Check for updates

Original Research Article

Evaluation Review
2022, Vol. 46(3) 296–335
© The Author(s) 2022
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0193841X221082887
journals.sagepub.com/home/erx
SAGE

A Combinatorial Optimization Framework for Scoring Students in University Admissions

Lucy Shao[1], Richard A. Levine[2], Stefan Hyman[3], Jeanne Stronach[4], and Juanjuan Fan[5]

Abstract

Background and Objectives: Selecting applications for college admission is critical for university operation and development. This paper leverages machine learning techniques to support enrollment management teams through data-informed decision-making in this otherwise laborious admissions processing. **Research Design and Measures:** Two aspects of university admissions are considered. An ensemble learning approach, through the SuperLearner algorithm, is used to predict student show (yield) rate. The goal is to improve prediction accuracy to minimize over- or under-enrollment. A combinatorial optimization framework is proposed to weigh academic performance and experiential factors for ranking and selecting students for admission. This framework uses simulated annealing, and an efficacy study is presented to evaluate performance. **Results:** The proposed framework is illustrated for selecting an incoming class by optimizing

[1]Division of Biostatistics, Herbert Wertheim School of Public Health and Human Longevity Science, University of California San Diego, San Diego, CA, USA
[2]Department of Mathematics and Statistics, San Diego State University, San Diego, CA, USA
[3]Enrollment Student Services, San Diego State University, San Diego, CA, USA
[4]Analytic Studies & Institutional Research, San Diego State University, San Diego, CA, USA
[5]Department of Mathematics and Statistics, San Diego State University, San Diego, CA, USA

Corresponding Author:
Richard A. Levine, Department of Mathematics and Statistics, San Diego State University, 5500 Campanile Drive, San Diego, CA 92182, USA.
Email: rlevine@sdsu.edu

predicted graduation rate and by developing an eligibility index. Each example presents a selection process under potential academic performance and experiential factor targets a university may place on an admitted class. R code is provided for higher education researchers and practitioners to apply the proposed methods in their own settings.

Keywords
ensemble learning, enrollment management, simulated annealing, SuperLearner, yield rate

Introduction

University admissions, particularly at selective universities with larger applicant pools, entails scoring, ranking, and selecting applicants based on student characteristics and predictions of future success at the institution. (We refer the reader to Rigor 2003 from the College Board Admissions Models Project, Hossler and Bontrager 2015 for the enrollment management perspective, and Selingo 2020 for a higher education journalist's perspective.) At its core, a student scoring system boils down to assigning points for behavior the university wishes to reward in its applicants, be it academic performance or attributes. The process is not simple as most universities have commitments and initiatives that, from a statistical learning perspective, place constraints on the ranking system. Universities may wish to maintain a minimum level of academic prowess, have a mission of serving local region students, award scholarships for specific student subgroups, or aim to achieve a level of diversity, equity, and inclusion.

In this paper, we pose this admissions problem within a combinatorial optimization framework. The optimization problem is to maximize success for the admitted class at the university. The optimization aims to identify a point structure on student characteristics that maximizes success for the admitted class at the university, while working within specified targets on the makeup of the incoming class. By combinatorial, we are recognizing that the optimization routine must scan over a multi-factor admission scoring of academic performance variables and student demographics. In particular, the algorithm must search over a high-dimensional combination of potential points to assign to each variable.

Consider the following example. Suppose we wish to rank students for admission based on a score composed of high school GPA, number of STEM courses taken in high school, and identifying as a first-generation college student. We may wish to assign points for each of these three scoring model inputs. The optimization routine scans over combinations of points on each of these three variables, with a goal of maximizing predicted 4-year graduation success. Our admissions process may have enrollment targets; for example, the university may wish to produce an incoming class that consists of at least

40% of students from the local service region and a minimum number of admissions to a scholarship program in a local school district. These goals need to be included in the optimization routine.

This type of optimization problem occurs across the sciences and engineering, a number of algorithms proposed to quickly and efficiently explore the space of, in our case, variable point combinations while achieving pre-specified subgroup targets. Though a slightly older presentation, we like the exposition of Dreo et al. 2006, presenting the core algorithms in the combinatorial optimization literature. We will focus on simulated annealing (Guilmeau et al. 2021), a combinatorial optimization routine with which we have had success in statistical decision-making. The simulated annealing algorithm is easy to code and introduces a flexible randomization element to direct the computer to intelligently jump around the space of point combinations.

Upon scoring and ranking applicants, a critical component to selecting students for admissions is predicting the probability that each student will accept an invite; this is called the yield rate or show rate. Poorly estimated show rates could lead to over- or under-enrollment of target class sizes, each lending to potentially significant costs especially in resource-constrained environments universities often find themselves. In our combinatorial optimization scheme, show rate prediction is crucial to satisfy the desired academic quality and student subgroup distributions in the selection of the incoming class. Machine learning approaches have found great success in predictive analytics problems. In particular, ensemble learning approaches which combine predictions from a suite of models has shown a proclivity for improving predictive performance over a single machine learning tool. We introduce a SuperLearner algorithm (E. C. Polley et al. 2011) for our particular application of estimating the show rate.

In the *Methodological Set-Up and Literature Review* section, we further motivate the university admissions problem and briefly review the literature on machine learning solutions. In the *Methods* section, we introduce our SuperLearner show rate model and our simulated annealing algorithm for scoring and ranking university applicants. In the *Examples of Applications in Admissions Practice* section, we illustrate our combinatorial optimization framework in two primary settings: selecting applicants relative to predicted university graduation success and constructing an admissions scoring formulation. As part of these applications, we further assess the efficacy of our proposed approaches. In the *Discussion* section, we discuss the modular nature of our optimization framework and conclude with final thoughts. For the reader interested in more detail, in the Appendix, we present a section evaluating the performance of our simulated annealing algorithm through a series of simulation experiments mimicking realistic bonus point structures in admissions processing. We also present sections with more mathematical details on our proposed algorithms and a section with R code for putting our methods into practice.

Methodological Set-Up and Literature Review

We broadly envision university admissions in two steps: rank students relative to the desired scoring criterion and then admit students based on an estimated yield or show rate. These tasks each entail a machine learning process for predicting student success (scoring criterion) and the probability a student accepts an invite for admission (show rate), respectively. We will place our work within the literature concerning these two statistical modeling endeavors.

On Admissions Modeling

Student ranking for university admissions is based on a balance of student academic performance and experiential/environmental factors. For example, a student is evaluated through high school measures such as high school GPA, standardized tests (SAT, ACT, and subject-specific state or College Board exams), successful completion of subject area courses (e.g., STEM), and sitting for advanced placement exams. Universities also weigh regional distribution of students (local region and in-state/out-of-state/international), adversity measures (first-generation college students, veterans, sociodemographics, etc.), and target support programs. Concerning the latter, the College Board previously introduced Landscape as a collection of high school and neighborhood information for colleges and universities (cb.org/landscape, visited on 8/10/2021). We mention the College Board's attempts at an adversity index to place these experiential factors into context. The precise means of incorporating equity and inclusion measures in university admissions scoring is a critical discussion complimentary to the proposed statistical approaches. Our point is that upon selecting features on which to score students for academic performance and for experiential factors, a ranking of students for admissions may be presented as a constrained combinatorial optimization problem within which these two measures may be objectively balanced.

