A Course on Probability Theory for Computer Scientists

Mehran Sahami
Computer Science Department
Stanford University
Stanford, CA 94305
sahami@cs.stanford.edu

ABSTRACT
During the past 20 years, probability theory has become a critical element in the development of many areas in computer science. Commensurately, in this paper, we argue for expanding the coverage of probability in the computing curriculum. Specifically, we present details of a new course we have developed on *Probability Theory for Computer Scientists*. An analysis of course evaluation data shows that students find the contextualized content of this class more relevant and valuable than general presentations of probability theory. We also discuss different models for expanding the role of probability in different curricular programs that may not have the capacity to teach a full course on the subject.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education.*

General Terms
Algorithms, Theory, Human Factors, Design.

Keywords
Probability Theory, Computing Curricula, Discrete Mathematics

1. INTRODUCTION
During the past two decades, probability theory has come to play an increasingly important role in many areas of computer science. Understanding probability theory is not only useful for simulations, but it is becoming an essential tool for analyzing large-scale systems and is used as an integral part of many computing applications. Probability is also the cornerstone for most machine learning (a.k.a. data mining) techniques used to build predictive models from huge volumes of data. In this paper we describe a course on *Probability Theory for Computer Scientists*, explaining why such a course is needed and showing the efficacy of teaching this material in a computing context.

In Section 2, we present a case for why knowledge of probability theory and machine learning is critical for students studying computing today. Addressing this point, we describe a course on *Probability Theory for Computer Scientists*, which has been successfully offered three times in the past two years at Stanford University. The course is detailed in Section 3. We discuss the computing-specific applications presented in this course, distinguishing it from traditional courses in probability theory often taught in Mathematics or Statistics departments. Section 4 gives a comparative analysis of course evaluation data from several different probability theory courses at Stanford, providing evidence for the efficacy of a computing-centric course as opposed to general courses in probability theory. We conclude in Section 5 with thoughts on how themes from this course may be adopted in other contexts at other institutions.

2. THE NEED FOR PROBABILITY THEORY IN COMPUTING CURRICULA
Work in computing increasingly relies on probability theory as a tool for analysis and data modeling. Furthermore, probability theory now plays a foundational role in making advances in many areas of computing. For example, while probability was once simply used as a tool to analyze the average running time of algorithms, it has now become a central tenant in developing a new class of *randomized* algorithms [14]. In the area of Systems, probabilistic analyses of network routing and machine failures have become essential elements for building robust large-scale distributed systems, and will continue to become more important with the growth of "cloud computing". Much work in Artificial Intelligence is now based on probabilistic formalisms, encompassing work in reasoning, robotics, natural language understanding, and machine learning [18]. Specifically, the widespread use of data mining techniques for analyzing large data sets involves algorithms grounded in probability theory. Even application areas such as Graphics make use of random sampling in image rendering and work in Human-Computer Interaction involves probabilistic models of uncertainty in user behavior.

Despite the growing importance of probability theory in computing, the role of probability in the computing curriculum has remained relatively minor. Although both Computing Curricula (CC) 1991 and 2001 [1, 2] include probability as one of several topics to be included as part of a discrete math course, CC 1991 provides little guidance on the depth of coverage and CC 2001 suggests only six total hours of instruction. While some institutions suggest (or sometimes even require) that students take a full course in probability theory, the designated courses are often surveys taught in Mathematics or Statistics departments and do not offer any computing-specific applications. As a result, even students who receive a full course in probability theory are often at a loss with regard to how it relates to problems in computing, and this lack of context means students may less effectively learn and apply the material [6]. This point is further corroborated (later in this paper) by an analysis of data comparing the *Probability for Computer Scientists* course described here with several more traditional courses in probability theory.

Nevertheless, the importance of probability theory has not been lost in the Computer Science Education community, as some have previously suggested the incorporation of more concepts from probability into early computing courses [7]. However, that discussion focused more on the use of randomness to generate problem simulations in CS0 and CS1 courses as opposed to a formal treatment of probability theory in computing.

More recently, Anderson [4] has argued for combining probability with computing, suggesting a course in *Simulation, Probability and Statistics* as a means for students to satisfy a college "quantitative reasoning" requirement. The suggested course focuses primarily on building simulations, which is certainly useful, but quite different in content, applications, and goals from the course we present here.

