a set of articles that the user is interested in reading (training data). Our task is to predict for a new day, given the full set of articles, the best subset to show the user; evaluation metric would be prediction accuracy.
- Gather and clean the necessary data (this might involve scraping websites, filtering outliers, etc.). This step can often take an annoyingly large amount of time if you're not careful, so do not try to get bogged down here. Simplify the task or focus on a subset of the data if necessary. You might find yourself adjusting the task you're trying to solve based on new empirical insights you get by looking at the data. Notice that even if you're not doing machine learning, it's necessary to have data for evaluation purposes.
- Implement a baseline algorithm. For a classification task, this would be always predicting the most common label. If your baseline is too high, then your task is probably too easy. One baseline is to always produce the first document from each news site. Also implement an oracle, for example, recommending the document based on the number of comments. This is an oracle because you wouldn't have the number of comments at the time you actually wanted to recommend the article!
- Formulate a model and implement the algorithm for that model. You should try several variants and compare them. Remember to try as much as possible to separate model (what you want to compute) from algorithms (how you do it). You might train a classifier to predict, for each news article, whether to include it or not.
- Perhaps the most important part of the project is the final step, which is to analyze the results. It's more important that you do a thorough analysis and interpret your results rather than implement a huge number of complicated heuristics in trying to eke out the maximum performance. The analysis should begin with basic facts, e.g., how much time/memory did the algorithm take, how does the accuracy vary with the amount of training data? What are the instances that your system does the worst on? Give concrete examples and try to understand why. Is there a bottleneck? Is it due to lack of training data?

Libraries

You are free to use existing tools for parts of your project as long as you're clear what you used. When you use existing tools, the expectation is that you will do more on other dimensions.

- scikit-learn: machine learning library implemented in Python
- Tensorflow/PyTorch + Keras: deep learning toolbox.
- Caffe: deep learning toolbox, highly optimized for CNN.
- OpenAI Gym: deep reinforcement learning toolbox, also contains usual deep learning tools.

- Natural language Toolkit (NLTK): a set of tools for basic NLP in Python
- OpenCV: Python libraries for simple computer vision

Some project ideas

Machine Learning

- Hierarchical Machine Learning Algorithms: We have introduced many machine learning algorithms. Can you combine them to make a hierarchical algorithm? For example, Naive Bayes on top of a decision tree. And there are also many other options.
- Variant of Standard Machine Learning Algorithm: Can you tweak the existing machine learning algorithms? For example, we introduced several ways to build different trees in random forest, can you build a different random forest where each tree is completely random built, or some other way around?

Planning

- Atari: Deep Reinforcement Learning: Use the platform offered by(http://karpathy.github.io/2016/05/31/rl/), see if you can beat their performance. Let us know what tricks and hacks you used.

Applications

- Predict the price of airline ticket prices given day, time, location, etc.
- Predict the amount of electricity consumed over the course of a day.
- Predict whether the phone should be switched off / silenced based on sensor readings from your smartphone.
- Auto-complete code when you're programming.
- Answer natural language questions for a restricted domain (e.g., movies, sports).
- Search for a mathematical theorem based on an expression which normalizes over variable names.
- Find the optimal way to get from one place on campus to another place, taking into account uncertain travel times due to traffic.
- Solve Sudoku puzzles or crossword puzzles.
- Build an engine to play Go, chess, Poker, etc.
- Break substitution codes based on knowledge of English.
- Automatically generate the harmonization of a melody. Generate poetry on a given topic.