To our knowledge, there is limited literature on statistical approaches to scoring students for admissions relative to academic performance and experiential factors. A classic paper (Bruggink and Gambhir 1996) constructs a logistic regression model to estimate the probability of acceptance using both academic and non-academic variables, and then estimate the student's probability of enrollment for admitted students. Two other papers also consider machine learning approaches for this admissions problem relative to graduate programs. University of Texas at Austin researchers (Waters and Miikkulainen 2013) apply logistic regression to identify strong candidates to admit. They also consider non-subjective factors such as recommendation letters and statements of purpose in the process. For the latter unstructured data, they performed text mining, creating a bag of words for each text document, and

then implemented dimension reduction via latent semantic analysis to condense the text into a 50-dimensional variable. Northeastern University researchers (Zhao et al. 2020) create models of quantitative student features using variants of support vector machines that rank and score students to admit.

While a direct machine learning approach for scoring students relative to academic performance and experiential factors is desired, our additional challenge is developing a model that prioritizes student attributes, and balancing it with other institutional commitments. For example, a public university may be mandated to maintain a reasonable proportion of in-state or regional students in an incoming class. A university may have a goal of maintaining or improving upon its diverse population. A university may also wish to balance an incoming class's academic performance, say summarized by an average high school GPA, relative to experiential factors. In these situations, using statistical modeling terminology, we must fit a model to score students subject to constraints. These modeling constraints may be specified as, for example, the percentage of regional students or the percentage of first-generation college students above a specified threshold among the students ultimately accepting admission. Straightforward application of standard machine learning tools, including logistic regression modeling, support vector machines, and random forest, cannot impose these constraints. Thus, we propose a simulated annealing combinatorial optimization modular approach that can score students using a machine learning method and features of choice while developing a scoring system under specified population targets (modeling constraints). Relative to the previous work on graduate admissions, which are aimed at smaller application pools, we develop our approach to handle a potentially large pool of undergraduate applications.

Simulated annealing is a flexible routine for solving complex combinatorial optimization problems, introduced in the early 1980s. Though we are not aware of applications quite along the lines of ours where we wish to construct a score and rank individuals (or plans or objects or systems) relative to distributional and performance targets, applications along these lines appear for selecting expert or recommendation systems relative to risk or cost profiles. To give the reader an idea of the broad array of scientific applications, we present here some recent examples. Shahandashti and Pudasaini (2019) rank water pipe leaks susceptible to deterioration from seismic activity, under budget constraints. Zhang et al. (2018) select a hybrid renewable energy system with respect to energy storage, energy supply, and life-cycle cost. Ghorbani and Jokar (2016) select inventory management solutions relative to the supply chain which includes inventory location, routing, backlogging, and demand. Other applications of simulated annealing include business risk assessment (Erana-Diaz et al. 2020), network alignment in bioinformatics (Mamano and Hayes 2017), and construction and manufacturing systems (Benderbal et al. 2018; Piryonesi et al. 2018).

On Show Rate Modeling

Many methods have been proposed for enrollment prediction. Aksenova et al. (2006) utilize curve-fitting techniques and causal models. More recent literature utilizes machine learning classification techniques to predict enrollment rates. Haris et al. (2016) introduce the enrollment prediction process and predicted enrollment using a decision tree. Similar studies were done by Saini and Jain (2013) and Soltys et al. (2021), though using a variety of machine learning methods including CART, naive Bayes, *k*-nearest neighbors, artificial neural networks, support vector machine, boosting, and logistic regression for enrollment rate prediction. These works focus more on the whole cross-industry standard enrollment prediction process for data mining; our work will focus specifically on the methodology side.

For our show rate model, we propose to use ensemble learning techniques to make the yield prediction. Ensemble learning combines multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. We will focus particularly on SuperLearner (E. C. Polley et al. 2011), a flexible yet easy to use ensemble learning package that has shown good predictive performance. Our development is aimed at the practitioner with exposition of the algorithm, guidelines on utilizing diverse learners, and R code for higher education applications.

Note that the following examples and model information are not what San Diego State University does in practice. That said, our goal is to illustrate the modular nature of our model structures via realistic examples, and provide readers a general view of how they can apply the approaches in production.

Methods

SuperLearner for Show Rate Modeling

This section assumes a background in machine learning terminology. In the appendix, we review these concepts for the interested reader.

Ensemble learning combines multiple machine learning models to improve model stability and predictive power. These individual machine learning models are called base learners, and the ensemble model is called the meta learner. Examples of ensemble learning include stacking, boosting, and bagging. Contrasting with bagging, stacking base learners are typically different and fit on the same data set. Contrasting with boosting, stacking uses a single model (meta learner) to learn how to best combine the predictions from the contributing models (Rokach 2019). Stacking usually involves two levels of models: the level-0 models (base learners), which fit on the training data and produce individual sets of predictions on testing data, and the level-1 model (meta learner), which learns how to combine each individual set of predictions obtained from level-0 models.

For classification problems, ensemble learning improves accuracy over a single classifier (Moon et al. 2007). In regression problems, the ensemble learner may not always win; a number of approaches are available with an aim of improving prediction accuracy. A common theme in ensemble learning is to combine a diverse set of learners, where predictions are not too highly correlated and the overall ensemble prediction may benefit from individual model predictive performance (see, for example, Sagi and Rokach (2018)).

We will lean on the SuperLearner algorithm (E. C. Polley et al. 2011) and related R package (E. Polley et al. 2021), a popular ensemble learning model. The SuperLearner algorithm first fits individual learners through a cross-validation routine, resulting in predictions from each individual learner for each observation. The SuperLearner algorithm then combines these predictions through weights on each individual learner as optimally determined by a meta learner. This "ensemble" model produces the final prediction. This model often is at least as good as the best machine learning model that is tested (E. C. Polley et al. 2011). Algorithm 1 outlines the SuperLearner algorithm.

Algorithm 1 SuperLearner

1. Choose m base learners.
2. Randomly split data into 10 blocks (to prepare for cross-validation)
3. Do the following (k-fold cross-validation) for each base learner:
4. **for do** each i from 1 to 10
5. Remove the ith block of data.
6. Train m base learners on remaining nine blocks.
7. Obtain predictions for each learner on the ith (held-out) block.
8. Store the m sets of predictions for each data point.
9. Fit the meta learner: predict outcome from the m base learner predictions.
10. Store the meta learner weights for each base learner, $w_1, \ldots, w_m$.
11. Fit each base learner to the entire data set.
12. Obtain predictions from the full data set for each learner.
13. Combine the full data predictions using the meta learner weights $w_1, \ldots, w_m$ to produce the final predictions.