Several textbooks on probability and statistics in computing have also been introduced in the past few years. Some of these texts [12, 16] are not introductory in nature, requiring previous coursework in probability. Other texts with a more introductory flavor [5, 10] tend to focus on simulation and queuing theory, without giving much, if any, coverage of machine learning.

While building randomized simulations is itself an important goal, the use of probability theory as a foundation for machine learning (i.e., building statistically valid models of data for prediction and insight) is a more central problem in the foreseeable future. A recent report by the NSF Task Force on Cyberlearning [15] points to the need to "prepare students for the data deluge" by giving them the mathematical tools to analyze data as part of their training in computing. The report goes on to state that "some of the world's largest companies (Google, Yahoo, Microsoft, Amazon, eBay) are struggling" with how to productively identify and exploit patterns in large amounts of data. The need for computer scientists with experience in statistical data analysis is also reported in a recent Wall Street Journal article [21], stating that "technology companies... are in hot pursuit of a particular kind of employee: those with experience in statistics and other data-manipulation techniques." Indeed, the article goes on to quote an executive at a high-tech recruiting firm who states that engineers "with strong statistics backgrounds will earn up to 20% more than generalist engineers", reflecting the strong industrial demand for software engineers with statistical modeling skills.

The ever-present discussions of Computational Learning also point to the importance of probability theory and machine learning. This point is forcefully made in Wing's seminal paper [22], which states that "machine learning has transformed statistics. Statistical learning is being used for problems on a scale, in terms of both data size and dimension, unimaginable only a few years ago. Statistics departments in all kinds of organizations are hiring computer scientists." As a result, we believe that giving students in computing a deep understanding of probability theory, grounded in real computing applications, is becoming increasingly important. From both the standpoint of preparing students for research in the field as well as meeting industrial demands, there is a need to give students greater preparation in probability theory and machine learning methods. We describe a course aimed at addressing this need presently.

3. A COURSE ON *PROBABILITY THEORY FOR COMPUTER SCIENTISTS*

3.1 Historical background

During the past two years we have taught three offerings of a course entitled *Probability Theory for Computer Scientists* at Stanford University. The course was created as part of recent curriculum revision of our undergraduate Computer Science program [19] and is a core class required for all CS majors. Prior to the curriculum revision, we required all CS majors to take a full quarter-long course in probability theory. Students could select any one of three classes to satisfy this requirement (all of which are rigorous calculus-based courses):

1. *Theory of Probability* – Offered by the Statistics department, this course provides a general introduction to probability theory without a computational component.
2. *Intro. to Probability and Statistics* – Offered by the program in Computational Mathematics, the course provides an engineering perspective on probability. The course uses MATLAB for computational work.
3. *Probabilistic Analysis* – Offered by the Management Science and Engineering department, the course emphasizes probabilistic model building and uses Microsoft Excel for computational work.

While all of these courses provide students with significant exposure to probability theory, even from various engineering perspectives, we often heard that students had difficulty recalling and applying the material from these courses in subsequent computing classes (e.g., algorithms, artificial intelligence, etc.). Since the probability courses were developed for more general audiences, they, understandably, provide no motivating examples that are computing-specific. Moreover, the computational work provided in some of the courses is more focused on simulation and does not address the issue of analyzing data to build predictive models (i.e., machine learning).

To remedy this issue, we developed a new course, specifically for computer science students, which we describe shortly. As mentioned previously, the new course is now required for all our undergraduate computer science students (and is also strongly suggested for Masters students who may not already have a solid background in probability). CS students no longer have the option of taking the other three probability courses, although students who took such a course prior to the availability of our new class were "grandfathered" into satisfying the requirement.

3.2 Course Prerequisites

As with the other probability course options previously available, the new course requires a background in calculus, which was already a mathematics requirement for our program. However, the new course, being aimed at computer scientists, also has CS2 as a prerequisite. CS2 provides a critical level of background in computing that allows for excellent contextually-relevant motivating examples in the probability course. For example, the probabilistic nature of hash functions, insertions or look-ups into ordered data structures (e.g., lists, binary search trees), and the randomized selection of pivots in QuickSort all provide rich examples that we analyze in our new course. Furthermore, by having CS2 as a prerequisite, we guarantee that all students have a

