# Probabilistic Programming Languages
# Syllabus

Fall 2016

## Introduction and welcome

Any computer program can compute *forwards*, from causes to effects. But in many modern applications, users want computers to reason backwards, from effects to causes:

- I see constellation of medical symptoms. What, if any, is the disease that causes those symptoms?
- I see an image of a scanned bubble form. What is the UTLN of the student who coded it?
- I see an image from a mammogram. Was any part of the image caused by a cancer?
- I see words in an email. Were they written by a spammer?

From the last two examples especially, I hope you agree that a good answer should include a *probability*.

At present, programs that do this kind of probabilistic reasoning are written primarily by highly trained specialists using old, general-purpose programming languages like Matlab or C++. But soon, programmers will use new languages to write programs that reason probabilistically. The design and implementation of *probabilistic programming languages* is a problem at the leading edge of research.

This research is ramified and chaotic; new ideas are still happening. I'm pleased to be able to bring some of the intellectual ferment into the classroom; I hope it will engage your interest and your intellectual curiosity. Probabilistic programming is an exciting new model of programming and one that I hope will become influential.

Beyond the immediate pleasures of this particular research, almost all of you are soon to embark on what I hope will be long careers in computing. Computing technology changes so rapidly that anyone who has a long career can expect to spend some of it putting research results into practice. During this course I will model for you how to approach potentially practical research in a new, unproven area. When in the future, you have to consider applying research on your own, the experience will give you a leg up.

## What will we learn?

The course is designed around three learning goals:

- I want you to understand, in a way that you can relate to your experience in computing, the problem that the designers of probabilistic languages are trying to solve.
- I want you to see that parts of the problem yield to a combination of 19th-century mathematics and 21st-century program analysis.
- I want you to see how, despite the satisfying mathematical underpinnings, the existing solutions to the problem are diverse and are, at least to a degree, unsatisfying.

Together, these goals address a classic question that should be asked about any program of research: why is this stuff worth working on? You may also learn, to the degree possible given the current state of the art, how to use probabilistic programming languages to solve problems.

In the rest of this section of the syllabus, I describe in more detail the direction I plan for your learning to take. This direction is informed by my experience teaching these ideas once before, in 2014.

### Learning priorities

My highest priority is for you to understand the two key ideas that underlie the expression of problems in probabilistic languages and the algorithms that can solve those problems:

- You will understand what *inference* means. You will be able to relate language designs to inference algorithms. You will understand that there are no universal inference algorithms, and you will be able to explain when and how inference fails. You will understand some of the limitations that inference imposes on the design and use of a probabilistic language.
- You will understand what *conditioning* means. You will understand how conditioning is expressed linguistically and why it is important for applications. You will be comfortable using conditional probability to describe real-world scenarios. You will understand that conditional probability can be

interpreted computationally in two directions: from cause to effect and from effect to cause.

My other priorities involve your ability to make intellectual connections between snapshots of different ideas in probabilistic programming languages.

- You will be able to extract interesting information from a research paper.
- You will be able to compare two different approaches to the design of probabilistic programming languages, which I'll call the *machine-learning approach* and the *programming-language approach*:
  - You'll understand that the machine-learning people care deeply about algorithms and their performance.
  - You'll understand that the programming-language people care deeply about foundations and about modularity.
  - You'll be able to identify hallmarks of the two different approaches, including aspects of a language design that people from each approach would consider markers of quality.
- You'll be able to connect the semantics of probabilistic languages with the mathematics of probability.
- You'll be able to explain, probably by appealing to examples, why *program analysis* could help implement probabilistic languages.

### Subject matter

Probabilistic programming languages are a young, interdisciplinary subject, and at present the intellectual story is somewhat incoherent.

- The programming-language approach builds on proven language technology such as formal semantics and first-class functions. This approach shows practitioners' concerns with classical problems of language design: abstraction, modularity, clarity of meaning, and elegance of expression.
- The machine-learning approach builds on proven machine-learning technology, primarily technology for representing joint probability distributions. This approach highlights sophisticated, reusable, valuable technology—technology which researchers valued so highly that they wished to embody it in a language design.

To force some coherence on the subject matter, the class will emphasize the programming-language approach, and we will focus on program analysis. The machine-learning approach will be treated as a kind of "wild West" which we will study from a distance, without trying to tame it.