Let us step through the details of Algorithm 1. We first choose a set of base learners; we will discuss this choice later in this section and in our applications. We then randomly split our training data into 10 blocks to prepare for cross-validation, labeling the blocks from 1 to 10. The for-loop sequentially holds out each block as an independent set for prediction purposes. At each step in the for-loop, we train each base learner on all blocks except the held-out one. Since every block serves as the held-out block during the training process, at the end of the for-loop, we have a complete dataset of predictions for each of m base

learners; that is, we have *m* sets of predictions for each observation. In line 9, we fit a new learner, the meta learner, to predict the outcome using the set of *m* base learner predictions. A meta learner is an ensemble of the individual learners, the ensemble prediction made by taking a weighted average of the individual learner predictions. A meta learner can be as simple as a linear regression model, the output regressed on each base learner prediction and the weights corresponding to the regression coefficient estimates for each base learner. The final ensemble prediction is made by first fitting each base learner to the entire data set. We then combine these predictions using the weights in line 10 for each base learner as obtained by the meta learner. This ensemble prediction is thus a weighted average of our base learner predictions.

Figure 1 presents a flow chart of the SuperLearner algorithm. The graphic color codes the key components of the algorithm: cross-validation and related prediction (blue-green), prediction from each base learner fit to the entire data set (orange), and link through the meta learner (yellow). For simplicity, the flow chart depicts 10-fold cross-validation and two base learners; the algorithm may be extended beyond those in the obvious way. In the appendix, we present a more mathematically detailed SuperLearner algorithm for the interested reader.

The specific application of the SuperLearner for our show rate prediction is as follows. We first fit each of the *m* base learners to academic and experiential variables available for modeling. We predict a show rate from each base learner—we thus have *m* predicted show rates for each student. We used the default meta learner NNLS, non-negative least squares based on the Lawson–Hanson algorithm (Mullen and van Stokkum 2012) to combine these individual predictions. Choice of meta learner is up to the user.

The choice of base learners comes down to a trade-off among computational expense, predictive accuracy, and user time to code and fine-tune. On the one hand, we can think of learner choice as a combinatorial optimization problem, identifying the best combination of models for predictive performance. In practice, we have found that time is not well spent to identify a best learner set as suboptimal combinations work well, and much better than individual learners. In fact, in the show rate analysis of this paper, we choose only two base learners: random forest (Breiman 2001) and Xgboost (Chen and Guestrin 2016). Our model trained from 2018 and 2019 produced a weight of 0.35 for the random forest learner and 0.65 for the Xgboost learner.

Generalized Simulated Annealing for Admissions Scoring

A challenge in combinatorial optimization problems is that since the number of states is huge and typically the objective function is complex, the surface over which we are optimizing is bumpy. Figure 2, inspired by a graphic in Martinez and Cao (2019), presents an illustrative, though

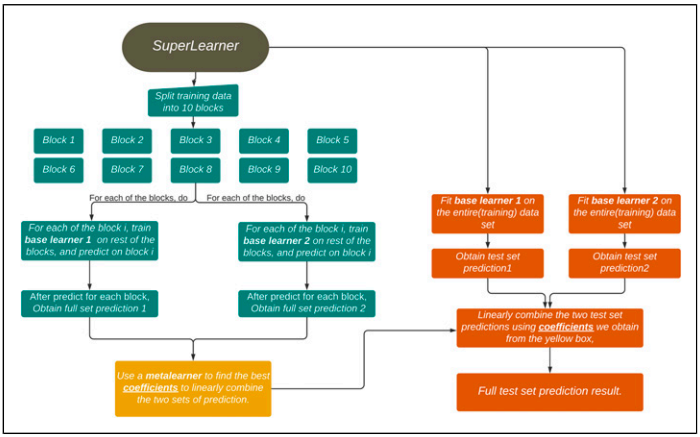

Figure 1. Flowchart for the SuperLearner algorithm.

simplified, graphic of a potential multi-modal objective function for which we may wish to find the minimum. Our goal is to find the so-called global optimum; in our case, the set of bonus points that leads to the best-performing incoming class. However, a bumpy objective function presents potentially many so-called local optima, suboptimal points at which an optimization algorithm can get stuck. Figure 2 presents two local minima (with potentially the left and right edges creating problems for an optimization search routine). We chose to use simulated annealing, an algorithm we have applied with success and identified in the literature as having the ability to reach the global optimum for the function that has multiple local minimums; see Guilmeau et al. (2021) for a review and references to the original work dating back to the 1980s.

The name "simulated annealing" presents two important components of the algorithm for solving combinatorial optimization problems. By "simulated," the algorithm proposes randomly chosen states or steps in exploring the objective function. The algorithm also randomly keeps suboptimal moves. The "annealing" portion of the name comes from an analogy with the annealing process where metals are slowly cooled to remove internal stresses and improve ductility in the manufacturing process. The simulated annealing algorithm parameterizes the objective function with a "temperature," T, decreased (cooled) as the process narrows in on an optimum. In combination, the algorithm is able to make potentially large and potentially suboptimal jumps away from local optima in search of the global optima. After a period of exploration over the objective function (iterations of the algorithm), the temperature cooling limits these large moves allowing the algorithm to zone in on, hopefully, the global optimum.

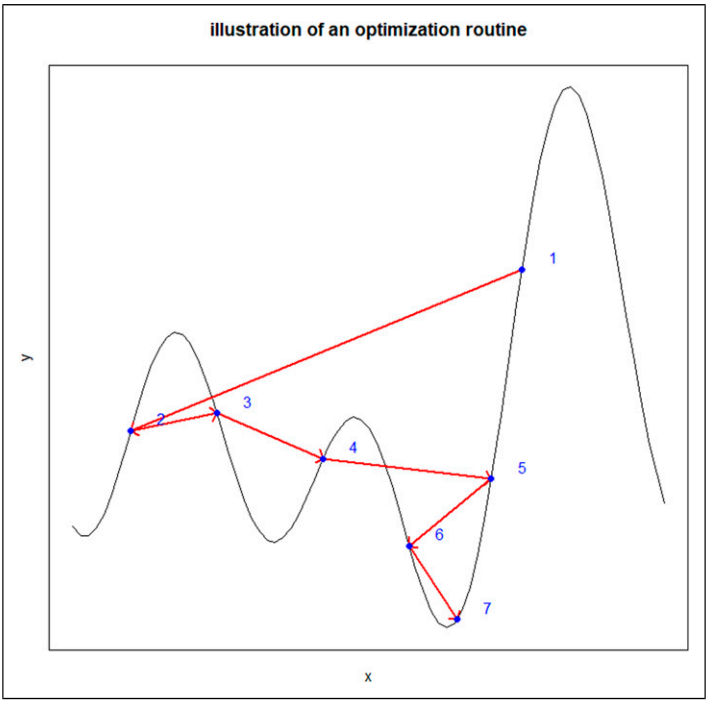

Figure 2. Illustration of an optimization routine.