Table 1. Weekly outline of *Probability Theory for Computer Scientists* course

Week	Topics	Sample of Motivating Examples
1	Counting, Combinatorics (Combinations/Permutations), Intro. to probability, Sample spaces, Axioms of probability	Counting degenerate Binary Search Trees, Sharing a birthday, Likelihood of poker hands, Sampling computer chips for defects
2	Conditional probability, Bayes Theorem, Independence, Discrete random variables, Expectation, Variance	Analyzing hash table load, Email spam detection, Arrangements of bit strings, HIV testing, "Monty Hall problem", Value of lotteries
3	Discrete probability distributions (Binomial, Multinomial, Poisson, Geometric, Negative Binomial, Hypergeometric)	Flipping coins, St. Petersburg paradox, Error-correcting codes, Packet corruption in networks, Web server overload
4	Continuous random variables and probability distributions (Uniform, Normal, Exponential), Joint distributions	Disk crashes and expected lifetime, Likelihood of error in digital/analog conversion, Probabilistic text analysis
5	Independent random variables, Conditional distributions and independence, Expected algorithm running times	Distributing requests in a cluster, Recommendation engines, Hash tables as "coupon collecting", QuickSort expected running time
6	Covariance, Correlation, Conditional expectation, Moment generating functions	Computer cluster utilization, Analyzing recursive functions, Simple predictive models, Optimizing hiring (software engineers)
7	Inequalities (Markov, Chebyshev, Chernoff, Jensen), Law of Large Numbers, Central Limit Theorem	Analyzing midterm scores, Modeling a risky investment, Estimating algorithm clock running time via repeated trials
8	Prior probabilities, Parameter estimation (Method of moments, Maximum likelihood, Maximum a posteriori)	Number of idle machines in a computer cluster, Estimating probabilities from rolling dice, "Two envelopes" problem
9	Intro. to Machine Learning, Naive Bayesian classifier, Logistic regression, Simple Bayesian networks, Utility	Predicting tomorrow's weather, Email spam filtering, Simpson's paradox in data analysis, Utility of lotteries
10	Computational generation of probability distributions, Monte Carlo simulation	Generating other distributions using only the `rand()` function, Algorithms for random shuffles, Monte Carlo integration

reasonable foundation in programming. This allows for class examples and homework problems that involve the direct probabilistic analysis of code. Moreover, it allows us to have significant programming projects, where students implement algorithms that build predictive models of real-world data and then analyze the results of their programs.

3.3 Course Details

3.3.1 Structure

The *Probability Theory for Computer Scientists* course is taught during a 10 week quarter, and has three 75 minute lectures (including interactive demonstrations) each week. A weekly outline for the course is shown in Table 1, showing both the topics covered as well as a sampling of the examples that are used to motivate the material in class. A complete set of course slides providing detailed course contents is available on the web [3].

3.3.2 General introduction to probability

As seen in Table 1, the topical coverage of the course during the first seven weeks aligns with many traditional probability theory courses. In fact, the textbook [17] we have used for our course is a general probability text that is also used in two of the other three probability courses taught at Stanford (previously mentioned).

What distinguishes our course during the first seven weeks is that the vast majority of examples used to motivate and explain the various topics covered are drawn from computer science. For example, in the first week, while we examine the traditional problems of computing the probability of various poker hands (via counting) or determining the probability that two people in a room share the same birthday, we also examine the number of degenerate (i.e., linear) Binary Search Trees that can be generated to show how slowly that number grows in comparison to the total number of possible Binary Search Trees. The following week, we analyze the well-known "Monty Hall" problem [13], but also look at questions related to how evenly elements are likely to be spread across buckets in a hash table. In the weeks following, there is a steady stream of real-world computing problems presented, including how error-correcting codes are used to increase the probability of robust communication in networks, determining the probability that web servers will become overloaded under particular distributions of requests, how probability is used in recommendation engines and email spam filters [20], as well as showing how to analyze the expected running time of algorithms.

These examples significantly engage students as they see the role probability plays in many of the computing technologies presented in their other courses as well as applications they use on a regular basis (e.g., email, on-line shopping). More importantly, by seeing probabilistic concepts in a domain that students are already familiar with, they have a framework to better assimilate the salient aspects of the new information they receive. They are no longer learning probability in a vacuum (using the common notion of "balls and urns"), but can directly relate it to real problems in computing they will likely have to face. By learning to identify which probabilistic aspects of a problem (e.g., independence, conditioning, etc.) give rise to which issues in real computing applications, they come away better prepared to tackle problems in the future. Moreover, they gain a greater appreciation for the significant role that randomness plays in a domain that they tend to think of as strongly deterministic.

