CS221 Final Project Guidelines

[slides]

In the final project, you will work in groups of **one, two, three, or four** to apply the techniques that you've learned in this class to a new setting that you're interested in. Note that the larger the group, the higher the expectations for the project.

You will build a system to solve a well-defined task. Which task you choose is completely open-ended, but the methods you use should draw on the ones from the course.

After you submit your proposal, you will be assigned one of the CAs as your official mentor. He or she will grade all your work and really get to know your project. You are encouraged to come to office hours often to discuss your project; you can also go to Percy or other CA's office hours to get a second opinion. Note that it will take several iterations to find the right project, so be patient; this exploration is an essential part of research, so learn from it. 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. For example, building a system to answer any natural language question is too broad, whereas answering short factoid questions about movies is more reasonable. 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 of articles.

  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, logic, 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?

  Before developing your primary approach, you should implement **baselines** and **oracles**. These are really important as they give you intuition for how easy or hard the problem you're solving is. Intuitively, baselines give lower bounds on the performance you will obtain and oracles give upper bounds. If this gap is too small, then you probably don't have a good task. Importantly, baselines and oracles should be relatively easy to implement and can be done before you invest a lot of time in implementing a fancier approach. This can prune out problems early and save you a lot of time!

  Baselines are simple algorithms, which might include using a small set of hand-crafted rules, training a simple classifier, etc. (Note that baselines are extremely simple, but you might be surprised at how effective they are.) While a predictor that guesses randomly provides a lower bound (and can be reported in the paper), it is too simple and doesn't give you much information. Predicting the majority label is a slightly less trivial baseline, and whether it's acceptable depends on how insightful it is. For classification, if the different labels have very different proportions, then it could be useful; otherwise it won't be. You are encouraged to have multiple baselines.

  Oracles are algorithms that "cheat" and look at the correct answer or involve humans. For human-like classification problems (e.g., sentiment classification), you can have each member of your project team try to annotate ~50 examples and measuring the agreement rate. Note that some tasks are subjective, so even though humans are providing ground truth labels, human accuracy will not be 100%. When the classification problem is not human-like, you can try to use the training error of an expressive classifier (e.g., nearest neighbors) as a proxy for oracle error. The idea is that if you can't even fit the training data using a very expressive classifier, there is probably a lot of noise in your dataset, and you have a slim chance of building any classifier that does well on test. While returning 100% is an upper bound, it is not a valid oracle since it is vacuously an upper bound. Sometimes, oracles might be difficult to come by. If you think that no good oracles exist, explain why.

Overall, the main point of baselines and oracles is to get you to look at the problem carefully and think about what's possible. The accuracies of state-of-the-art systems on the dataset could be either a baseline or an oracle. Sometimes, there are data points that are neither baselines nor oracles: for example, in a two component system, you use an oracle for one and a baseline for another.

- **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 quarter, there will be several milestones so that you can get adequate feedback on the project.

- **Proposal** (10% Points) (2 pages max): 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.

- **Progress report** (20% Points) (4 pages): 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.

- **Poster session** (20% Points): By the poster session, you should have finished implementation, run a good chunk of your experiments, and done some basic error analysis. In the poster, you should describe the motivation, problem definition, challenges, approaches, results, and analysis. The goal of the poster is to convey the important high-level ideas and give intuition rather than be a super-detailed specification of everything you did (but you should still be precise). You will be evaluated on both the contents of the poster as well as your presentation. You will be assigned to either Session A (1-2:20pm), Session B (2:20-3:40pm) or Session C (3:40-5pm). You should stand by your poster during your assigned slot. During the poster session, the course staff will come around. You should be able to give a 30-second elevator pitch providing the highlights of your work, but also be prepared to take questions about any of the specific details. Additional tips:
  - Use lots of diagrams and concrete examples. Use bullets for the key points, and make sure you use a large font that can be read from a distance. Don't write long sentences and lots of complex equations.
  - Organize your poster into sections, so it's clear what the components of the project are. This also makes it easy to delve into a particular component.
  - Practice and polish your 30-second pitch. Someone should be able to understand what you're doing and importantly, why, from listening to it.
  - If you can make a live demo, you should do it!
  - The posterboards are 20" x 30" (not that big!). You can print your poster at Meyer library or Kinkos. Or, for a super convenient/cheap option, you can use blockposters, which splits up your poster into small pages which can be printed on a normal printer.
  - At the beginning of the poster session, you should check in with your mentor, who will provide easels and stands in exchange for a group member's student ID card. You should setup your poster around the mentor's designated area.
  - All group members are expected to be at the poster session. If you cannot make it, coordinate with your mentor.
  - In all cases, submit your poster as a PDF using the submit script by 11pm that evening.

- **Poster session peer review**: During the time that you're not assigned, you should wander around and look at other people's posters — after all, the whole point of having a poster session so that everyone can share what they've accomplished over the quarter. You might even get ideas for your own project. Based on your wanderings, you should choose 3 posters that you liked the most and write a sentence or two describing each poster and why you liked it. These reviews will not influence anyone's grade, but is just a fun way to encourage you to engage in the poster session.