Figure 2 presents a simple illustration of such an exploration. Note in this figure that the move from steps 1 to 2 presents a large jump between two peaks, followed by a suboptimal move to step 3. This latter move allows the algorithm to escape a potential fall down to a suboptimal region on the left-most side of the function. The move from steps 4 to 5 is again a larger jump between two peaks, again allowing the algorithm to escape another fall to a suboptimal, local trough. Though this simplified illustration presents only seven iterations, the point is that as the temperature cools, the algorithm finds the slope down and progresses to the global minimum, steps 5 to 6 to 7. On more complex surfaces, the algorithm could cool slower and take many more iterations to explore the function (local peaks and valleys). Algorithm 4 presents the specifics of simulated annealing.

Algorithm 2 Simulated Annealing: maximization of objective function $f(x)$

1: Initialize an iteration count $j = 0$
2: Initialize temperature, T^0, and a cooling schedule T over iterations j
3: Initialize the number of trials m over iterations j

4: Initialize a random starting point x^0
5: **while** $T^j>0$ **do**
6: Set temperature as T^j and number of trials as m_j
7: **for** i in 1:m_j **do**
8: Choose a random step size Δ according to a visiting distribution
9: Select a new point $x^i = x^{i-1} + \Delta$
10: Compute the objective function difference $c = f(x^i) - f(x^{i-1})$
11: Compute probability $p = \exp(-c/T^j)$
12: If $c > 0$, then accept the new point x^i;
13: otherwise accept x^i with probability p and x^{i-1} with probability $(1 - p)$.
14: Set current position as $x^0 = x^{m_j}$ and increment $j = j + 1$
15: Output the final point $x^{imin} = x^0$ and its function value $f(x^{imin})$

The simulated annealing algorithm aims to optimize objective function $f(x)$. We must specify a cooling schedule, namely, how the temperature will decrease down to zero as the algorithm explores the objective function space. We denote this schedule as T, the temperature decreasing down to zero as the algorithm steps through iterations counted by j. We must specify a visit count schedule at each temperature. We denote this schedule as m where at temperature T^j at iteration j, we explore m_j moves in search for an optimum. Both T and m are tuning parameters; the user can specify the rate at which the temperature cools and how many moves to make at each temperature depending on how well the algorithm explores the objective function.

Algorithm 4 is presented for maximizing an objective function, $f(x)$, say finding the best predicted graduation rate for an admitted class. The general process of simulated annealing is as follows. We start at a random point, denoted as x^0. We have two loops. The first loop follows iteration j which specifies the temperature as it cools over j (while-loop at step 5) and specifies the number of moves we make each iteration j (for-loop step 7). At a given step i of the algorithm, we have a current position x^{i-1}. We randomly step away from the current position at step 9 according to a step size Δ. If this move is in the right direction, in that the objective function $f(x)$ increases, we keep the new position x^i. If not, we allow for accepting the suboptimal move. We flip a coin with probability p: if heads, we accept the suboptimal move; if tails, we reject the new move and remain at x^{i-1}. At iteration j, we make m_j of these moves. We then start iteration $j + 1$ with a new temperature T^{j+1} at the spot we left off $x^0 = x^{m_j}$, and we study now m_{j+1} moves analogously. We keep doing this over iterations j until the temperature has cooled down to zero.

Note that the temperature impacts the probability of a suboptimal move in that $p = \exp(-c/T^j)$. In particular, for larger temperatures, T^j, we have a higher likelihood of accepting the suboptimal move. We thus are less willing to make these suboptimal moves as the algorithm progresses, figuring we

explored the space well and are approaching the global optimum. The choice of temperature schedule thus is made with this exploration and moves to suboptimal values (downhill moves in the maximization process described here) in mind. Furthermore, for each temperature, T^j, we choose a visitation schedule, m_j, deciding how many moves we would like to study. Since the temperature schedule specification is mathematically involved, we provide details in the appendix.

Each new move is determined by a random step size Δ in step 8. The step size Δ is randomly generated from a pre-specified visiting distribution. We may think of this as a nearest neighbor move, where we randomly jump to a new point x^i in the neighborhood of the current position x^{i-1}. Specification of the neighborhood determines how big a jump we are willing to take across the objective function $f(x)$. There are many variations on the simulated annealing theme. The one we will use for the admissions scoring problem is generalized simulated annealing (GSA) (Xiang et al. 2017). GSA uses a rather flexible visiting distribution and temperature cooling schedule, allowing for a more thorough, yet quick, surface search for the global optimum. As such, GSA can actually converge faster to an optimum than classical simulated annealing. In the appendix, we provide brief mathematical details of both the classical simulated annealing algorithm and the extension to GSA.

Examples of Applications in Admissions Practice

In this section, we present illustrations applying our machine learning methods to estimating the yield rate and scoring university applicants. The examples derive from our collective experiences in enrollment management and institutional research problem-solving. But due to student privacy and confidentiality concerns, we emphasize that these are not the processes put into practice at San Diego State University. Our aim here is to provide the reader realistically based scenarios for demonstration purposes, from which our methods can be implemented for admissions practice in their own university setting.

Show Rate Analysis

The problem we are considering is estimating the probability a student accepted for admission to a university will enroll as a first-time freshman. As mentioned, SuperLearner, as an ensemble averaging method, estimates this probability from a set of student characteristics over a suite of models. Table 1 presents the continuous variables we consider in this analysis. The data are based on student cohorts entering 2018 and 2019 with high school performance variables including grade point average (GPA) overall and in specific subjects as well as the number of courses taken in different course groupings.

We also consider non-academic variables of ethnicity, gender, age, and acceptance into a scholarship program. In this data set, 18% of students are from the local service region, 59% are female, and 18% are first-generation college students.

We use the R-package SuperLearner (E. C. Polley et al. 2011). As mentioned earlier, we use two individual learners in this Super Learner application: random forest and boosting. In practice, one should choose learners by first reviewing all the available learners. Our recommendation is to choose learners with which one is familiar, and fit individual learners first to ensure viable predictions are made, and to tune the parameters for that specific learner. In our application, the learners we experimented with are earth, biglasso, glm, randomForest, and xgboost; each of these learners is written into the SuperLearner R-library. When fitting all the learners of one's choice, the SuperLearner algorithm will shrink the weight of poorly performing learners to zero. In our application, xgboost and random forest are the two best-performing learners with weights greater than zero. Therefore, in the following results, we keep only xgboost and random forest as the base learners.

Each learner has a set of tuning parameters allowing implementation changes to improve predictive accuracy. The practitioner can decide for

Table 1. Summaries of the continuous variables in the show rate analysis.