During the course, we do also discuss classic problem set-ups in probability, such as "coupon collecting" and reasoning about distinguishable vs. indistinguishable "balls" in urns. However, each of these cases is also directly related to applications in computing. In this way, students not only see the general concepts, but also look at analogous computing-specific examples to gain a firm grounding of how such problems arise in the real world. As a result, they gain experience mapping general concepts to solutions of actual problems, a key ingredient missing in the general probability courses students were previously taking.

3.3.3 Introducing Machine Learning

The last three weeks of the course leverage what students learned during the prior seven weeks to introduce them to Machine Learning, specifically building classification models from data. *Classification* is the task of predicting a particular discrete value (i.e., the "class" or "prediction output") as a function of a set of observed input variables. For example, we might consider building a model to predict if the weather tomorrow will be "sunny", "overcast", or "rainy" based on variables that can be measured today (e.g., average temperature, humidity, if the sky is currently cloudy, etc.). Such prediction models are "learned" (i.e., parameters are estimated) using historical data to help determine how the prediction output is probabilistically related to the input variables. Another common example of classification in a computing context is the detection and filtering of email spam. Indeed, most modern email servers treat spam detection as a classification problem, where the server must classify each email message as "spam" or "legitimate" based on characteristics of the message (e.g., the sender, the words contained in the messages, which SMTP server it was received from, etc.)

Prior to discussing classification tasks, however, in week 8 of the course we get students to understand that in real-world applications of probability, the parameters for distributions (e.g., mean, variance, covariance, etc.) are not given, but rather must be estimated from data. This begins a week of in-depth investigation of various parameter estimation techniques and their mathematical properties. It also provides a natural context for discussing Bayesian techniques and the role of subjective prior probabilities. Indeed, some students have difficulty grappling with the general notion of subjective probabilities at first. But through grounded examples we show how people naturally make use of such probabilities and how they are mathematically incorporated into probabilistic inference, giving students a concrete, yet formal treatment that they are more comfortable with. Moreover, we explain various probabilistic smoothing techniques as forms of parameter estimation with the incorporation of subjective prior probabilities (e.g., Laplace smoothing [11]).

Armed with knowledge about how data can be used to (computationally) estimate parameters of probability distributions, students are then introduced to the classification task and specific probabilistic models for addressing this problem. Specifically, we focus on the Naive Bayesian classifier [8] and Logistic Regression models [9]. We start with the Naive Bayesian classifier, as this model is simple to understand. Given a set of *n* input variables X_1, X_2, ..., X_n, the Naive Bayesian classifier predicts the class (output) c_i that maximizes the probability $P(X_1, X_2, ..., X_n, \text{Class} = c_i)$ over all choices of *i* (i.e., all possible values of the discrete output variable) under the assumption that all the input variables are independent of each other, conditioned on the variable *Class*. More formally, the Naive Bayesian classifier predicts the output c_i given by:

$$\underset{c_i}{\operatorname{argmax}}\ P(X_1, X_2, ..., X_n \mid \text{Class} = c_i) \cdot P(\text{Class} = c_i)$$

under the "Naive Bayes assumption" that:

$$P(X_1, X_2, ..., X_n \mid \text{Class} = c_i) = \prod_{j=1}^{n} P(X_j \mid \text{Class} = c_i)$$

Importantly, this model shows the power of conditional independence (which students learned about earlier in the course) as a means of reducing the number of parameters needed to be estimated in the model from exponential to linear in *n* (the number of input variables). Computer science students, already having an appreciation for the extreme difference between exponential and polynomial running times, quickly appreciate how the simplifying assumption made in the Naive Bayes model can lead to a dramatic change in the efficiency of probabilistic inference.

After presenting the Naive Bayesian classifier, students then see the Logistic Regression model, which has many mathematical similarities to Naive Bayes, but a few critical differences that prevent its parameters from being estimated analytically (as can be done with Naive Bayes). As a result, Logistic Regression immediately forces a computational solution in which parameters are optimized using gradient descent over a convex objective function. To better understand the differences between the two models and gain hands-on experience with the issues that arise in doing actual data analysis, students implement both Naive Bayes and Logistic Regression and estimate their parameters using provided real-world data sets (described in more detail below).