## What will happen in the classroom?

Most class periods will be spent working on small problems or answering discussion questions. The problems and questions will usually be grounded on outside reading. Work will be undertaken at the board in groups of 4 to 5 students.

As permitted by the number of students enrolled, some class periods will be spent reviewing your solutions to small problems posed outside of class. The purpose of these problems is not to assess your performance but to enable deeper and more informed class discussion than would otherwise be possible. For this reason, I do not plan to give you grades for these problems. I also encourage you to tackle the problems in pairs or in *small* groups.

A few class periods will be spent reviewing programming-language fundamentals from COMP 105 as well as fundamentals of probability.

## What kind of class is this? How is it different?

Most classes in computing are organized around lectures and homeworks (problem sets or programming assignments), perhaps with some laboratory experiences. This class is a *seminar*,[1] and as such it is organized around small-group collaboration. So that you understand the distinction, I compare the two kinds of classes.

In my mind, a good lecture course has these properties:

- The instructor provides challenging problems that students work on outside of class.
- The problems are the primary focus of the class.
- The experience is carefully scripted in advance.
- The pace of the course is set by the instructor, and once the train leaves the station, it does not stop for anybody.
- If as a student, you are not fully engaged with the class, you hurt nobody but yourself.

A seminar should offer a dramatically different experience. Like a lecture course, it should offer challenging problems, but everything else should be different:

- A seminar should not have a narrow focus. The experience should be broader and should revolve around three kinds of activities:
  1) Reading, understanding, and evaluating the work of scholars

[1] According to the Oxford English Dictionary, a select group of advanced students associated for special study and original research under the guidance of a professor.

2) Tackling new problems and new techniques

3) Learning how members of a particular scholarly community think, talk, and write

- A seminar should not be scripted in advance. Instead, it should have a general direction—a goal or two—and a plan for exploring the scholarly territory on that way to the goals.
- A seminar's pace should allow for reflective pauses and side excursions. It should be flexible enough to allow the class to pursue interesting ideas as they arise.
- In a lecture class, you do the bulk of the work outside class (in problem sets or programming assignments ), and the role of the class time is mostly to prepare you to do this work.

  By contrast, in a seminar we do the bulk of the work in class, together. Your time outside of class is mostly to prepare you to take maximum advantage of class.
- A seminar succeeds only to the extent that everyone is prepared and energized. Therefore, if you're not fully engaged, the entire outcome of the course is changed—everyone loses. Therefore, when you decide to take the course, you have a moral obligation to be a full part of the group, not just a passive observer.

One way to view what is going on is that we have a semester-long conversation that leads us to consensus on:

- What work is good
- What techniques are useful
- How to exploit what we've learned

Why do we teach seminars? Primarily because this is the way real science is done: through extended, vigorous conversations about problems and ideas. And when it goes well, a seminar is among the most rewarding experiences you can have in a classroom.

## How will everyone be evaluated?

My evaluation of your work, and your final course grade, will be based on your class participation and on your contribution to an engaging final project. In detail, here is what I expect:

- You will come to each class prepared to contribute.

  Being prepared does not necessarily mean that you have understood all the reading and solved all the problems. It does mean that you can clearly articulate what you have understood or solved, and where your understanding or solution is incomplete, you can identify where the difficulty lies.
- In class, you will engage with your classmates in analysis, discussion, and problem-solving.
- You will not only contribute to discussions yourself, but you will also leave room for your classmates to contribute to the discussions.
- You will contribute to a final project that is clearly written and that demonstrates *either* technical depth, or an original idea, or command of the literature.

How might these expectations relate to your grades? Well, you are all experienced students and well versed in computer science, or you would not be eligible to take a seminar like this one. You probably also know how to contribute effectively in a small-group setting, and how to put together a class project. In my past experience with these kind of classes, a large majority of students have earned A's, and almost every student has earned at least an A-minus.