	Average	Standard Deviation	Maximum
High school GPA	3.94	0.31	4.82
Age	18.47	0.46	39.60
Biology courses GPA	3.86	0.38	5.00
English courses GPA	4.04	0.83	5.00
Foreign language courses GPA	3.77	0.90	5.00
History courses GPA	4.08	0.85	5.00
Physics courses GPA	3.88	0.99	5.00
Visual performing arts courses GPA	3.26	1.53	5.00
Biology courses GPA	3.67	1.86	22.00
CPE course count	4.77	3.63	42.00
English course count	7.89	1.79	32.00
Foreign language course count	5.86	2.21	32.00
History course count	6.45	2.52	44.00
Physics course count	3.78	1.83	22.00
Visual and performing arts course count	4.16	3.46	56.00
Math course count	9.18	3.02	34.00
Math course GPA	3.77	0.83	5.00
STEM course count	17.20	4.88	56.00
College course count	0.41	1.39	23.00

themselves how actively involved they wish to be in tuning parameter selection; SuperLearner will apply the algorithms under default settings or the user can manually try different tuning parameter settings with an eye on predictive performance. For example, random forest has two primary tuning parameters: mtry and maximum number of leaf nodes. mtry specifies the number of features over which to make a random selection within each decision tree node. Maximum number of leaf nodes controls the depth (complexity) of each tree. The default value for mtry in the R package (E. C. Polley et al. 2011) rounds up the square root of the number of variables for classification problems. After tuning the parameters around its default value, we have mtry set to 7 in our applications. SuperLearner itself requires specification of the number of folds in the cross-validation scheme. There is a trade-off: more folds may improve prediction accuracy but at a cost of computational expense. We choose 10-fold cross-validation weighing this trade-off in our applications.

The boosting algorithm we implement is XGBoost (Chen and Guestrin 2016). Boosting entails a sequential process: weak learners are grown using the information from a previously grown weak learner one after the other. This process slowly learns from the data and tries to improve its prediction in subsequent iterations. XGBoost uses a gradient boosting framework. Gradient boosting performs prediction by creating an ensemble of weak prediction models. XGBoost has great flexibility as the user can tune many parameters and create user-defined objective functions. Another difference compared to random forest is that XGBoost incorporates tree pruning, which reduces the redundant size of a tree to increase accuracy.

Boosting has three primary tuning parameters: the number of trees to grow (maximum number of iterations), the learning rate labeled as eta, and regularization controls labeled as alpha and lambda. In our applications, we set alpha and lambda to the default values of zero, that is, we do not perform regularization. For readers interested in this tuning parameter, we provide brief details in the Appendix. The learning rate is the shrinkage performed at each step of the sequential learning procedure. When the learning rate is set to a smaller value, more steps are needed for the training process. Increasing eta speeds up the training process, but impacting prediction accuracy. A small value of eta is usually needed for a good result when training XGBoost models. In addition, a tree booster, like random forest, has tuning parameters related to tree growing. The SuperLearner package (E. Polley et al. 2021) uses the xgboost package (Chen et al. 2021), which sets the default values of the parameters.

Our particular interest is in comparing our ensemble averaged show rate estimate to the show rate computed through a lookup table at San Diego State University, applied to this data. The training data is based on admitted students from 2018 and 2019, the testing data is based on students admitted for 2020. In

the appendix, we present the R code for a run of the SuperLearner for a show rate analysis. Due to confidentiality of student records, we provide code to simulate data mimicking the type of variables and output we see in practice. We hope this structure will allow the user to substitute their own data in place of the simulated data.

Table 2 shows the comparison of show rates of SuperLearner, the traditional show rates, and show rates predicted from each of the base learners (random forest and xgboost) of the SuperLearner model we used. We see that Super Learner performs the best in terms of AUC. For sensitivity and specificity, SuperLearner tends to weigh specificity more. However, as long as we have a better ROC curve (Figure 3), we can always adjust the threshold and trade-offs between sensitivity and specificity. Appendix B reviews ROC curve background concepts for the interested reader.

After the show rate is estimated for each student, we start to make admission decisions. For example, if we have ranked students according to an admissions scoring system, we would sum the show rates down this ranked list until the show rates add up to the targeted enrollment number. We of course could split the population into student subgroups, say, based on majors, or scholarship programs, or athletics. Within each subgroup, we can use the show rate similarly, controlling the number of admitted students in that subgroup. The next critical piece then is a system for scoring and ranking students for admission.

Scoring University Applicants

The problem we are considering is to rank students relative to academic performance but potentially constrained by enrollment targets in student subgroups. We approach this problem by adding points to the admissions score for these subgroups. The challenge then is identifying an appropriate point structure. Our proposed algorithm is modular in that the user can specify how to score academic performance. In this section, we will show three possible options: ranking students via predicted graduation rate, an eligibility index created from high school grade point average (GPA) and SAT score, and

Table 2. Comparison of show rates estimated using SuperLearner and those traditionally used by the university.

	Sensitivity	Specificity	AUC
SuperLearner	0.66	0.78	0.81
Traditional show rate	0.72	0.61	0.72
Random forest	0.68	0.76	0.80
Xgboost	0.71	0.72	0.80

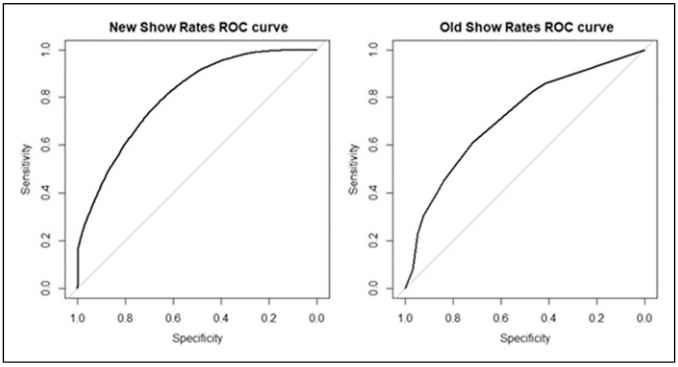

Figure 3. ROC curve for show rates estimated with SuperLearner for the 2018 cohort.

an academic performance score. Again, we emphasize that these are not practices in place at San Diego State University, but illustrations of our methods for a range of admissions processes and decision-making.

Likewise, our proposed algorithm can handle student subgroup enrollment targets, as long as we can represent the constraints in the optimization routine. For illustration purposes, in this section, we are thinking of a school that has the following population targets: incoming class has say 35% of students from the local service region, 20% first-generation college students, and 20% of students having received free- or reduced-price meals in high school (we will be varying these percentages in the applications). These factors represent targets we see in admissions processes, that is, for example, public institutions with a mission to enroll local area students, a university's goal to maintain a diverse student population, and equity efforts in the admissions process with respect to schools in less privileged neighborhoods, respectively. The optimization problem is thus admitting the best class possible with respect to academic performance score, subject to these three targets on experiential factors. Since we are working with an admissions scoring system for ranking students, we include bonus points for local students, first-generation students, and students receiving free- and reduced-price meals in high school. It is over these bonus points that the simulated annealing algorithm is looking to optimize, while maintaining the imposed targets (so in the example here, 35%, 20%, and 20%, respectively).

The final piece is the so-called objective criterion over which we are optimizing. As mentioned, we will consider different outcome measures in each subsection for illustration purposes. In these applications, we consider admissions for a cycle based on Fall 2018, for which we have that year's student rankings and admissions decisions. In practice, the admission process

typically involves a massive amount of manual labor, fine tuning student selection to balance academic performance and the distribution of experiential factors. We thus motivate our algorithms as well as present an automated approach, optimizing academic performance (the objective function value) relative to targets on experiential factors (the objective function parameters). While we do not recommend using the method as a black box, our algorithm produces a student ranking and presents an admission scoring point structure, enabling enrollment management staff the ability to rapidly determine the academic strength of its applicant pool without manually reviewing each application.