The course concludes with a discussion of how various probability distributions (e.g., Poisson, Normal, Exponential) may be computationally generated using only the uniform random distributions that are provided by a function such as the C standard library's **`rand()`**. This also provides an opportunity to discuss in detail the generation of pseudo-random numbers and the strengths and weaknesses of various random number generation techniques. Along the same lines, we also discuss how algorithms for common tasks involving randomization, such as shuffling a deck of cards, are often *incorrectly* implemented by programmers, as we can rigorously analyze the properties of such algorithms that on the surface are easily mistaken for producing a truly random shuffle (where all permutations of the cards are equally likely). Finally, we show how Monte Carlo simulation (making use of various random number generators) can be used to approximate the solution to many mathematical problems (e.g., evaluating an integral). Thus, while we do discuss simulation (which is also a topic touched on at other points in the course as well), it is not meant to be the central theme of the course.

3.3.4 Demonstrations

While the course is organized using primarily a traditional lecture-based format, the lectures also include a number of computer and interactive demonstrations to give students a better grounding in the material. These demonstrations include the "classics", such as selecting a student from the class to actually try the "Monty Hall problem" using three envelopes (where one contains money and the other two are empty), as well as determining if two students in the class have the same birthday.

We also consider more modern examples, such as providing a computer simulation of how sampling distributions are generated to show the Normality of such distributions as predicted by the Central Limit Theorem. While the Central Limit Theorem is one of the most powerful results in probability, it is often misapplied by those confused by the difference between a *sampling* distribution and the *underlying* distribution that the samples are drawn from. To clarify this issue, we play a simple game where students (in front of the class) roll 10 fair 6-sided dice to see if they can roll a total less than 25 or greater than 45 (and thereby win a prize). This demonstration allows students to map theoretical concepts to a physical process (i.e., the 6-sided die represents the underlying distribution, the 10 rolls represents a sample from the distribution, and the average of the 10 rolls is a value from the sampling distribution of the mean), allowing them to better remember the material. The physical demonstration is then reinforced by the computer application which they can use to try various other underlying distributions and see that the sampling distribution of the mean tends to always be Normal. Students report that class demonstrations help them better remember the material, especially when they have the opportunity to follow-up on them afterwards.

3.3.5 Assignments

There are six assignments in the course, spanning a range of problem modalities. Most assignment problems are word problems that involve analytically deriving a mathematical result based on appropriately modeling the probabilistic dynamics of the problem. What distinguishes such problems from a more traditional probability course is that a large majority of them involve realistic problems in computing, such as analyzing the distribution of elements in hash tables and determining request distributions to web sites. Also, in such problems, students are encouraged (but not required) to write simulation programs to verify their answers. Several assignment problems also involve the direct analysis of code. For example, students may be asked to determine the expected running time of a stylized recursive algorithm or determine if an algorithm is correct with respect to modeling some probabilistic process.

In the final portion of the course, where the focus shifts to Machine Learning, students are required to implement various learning algorithms (specifically, the Naive Bayesian classifier and Logistic Regression, as mentioned above) and test their implementations using real-world data sets. In the past, students have built models using data to predict Congressmembers' political affiliation based on their voting records as well as tackling medical diagnosis tasks, such as determining if a patient has a heart abnormality based on tomography (X-ray) data. Other tasks considered for future courses include building an email spam filter or predicting the locations of splice junctions in DNA.

Students have their choice of implementing their algorithms in Java, C/C++, or R (a functional statistical modeling language). All of the students in the course are expected to have worked with Java or C/C++ previously (as a result of CS2 being a prerequisite). We also allow R as an option for programming assignments as we have recently started to offer an optional one unit adjunct lab course entitled *Statistical Programming with R*.

The R course provides two main benefits for students. First, it provides them with an opportunity to see a more direct relationship between programming and the statistical theory they are learning, as the topics covered in the R course mirror those covered in the probability course. This parallel structure also allows many examples in the R course to be relevant to building computational simulations to verify analytical results of problems in the probability course. Second, the course provides a convenient way to expose students to functional programming in a context which is not just about learning programming, but motivated by solving real statistical problems.