Finally, here's how I expect you to evaluate me:

- You should expect that I have put substantial thought and effort into the design and implementation of the class. In particular, you should expect that the learning goals I have identified will be enjoyable to explore and will be worthy of your time and effort.
- You should expect me to bias our reading and problem-solving toward the best available papers and the most illuminating problems. You should not, however, expect that all papers we read will be uniformly great—sometimes the great paper on an important topic has not yet been written, and sometimes I make a mistake. You can expect at least once during the term that I paper I thought would be good works out badly.
- You should expect me to make adjustments to the class schedule, as needed, to provide the best practicable support for everyone's learning.
- You should expect me to lead productive analysis and discussion in class. You should expect me to create and sustain an environment in which everyone has an opportunity to contribute.
- During class, you should expect me to guide you toward consensus opinions on the day's questions, problems, or paper. I should acknowledge not only the majority opinion but any significant minority views.
- You should expect me to be able to compare your consensus conclusions with the consensus of scholars working in the field, at least as long as the field is programming languages.[2]
- You should expect me to guide you in forming effective project teams and in choosing a project of appropriate scope.

[2] I am much less well able to represent the consensus of machine-learning scholars.

## What about the final project?

The ideal project will give you a chance to dig more deeply into whatever aspects of probabilistic languages attracted you to the course in the first place. Here are the ground rules:

- You may work on a project by yourself or on a team of any size.
- Your project will have a number of milestones.
  - Starting immediately after the October holiday, you will meet with each other outside of class and formulate tentative project ideas and project teams. By the time October is two-thirds over, you will have put three to six hours of work into your project—enough to have firm plans.
  - In late October, you will bring your firm plans to class, where you will engage in mutual presentation and criticism with one or two other project teams.
  - At the end of October, you will deliver a written proposal of one to two pages. The proposal will identify the members of your team and will describing the project you wish to undertake. To put forward a credible proposal, you will need to have invested in at least five to ten hours on your idea—plus writing time. I will either approve your proposal or ask you to revise it.

    I will approve your proposal provided it is in scope for the class and provided I believe the work you have proposed is a good match for the size of your time and the time you have available.
    - I don't think you need to worry about scope: The scope of this class is quite broad, and in my entire teaching career, I can remember only one instance in which students submitted a proposal that was out of scope for the class in which they were enrolled.
    - By contrast, it is relatively common for students to want to tackle a project that is too big for them to handle. If that happens to you, I will help you cut it down.
    - It's rare for students to propose a project that I think is too small. But if I do think your project is too small, I will help you find a way to enrich it.
  - By the first Monday in December, your project will be substantially complete. On **Friday, December 9**, you and your team will give a formal presentation of your project before your classmates and before a jury of external examiners. Expect to have 25 minutes to present and 5 minutes to answer questions. (If there are many one-person projects, I may be forced to cut the presentation time.)

    I expect everyone to attend all project presentations, so *please clear your calendar* from 10:00AM to 5:00PM on Friday, December 9.
  - On **Thursday, December 15**, you will send me a paper describing your project. The content of the paper will be guided both by general criteria that we will develop in class and by specific feedback you will receive from me and the jury, based on your presentation of the previous Friday.

  Meeting each of these milestones is mandatory, but the only part that receives a letter grade is the final paper. All the other milestones are ungraded.
- To grade your project paper, I will assess both the writing and the content. As the deadline approaches, I will state my expectations in great detail; for this syllabus, I summarize.

  My expectations for your writing are probably in line with the expectations for any other term paper, with one exception: I do not prescribe a length. Your paper should be just as long as it needs to be to say what needs to be said.

  My expectations for content is that your paper show solid achievement along *one* of the following three dimensions:
  - By demonstrating mastery of challenging ideas, you can show *technical depth*.
  - By highlighting an original idea and supporting it with a theorem, implementation, counterexample, definition, or other technical apparatus, you can show *originality*.
  - By citing and discussing a suitable collection of papers, and by demonstrating mastery of what they say, how they relate to one another, and how they relate to your project idea, you can show *scholarship*.

Any one of these three outcomes should lead to a top grade on your project; nobody should aim for all three.

The most challenging aspect of the project is that you need to identify something interesting before class is even half over. It is always easier to wait another week when you know more and can make more informed decisions. But every week you wait means one fewer weeks that you can actually work on the project itself. The key date is the day of scheduled for mutual criticism of proposals in class. I have tentatively scheduled that class for October 24, but depending on how things go, we might try to have it earlier.

If in the middle of November, you feel you would benefit from some class discussion of your project in progress, please let me know—we may be able to schedule something.