From a quality-control perspective, we would like our automated approach to maintain a similar admission selection to this Fall 2018–based data. We thus choose to maximize the rank correlation between the 2018 student performance score without bonus points and the 2018 student performance score with bonus points. That is to say, we use rank correlation to minimize the difference between the new academic performance score with experiential factor targets (bonus points) and the original academic performance score actually used in 2018. More specifically, the Spearman's rank-order correlation is defined as

$$\Gamma = \frac{\sum_{i,j=1}^{n} \left(r_j - r_i\right)\left(s_j - s_i\right)}{\sum_{i,j=1}^{n} \left(r_j - r_i\right)^2}$$

for n students, where r_i and s_i are the ranks of two columns of data: student ranking via our scoring system with bonus points and the actual 2018 student ranking, respectively.

Algorithm 3 Simulated Annealing objective function $f(x)$ for admissions scoring

1. Pass in the bonus points (x) for current iteration of the simulated annealing algorithm.
2. Pass estimated show rates from the SuperLearner algorithm.
3. Calculate the rank correlation of (admissions score) and (admissions score + bonus points).
4. Rank students according to (admissions score + bonus points).
5. Admit students using the show rates to hit target enrollment.
6. Weight the admitted students by show rate and calculate the target feature percentages
7. If the feature percentages meet the set targets, return the rank correlation.
8. If the feature percentages fail to meet the set targets, return 0.

Algorithm 3 presents the basic structure of the objective function we are aiming to optimize in our admissions applications. The objective function is computed for a set of bonus points fed from the simulated annealing algorithm: we call it x here. The admissions score is a number computed from academic performance measures for each student. As the admission scoring depends on the application, we will detail the specific formulation in each of the three application subsections below in this section. The objective function runs an admissions cycle, ranking students on this admissions score with bonus points and selecting admits relative to the estimated show rates. The function then determines if with this set of bonus points creates an incoming class that maintains the enrollment targets (feature percentages; for example, 35% local students and 20% first-generation students). If the targets are satisfied, the function returns the rank correlation to the simulated annealing algorithm; if the targets are not satisfied, this is a failed point structure and is no longer considered in the optimization process (return a value of zero).

We use the R package GenSA (Xiang et al. 2013) to implement the generalized simulated annealing algorithm. The package requires coding the function over which we optimize. The function must, for a given set of bonus points, return a value for the objective criterion, in our case rank correlation. This function is fed as a parameter into GenSA so that the simulated annealing algorithm can scan the space of bonus points for the "optimal" set. Let us summarize the process for the specific application here. For a given set of bonus points, we rank students with respect to our admissions scoring. We estimate the show rates using the SuperLearner algorithm of the *Methods* section. We then admit students by adding up the show rates of the ranked students to achieve a desired target enrollment number. We compute the feature percentages: in this section, percentage local, first-generation and free- and reduced-price meals. These feature percentages are computed by the enrollment-rate–weighted mean over all admitted students; so here, the show rates are utilized again. If all the feature percentages meet the set targets, return the rank correlation between the actual student performance score without bonus points and the student performance score with bonus points; otherwise, return zero (as an indicator that the targets are not met).

Graduation Rate as the Academic Performance Metric. In the application of this subsection, we consider the graduation rate as the academic performance measure for ranking students. For incoming students, we predict graduation rate through a logistic regression model on features available to us (see Table 1). We add bonus points for students in the local service region, first-generation college students, and students receiving free- and reduced-price meals in high school. These bonus points are fit to obtain constraints relative to

these three factors. We consider four target scenarios (see Table 3). We minimize the rank correlation from ranking students based on graduation rate with the bonus points and without bonus points.

Tables 3 and 4 present the optimal bonus points determined relative to the targets in each of the four scenarios. As expected, we see that for higher target percentages, we have larger bonus points. Nonetheless, average high school GPA and SAT scores in these classes are all similar, maintaining academic strength across the scenarios. These results also show great flexibility in the simulated annealing algorithm, as the model is able to identify different bonus points for various targets and various features.

Formulating the Eligibility Index. Many universities have an admissions scoring formula used for ranking students or establishing minimum qualifications for entry. For example, up until Fall 2021 admissions, the California State University system used a so-called eligibility index (EI) being $800 \times HS\ GPA + SAT\ score$. Our simulated annealing algorithm can be used to choose the weights for such scoring formulas over a specified set of inputs.

Here, with EI inputs high school GPA and composite SAT score, we use our proposed simulated annealing to determine the weight for high school GPA. To do this, the academic performance score is a logistic regression model of successful graduation outcome on GPA and SAT. After obtaining the proper weights for high school GPA and SAT score, we calculate the academic performance score by combining high school GPA and SAT score using those weights. The rest of this analysis follows the same as that in the *Graduation Rate as the Academic Performance Metric* section. The optimal EI presents a weight of 350 on high school GPA. Tables 5 and 6 present the bonus point and factor percentages for the three factors of local students, first-generation students, and students receiving free- and reduced-priced meals, analogous to the results presentation of the *Graduation Rate as the Academic Performance Metric* section.

Table 3. Graduation rate: Four scenarios, targets 1–4, of incoming class enrollment targets on percentage of local students, first-generation college students, and students receiving free- and reduced-priced meals in high school, respectively. The table *presents the bonus points* for each factor obtained by the simulated annealing algorithm ranking students on graduation rate and optimizing rank correlation.

	Local	First generation	Free-reduced priced meals
Target 1: 35%, 20%, 20%	5.94	2.84	4.15
Target 2: 40%, 15%, 15%	17.68	0.13	0.00
Target 3: 45%, 20%, 20%	28.70	0.00	0.00
Target 4: 40%, 25%, 20%	18.72	1.71	2.58

Table 4. Graduation rate: Four scenarios, targets 1–4, of incoming class enrollment targets on percentage of local students, first-generation college students, and students receiving free- and reduced-priced meals in high school, respectively. The table *presents the percentages for each category*, average high school grade point average, and average composite SAT score for the incoming class when no bonus points are given, and under the bonus points found by the simulated annealing algorithm and presented for each scenario.

	Without	Target 1	Target 2	Target 3	Target 4
—	Bonus pts	—	—	—	—
% Local	27.1	38.4	41.8	45.3	42.5
% First-generation	15.7	26.9	21.9	26.8	26.6
% Free-reduced priced meals	11.0	20.0	15.4	20.3	20.2
Avg HS GPA	3.85	3.84	3.84	3.83	3.84
Avg pred grad rate	92.3	91.9	91.8	91.7	91.3
Avg SAT score	1266	1239	1244	1231	1235

Academic Performance Scoring

In this subsection, we illustrate an admissions process ranking students on an academic performance score involving more features than the eligibility index of the previous subsection. Similar to the last subsection, we fit a model on graduation rate to predict the eligibility index. The difference is that we incorporate other academic and student background features listed in Table 1, not just high school GPA and SAT score. The rest of this analysis follows the same procedure as the last subsection.