4. COURSE EVALUATION ANALYSIS

To evaluate the efficacy of our course, we compare end-of-quarter student evaluation data from our *Probability Theory for Computer Scientists* course against all three other probability theory courses taught at Stanford. Since we want the data from the three other probability theory courses to reflect evaluations from CS students (to create a more direct comparison to the population of our new course), we consider every offering of the other three courses in the year and a half *prior* to the introduction of our new course (Fall 2007-08 through Winter 2008-09), when a substantial portion of the population of those other courses would still be CS students. We then compare the data we have on every offering of our new course over the immediately following year and a half period (Spring 2008-09 through Spring 2009-10). We denote the courses compared as follows:

- *Theory of Probability* – Denoted **TP**. The course was offered three times from Fall 2007-08 through Winter 2008-09. A total of 184 student evaluations were collected.
- *Intro. to Probability and Statistics* – Denoted **IPS**. The course was offered twice from Fall 2007-08 through Winter 2008-09. A total of 191 student evaluations were collected.
- *Probabilistic Analysis* – Denoted **PA**. The course was offered twice from Fall 2007-08 through Winter 2008-09. A total of 163 student evaluations were collected.
- *Probability Theory for Computer Scientists* – Denoted **PCS**. The course was offered three times from Spring 2008-09 through Spring 2009-10. A total of 277 student evaluations were collected.

In the evaluation data, we examine questions specifically related to the value of the course and the relevance of course content, which we believe are the most meaningful indicators for comparing across courses in this context. Using a 5-point scale (5 = Excellent, 4 = Very Good, 3 = Good, 2 = Fair, and 1 = Poor), students rated several course criteria. Table 2 presents these criteria (column 1), and the average score received in each of the 10 class offerings (columns 2 through 11 in the table each represents a distinct offering of a course). It is worth noting that *every* offering of the *Probability Theory for Computer Scientists* course (**PCS 1/2/3**) ranks higher on all of the five criteria than *any* of the more general probability theory courses.

To determine the significance of these results, we compute t-tests between the data for each of the four courses under consideration. Since it is infeasible to report the results of comparing every class offering versus every other class offering, we first pool the student evaluation data on a *per course* basis. This also provides the benefit that data is focused more on the content of the course in general as opposed to the particulars of any one offering. The pooled per-course data is given in columns 12 through 15 of Table 2. We compute t-tests (unpaired, heteroscedastic, two-tailed) on the pooled data for PCS vs. TP, IPS, and PA, respectively. The t-test results (p-values) are given in columns 16 through 18 of Table 2. On every criteria, PCS scores statistically significantly higher than any of the other probability courses with $p < 0.001$, providing strong evidence that students find the contents of the new course more relevant, valuable, and related to their studies in computing.

Table 2. Student evaluation and t-test results for probability courses.

Criteria \ Course Offering	TP (1)	TP (2)	TP (3)	IPS (1)	IPS (2)	PA (1)	PA (2)	PCS (1)	PCS (2)	PCS (3)	TP (all)	IPS (all)	PA (all)	PCS (all)	PCS vs. TP (all)	PCS vs. IPS (all)	PCS vs. PA (all)
The quality of the course content	3.60	3.58	4.17	4.29	4.06	3.41	3.84	4.53	4.32	4.51	3.80	4.18	3.60	4.45	< 0.001	< 0.001	< 0.001
Set out and met clear objectives announced for the course	3.00	3.57	4.22	4.44	4.23	3.60	3.76	4.62	4.57	4.61	3.58	4.34	3.67	4.60	< 0.001	< 0.001	< 0.001
Emphasized conceptual understanding and critical thinking	3.36	3.50	4.01	4.20	4.14	3.45	3.81	4.52	4.41	4.56	3.63	4.17	3.61	4.49	< 0.001	< 0.001	< 0.001
Related course topics to one another	3.17	3.52	4.14	4.31	4.24	3.40	3.72	4.59	4.45	4.79	3.61	4.27	3.54	4.57	< 0.001	< 0.001	< 0.001
Selected course content that was valuable and worth learning	3.37	3.50	4.21	4.25	4.07	3.28	3.64	4.55	4.51	4.69	3.70	4.13	3.43	4.56	< 0.001	< 0.001	< 0.001
# Evaluations (N)	75	43	66	102	89	90	73	131	101	45	184	191	163	277			

It is also worth noting that while the class was originally designed for CS students, it does attract a number of non-CS majors who report that they are interested in the Machine Learning aspect of the course, which is not available in the other probability courses.