### Languages and systems that may be interesting for a project

Here are a number of languages and systems that are interesting potential substrates for a project. The systems that are most mature and most friendly to beginners are listed first:

- WebPPL is the implementation language of the electronic book The Design and Implementation of Probabilistic Programming Languages.
- Church is Scheme-like language with probabilistic semantics. I have my issues with the computational model, but it is the foundation of many other projects. An extensive tutorial can be found in the electronic book Probabilistic Models of Cognition.
- Anglican is inspired by Church and integrated with Clojure. It compiles to relatively efficient code for the Java Virtual Machine. There is a highly polished tutorial.
- Hakaru is a probabilistic language embedded in Haskell. Its distinctive feature is that it can do exact inference by symbolic analysis. We will read about this inference in class.
- Probabilistic C is intended as a compilation target for higher-level probabilistic languages. It is surprising how much can be accomplished by adding just a couple of primitives to C—and by exploiting parallelism.
- Wolfe is a probabilistic language embedded in Scala. It has a particularly clean way of talking about probabilistic domains.

A longer list, with other commentary, can be found at `probabilistic-programming.org`.

## What challenges should we expect?

Any small-group discussion class poses predictable challenges. In addition, a class like this one, which crosses disciplinary boundaries and is close to the leading edge of research, poses special challenges. Here are some that I know about:

- You will learn primarily from reading research papers. Reading research papers is difficult, but if you haven't done it before, you will find that your skills improve quickly with practice. In this class, we will focus primarily on probabilistic programming, not on learning to read research papers. I will give you a little bit of guidance, but if you want more, you will have to ask for it.
- For what we are trying to do, the class schedule will be very challenging. My experience shows that in a typical seminar of a dozen people, a good length of time to support deep class discussion is about 105 minutes. We have only 75 minutes and we have almost twice that many people. To use 75 minutes effectively, I am going to have to develop new teaching skills. That is an experimental process, and as I try different techniques to see what will work best, you can expect some missteps.
- Probabilistic programming draws from three disciplines: programming languages, real analysis (the mathematics of probability and integration), and machine learning. As your guide to the subject matter, I have a deep and broad understanding of programming languages. I have a reasonably firm grasp of real analysis—far more than we will need for this course. But I have limited knowledge of machine learning, and my education in machine learning is based primarily on statistical approaches in which probability does not play an obvious role. I know enough of the big picture to set directions—and I also have gotten significant help from real experts—but when we get into the details, I will be learning them at the same time that you are.

  In other words, I won't have all the answers.
- As noted above, some of your work will involve tackling small problems between classes. Inevitably, some problems will arise on a Monday for discussion in the following Wednesday's class. Such a short schedule is inconvenient, and I will avoid it when possible, but you would be wise to book some regular time on Tuesdays or on Wednesday mornings, to be sure that you are able to respond nimbly to the events of a Monday class.

## What do I need to know coming in?

This course builds on probability and programming languages. Ideally you remember this material from Discrete Math (61) and from Programming Languages (105), but if you missed it, here is what you need to review:

- You should recognize and be comfortable with simple algebraic laws for probability, especially the equation that relates "conditional" probability of A given B to the "joint" probability of A and B. It will also help if you have some idea how to compute the expected value of a given function against a given probability distribution.

  We will review all of this material in class.
- You should recognize some operational semantics, and you should be able to write a small operational semantics in consultation with a textbook.

  We will review operational semantics in class.
- You should be comfortable with functional programming, especially with pure data structures. You should remember something about higher-order functions like `map`, `filter`, and fold.

## What else can I do to succeed?

If you're nervous about giving a technical presentation, or if you'd just like some help, the ARC offers one-on-one appointments with public-speaking consultants who will help you with a presentation. You can sign up through Tutor Finder, which is supposedly on iSIS.

One of my former students also contributed this advice:

- The learning you do at home is directly proportional to the amount of learning you will do in class.
- Read the assigned materials before class. Understand the paper completely. Re-read the paper. Take notes in the margins. Bring your annotated paper to class.

The same former student also had these comments about his experience in class:

- Even if it feels like you're learning stuff that will not be applicable to your future life, you will likely be surprised.
- If I could re-name this seminar it would be: "How to solve hard problems." Computer Science is not easy, and by solving hard problems you become a better programmer.