In our application pool based on 2018 admissions data, if we use the actual student performance score, we would have enrolled 27.1 percent local students, 15.7 percent first-generation students, and 11 percent of students having free- reduced price meals in high school. The first targets we set are 35%, 20%, and 20% (Target 1), respectively. Table 7 presents the bonus points for each feature. These bonus points provided the best rank correlation of our new ranking with the actual one used in 2018, but increases the features by 11.3 percentage points for local students, 11.2 percentage points for first-generation students, and 9 percentage points for students receiving free-and-reduced price meals in high school; the average high school GPA reduced by only 0.01 points. See Table 8 for these summaries. We also experiment with other targets, analogous set-up details and results shown in Tables 7 and 8. In each of these cases, the simulated annealing algorithm identifies a bonus point structure that achieves the targets while maintaining incoming class academic performance.

Table 5. A New Eligibility Index: Using the simulated annealing algorithm to identify an academic performance score: graduate rate = SAT score + c × HS GPA; c is found to be 350. The table then presents bonus points under four scenarios, targets 1–4, of incoming class enrollment targets on percentage of local students, first-generation college students, and students receiving free-and-reduce priced meals in high school, respectively.

	Local	First generation	Free-reduced priced meals
Target 1: 35%, 20%, 20%	178.93	6.21	215.54
Target 2: 40%, 15%, 15%	207.06	21.94	100.87
Target 3: 45%, 20%, 20%	305.35	0.31	187.61
Target 4: 40%, 25%, 20%	235.22	109.84	157.09

Table 6. A New Eligibility Index: Percentages for each of three factor categories, average high school grade point average, and average composite SAT score for the incoming class when no bonus points are given, academic performance is determined by the formula SAT + 350 × HS GPA found by the simulated annealing algorithm, and academic performance is determined by the original CSU EI SAT + 800 × HS GPA.

	Without	Target1 new EI	Target1 old EI
—	Bonus pts	—	—
% Local	24.3	37.1	38.4
% First-generation	13.6	20.7	26.9
% Free-reduced priced meals	9.3	15.0	20.0
Avg HS GPA	3.82	3.83	3.84
Avg pred grad rate	92.5	91.4	91.3
Avg SAT score	1288	1270	1240

Table 7. Academic Performance Score: Four scenarios, targets 1–4, of incoming class enrollment targets on percentage of local students, first-generation college students, and students receiving free- and reduced-priced meals in high school, respectively. The table presents the bonus points for each factor obtained by the simulated annealing algorithm ranking students on an academic performance score and optimizing rank correlation.

	Local	First generation	Free-reduced priced meals
Target 1: 35%, 20%, 20%	183.67	189.17	248.58
Target 2: 40%, 15%, 15%	293.90	60.76	87.03
Target 3: 45%, 20%, 20%	343.83	157.45	252.56
Target 4: 40%, 25%, 20%	283.69	154.89	254.29

Discussion

In this paper, we present a combinatorial optimization framework via simulated annealing for scoring university applicants. As part of the scoring system, and to select the ranked students for admissions, we introduce an ensemble learning scheme through the SuperLearner algorithm to predict the yield or show rate for admitted students.

The algorithms we developed are modular in nature. Enrollment management units may select their desired student characteristics for inputs into the admissions process. These inputs may span the gamut of academic performance metrics, school district–specific information, neighborhood or regional factors, student participation in programs (e.g., volunteer hours), and student demographics. Universities also collect text data on applicants, for example, student personal statements and letters of recommendation. These texts may be scored by readers or processed through text mining tools, the quantifications being additional inputs. Likewise, model constraints may be formulated by enrollment management units based on enrollment targets and admission programs. Metrics of student success must also be determined as output over which the scoring system is optimized. Of note, there is literature on computing student performance measures that may be used as output, for example, see Mengash (2020). As use of such data for admissions decisions are often mandated by university systems and state governments, this paper focuses specifically on the methods and algorithms, leaving choice of student attributes (inputs), enrollment targets (model constraints), and performance metrics (outputs) as pre-specified parameters entered into the optimization framework.

Table 8. Academic Performance Score: Four scenarios, targets 1–4, of incoming class enrollment targets on percentage of local students, first-generation college students, and students receiving free- and reduced-price meals in high school, respectively. The table presents the percentages for each category, average high school grade point average, and average composite SAT score for the incoming class when no bonus points are given, and under the bonus points found by the simulated annealing algorithm and presented for each scenario in the table.

	Without	Target 1	Target 2	Target 3	Target 4
—	Bonus pts	—	—	—	—
% Local	27.1	38.4	41.8	45.3	42.5
% First-generation	15.7	26.9	21.9	26.8	26.6
% Free-reduced priced meals	11.0	20.0	15.4	20.3	20.2
Avg HS GPA	3.85	3.84	3.84	3.83	3.84
Avg SAT score	1266	1240	1245	1231	1235

The algorithms developed require setting so-called tuning parameters. In the simulated annealing algorithm, the user specifies a temperature schedule directing the random jumps around the space of points for student scoring. The user also may specify the number of proposal point combinations explored at each temperature. We use default settings within the generalized simulated annealing R package applied. Nonetheless, the user has the flexibility to set these tuning parameters to achieve a desired convergence, in terms of speed and accuracy, to the optimum. In the SuperLearner algorithm, the user specifies the suite of base learners over which the ensemble prediction is made, the meta learner used to combine the base learner predictions into an ensemble prediction, and potentially tuning parameters for each base learner. The trade-off to consider is user and computational time, to select the options for and then run SuperLearner, and predictive accuracy. In our applications, we did not find the gain in prediction accuracy to warrant time spent on fine tuning the algorithm. We thus recommend a simple SuperLearner application with a handful of base learners at most, and default settings for the base learners. Nonetheless, the user has the flexibility to make these decisions toward improving prediction of show rates.

Perhaps of utmost importance in applying our methods is data curation and data quality control. As specific examples, during the course of our enrollment applications, we realized a potential target leaking problem. Data from a student's high school senior year, or more typically latter half of the senior year, is not available for admissions decisions made in winter or early spring. However, historical records of students already attending the university may be overwritten with student high school transcript updates the summer before they enter the university. Enrolled students thus present a different set of data, which in turn leaks information about their enrollment choice. Training data and testing data are thus not consistent leading to erroneous predictions of show rate and/or over-confidence in the accuracy of show rate predictions. Policy changes, at the school district, government, and/or university level may also lead to unexpected changes in historical data and potential challenges in training and testing phases. On another front, demographics may not be collected at the student-level. For instance, one may have access only to the percentage of students at a high school that receive free- and reduced-price meals. All students from a given high school share the same data point on this variable. And often in practice, a number of student demographic variables are self-reported, lending to potential biases and inaccuracies. Of course, as universities perform deeper analyses for data-informed decision-making, data pipelines will be developed for automated, accurate data processing at a fine-grain level. That said, these data issues should always be kept in mind as data is curated for analyses.