5. CONCLUSIONS

Probability theory and machine learning will continue to grow in importance for students studying computing. We believe the course presented here helps effectively prepare students for applying probability in computing contexts and using it as a tool for data analysis and modeling. The comparative analysis of course evaluation data shows that students are finding the new course content to be more relevant and valuable than more general courses in probability. Materials from our course are available on-line and have been freely distributed in various educational forums. Indeed, we have already learned of at least one other university planning to offer a course this coming year modeled on the one described here, and several other programs that are considering similar courses in the future.

Understanding that some CS programs cannot justify a full course in probability, we would encourage an expanded coverage of the topic with more computing-relevant examples in existing discrete math courses. In such contexts, probability can be taught using only *discrete* variables, which still provides sufficient foundation for discussing topics in Machine Learning (parameter estimation and Naive Bayes). In fact, such a treatment can be provided without requiring a background in calculus.

Alternatively, some schools have reported that they are planning a move toward the model described here by restructuring existing course offerings. For example, rather than requiring a full course in *Automata Theory and Computability*, these topics are now covered in a condensed treatment in an existing discrete math course (in place of probability) and the probability material is expanded into a separate course (perhaps with the inclusion of some additional topics). We believe there will be many models that will work for providing expanded coverage of probability theory in the computing curriculum.

6. REFERENCES

[1] ACM/IEEE-CS Joint Curriculum Task Force. 1991. Computing curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force, ACM Press, NY.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2001. ACM/IEEE Computing Curricula 2001 Final Report. http://www.acm.org/sigcse/cc2001.

[3] http://ai.stanford.edu/users/sahami/cs109/

[4] Anderson, S. D. 2007. A course on simulation, probability and statistics. *SIGCSE Bull*. 39, 1 (Mar. 2007), 110-114.

[5] Baron, M. 2006. *Probability and Statistics for Computer Scientists*, Chapman and Hall/CRC Press.

[6] Carraher, T. N., Carraher, D. W., and Schliemann, A.D. 1985. Mathematics in the streets and in the schools. *British Journal of Developmental Psychology* 3.

[7] Ginat, D., Anderson, R., Garcia, D., and Rasala, R. 2005. Randomness and probability in the early CS courses. In *Proc. of SIGCSE '05*, 556-557.

[8] Good, I. J. 1965. *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*. MIT Press, Cambridge, MA.

[9] Hosmer, D. W. and Lemeshow, S. 2000. Applied Logistic Regression (2nd Ed.), John Wiley & Sons, Hoboken, NJ.

[10] Johnson, J. 2008. *Probability and Statistics for Computer Science*, John Wiley & Sons, Hoboken, NJ.

[11] Lidstone, G.J. 1920. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Trans. of the Faculty of Actuaries* 8.

[12] Mitzenmacher, M. and Upfal, E. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press, New York, NY.

[13] http://montyhallproblem.com/

[14] Motwani, R. and Raghavan, P. 1995. *Randomized Algorithms*, Cambridge University Press, New York, NY.

[15] NSF Task Force on Cyberlearning. 2008. Fostering Learning in the Networked World: The Cyberlearning Opportunity and Challenge. http://www.nsf.gov/pubs/2008/nsf08204/nsf08204_1.pdf

[16] Ross, S., 2001. *Probability Models for Computer Science*, Academic Press, San Diego, CA.

[17] Ross, S., 2009. *A First Course in Probability (8th Ed.)*. Prentice Hall, Englewood Cliffs, NJ.

[18] Russell, S. and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach (3rd Ed.)*, Prentice Hall, Englewood Cliffs, NJ.

[19] Sahami, M., Aiken, A., and Zelenski, J. 2010. Expanding the frontiers of computer science: designing a curriculum to reflect a diverse field. In *Proc. of SIGCSE '10*, 47-51.

[20] Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. 1998. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*. AAAI Press Technical Report WS-98-05.

[21] Vascellaro, J. E., April 8, 2010. *New Hiring Formula Values Math Pros: Region's Employers Seek Statistical Experts Over Computer-Science Generalists*. Wall Street Journal (San Francisco Bay Area).

[22] Wing, J. M. 2006. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006).