- **Final report** (50% Points) (5-10 pages): You should have completed everything (task definition, infrastructure, approach, literature review, error analysis).

Note: you can have an appendix beyond the maximum number of allowed pages with any figures/plots/examples if you need.

Submission

Submit the milestones using the submit script as usual, but make sure the **same one member** of your group submits on behalf of the entire group.

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`, `progress.pdf`, or `final.pdf` containing a PDF of your writeup.
- `group.txt` containing the **SUNetIDs** (e.g., `psl`) of the entire group, one per line. Do not include your names or university ID numbers (e.g., `06091262`). Do not include the SUNet IDs of team members who are not enrolled in CS 221.
- `title.txt` containing the title of your project on one line.
- `keywords.txt` containing one keyword (e.g., Bayesian networks) per line.
- `cas.txt` containing the SUNet IDs of the CAs you wish to have as a mentor, one per line, ranked by preference.

For **p-final**, you should also submit supplementary material. There are two ways to do this. First, you can just package it up:

- `code.zip` containing the code (any language is fine; does not have to run on rice out-of-the-box) and a README file documenting what everything is and what commands you ran.
- `data.zip` containing the data.

The file size limit is **20MB** per file. If the data does not fit in the file size limit, submit a small but meaningful subset of the data.

We encourage you to submit your project as a CodaLab worksheet, which certifies that the experiments in your project are reproducible. With CodaLab, you upload your code and data and interleave the description of your experiments with their actual execution. **Extra credit** will be awarded to those that produce a meaningful CodaLab worksheet. Just include the link in the final report.

Grading rubric

- **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 CS221 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. *Write some scripts that download the RSS feeds from the news sites, run some basic NLP processing (e.g., tokenization), say, using NLTK.*

- 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. You might try to include these predictions as factors in a weighted CSP and try to find a set of articles that balance diversity and relevance.*

- 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?

Datasets

You are free to use existing datasets, but these might be not necessarily the best match for your problem, in which case you are probably better off making your own dataset.

- Kaggle is a website that runs machine learning competitions for predicting for monetary reward.
- Past CS229 projects: examples of machine learning projects that you can look at for inspiration. Of course your project doesn't have to use machine learning – it can draw from other areas of AI.
- MLcomp: a website with various learning algorithms and datasets, which you can run directly on the site or download. If you have a new dataset, you can upload it and run it on many different algorithms for a quick comparison.
- SAT competition: satisfiability problems are a special important class of CSPs.
- Natural language processing datasets: links to many NLP datasets for different languages.
- Datasets from CS224W (Social and Information Network Analysis).
- Poker hand dataset.

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

Some project ideas

- 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 Stanford 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, 2048, Poker, etc.
- Break substitution codes based on knowledge of English.
- Automatically generate the harmonization of a melody.
- Generate poetry on a given topic.

You can also get inspiration from last spring's CS221 projects (student access only).

Frequently asked questions

Can I use the same project for CS221 and another class (CS229, etc.)? The short answer is that you cannot turn in the identical project for both classes, but you can share common infrastructure across the two classes. First, you should make sure that you follow the guidelines for the CS221 project, which are likely different from those of other classes. Second, if any part of the project is done for a purpose outside CS221 (for the final project in CS229 or other classes, or even for your own research), then in the progress and final reports, you must clearly indicate which part of the project was done for CS221 and which part was not. For example, if you're taking CS229, then you cannot turn in the same pure machine learning project for CS221. But you can work on the same broad problem (e.g., news recommendation) for both classes and share the same dataset / generic wrapper code. You should then explore the machine learning aspect of the problem for CS229 (e.g., classifying news relevance) and another topic for CS221 (e.g., optimizing diversity across news articles using search or CSPs).

Are there restrictions on who I can partner up with for the final project? The only hard requirement is that each member of your group must be enrolled in CS221. Thus, if you choose to use the same project for CS221 and another class, all of your partners must be in CS221. If you feel like you have a compelling case for an exception, please submit a request detailing the parts of the project used for each class and the reasons for deviating from the project policies.

How do I choose a good baseline and oracle? Both baselines and oracles should be simple and not take much time. The point is not to do something fancy, but to work with the data / problem that you have in a substantive way and learn something from it. Here are some examples of baselines:

- Manually hand-code a few simple rules
- Train a classifier with some simple features

Guessing completely at random is technically a baseline, but is a really bad one because it doesn't really tell you much about how easy the problem is. Here are some examples of oracles:

- Manually have each member of your team hand label a handful of examples. The oracle is the agreement rate between people.
- Measure the training accuracy using reasonable features. The idea is that if you can't even fit your training data very well, then your test error is probably going to be even worse.
- Predict the label given information that you wouldn't normally have a test time (for example predicting the future given other information from the future).

Always guessing the correct label is technically an oracle, but it's a really bad one, because you'd always get 100% and you don't learn anything from it.