A reviewer queried how the complexity of different inputs, in combination with the complexity of different enrollment targets, may impact the quality of

the output in our approach. Colleges, universities, and university systems are hiring data science experts for their administrative teams. In fact, demand for such analytics expertise is being seen broadly across industry, not limited to higher education institutions. These individuals can take the framework and R code we posit and directly adapt it to scoring and yield rate models in an admissions cycle. A colleague made an analogy with sabermetrics, how baseball teams are increasingly using predictive models and analytics to make personnel and on-field strategy decisions. A similar data-informed decision-making "revolution" is occurring in higher education. Nonetheless, the question remains on how much technical effort is needed for a given ranking and selection problem. On the one hand, with the ever-increasing quantity and breadth of data collected on students, and evolving university enrollment goals, machine learning scoring models and combinatorial optimization routines as we put forth at the least provide a first pass on assessing features to predict student success. On the other hand, as the reviewer correctly suggests, future research needs to consider the ramifications of complex system inputs and constraints, and compare performance gain relative to the implementation effort.

Appendix A

Simulation Experiment: Evaluating the Efficacy of the Proposed Simulated Annealing Approach for Admissions Scoring

In this section, we wish to evaluate the performance of the simulated annealing algorithm we propose for solving the constrained combinatorial optimization problem presented by the admissions scoring problem. For this purpose, we set up a simulation experiment where we simulate data knowing the points assigned for student attributes. We then study whether an application of our approach to this simulated data successfully identifies these true point structures. We note the potential confusion in two uses of the word "simulation" here. This section is a simulation experiment where we are generating data under a known process. In this way, we know what the solution to the constrained optimization problem should be and thus can determine if our proposed methods work correctly. Simulated annealing is the name of the combinatorial optimization algorithm we use to solve this admissions scoring problem. The simulated annealing algorithm includes a step where randomly sized jumps are made around the objective function, as it searches for the optima. We are also sensitive to the concern that investigators often have different population features they want to balance out, so we experiment with varying population features and compare the results.

For the data-generating process, we simulated 1000 students and examined different population features. For the first scenario, we examined two population features: local status and first-generation college student status. We assume that in the population of applicants, 17% are local and 22% are first-generation college students. Our goal is to enroll 40% local and 30% first-generation students for the incoming class. Without loss of generality and for simplicity sake, we assume the show rates of all applicants are the same and equal to 0.25.

In the first simulation experiment, we set the true bonus point for a local student to be 15 and for a first-generation college student to be 5. The data generation process is as follows:

1. Generate student admission scores (including bonus points) uniformly from 0 to 100.
2. Select the final admission pool as students with admission score ≥ 80; record the number of admitted students to be the target admission number.
3. Randomly select students to be local (probability, 0.1) and first-generation college students (probability, 0.2) for students that are not in the final admission pool. Randomly choose students to be local (probability, 0.4) and first-generation college students (probability, 0.3) for students that are in the final admission pool.
4. Remove bonus points from the student admission scores (subtract 15 points from local students and 5 points for first-generation college students).
5. Set show rates for all students to be 0.25.
6. Run the simulated annealing process to identify appropriate bonus points for each of the two features.
7. Repeat Steps 1–6 one hundred times

Step 4 creates a data set without bonus points to feed into the simulated annealing algorithm. Our proposed method, if working correctly, should from this data set successfully identify the true point structure of 15 points for local students and 5 points for first-generation college students, and recover the true admission pool from Step 2. We initialize the simulated annealing algorithm with bonus points of 20. There will be variation across simulated data sets so we run the simulation experiment 100 times; that is, generate 100 data sets of 1000 students each. Figure 4 presents a distribution of the admission score and the two features for the simulated data set for scenario 1. Figure 5 presents the results over these data set replications, showing that our proposed method captures the true bonus points. The average of the local student bonus point is 15.3, and the average of the first-generation college student bonus point is 5.8, across the 100 simulated data sets. Looking at the variation over the simulated

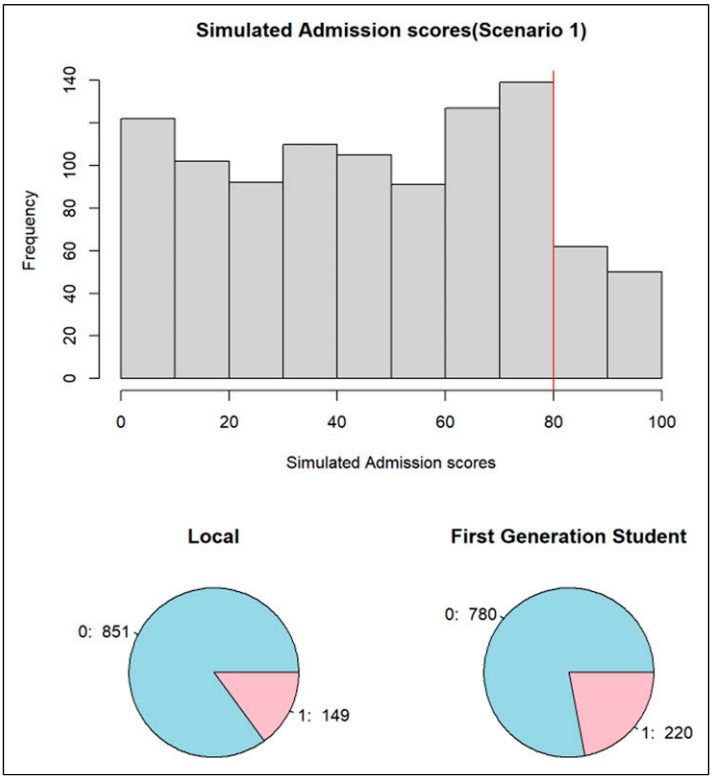

Figure 4. Simulation data illustration for scenario 1. The admissions scores are the initial scores without the bonus points. The red line identifies the admission cut-off score of 80.

data sets, the average bias is within 1 point and the standard deviation is within 1 point. We note that the smaller the population feature difference between the target percentage and the true population percentage, the less accurate the bonus point optimization. In this simulation, we set an admissions pool target of 30% first-generation college students while the population has 20% generation; this could lead the simulation annealing algorithm to be less accurate.

In the second scenario, we have three features: local students, first-generation college students, and free-and-reduced lunch indicator. The actual bonus points are set to be 15 for local students, 5 for first-generation college students, and 10 for students receiving free-and-reduced lunch. Figure 6 presents a distribution of the admission score and the three features for the simulated data set for scenario 2. Figure 5 shows the results of this experiment

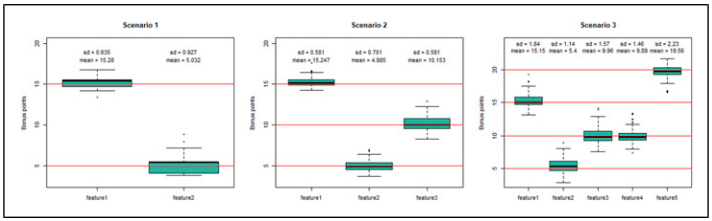

Figure 5. Simulation experiment results for assessing performance of the simulated annealing algorithm in scenarios one, two, and three, from left to right. Each graphic presents a box plot of the bonus points identified by the simulated annealing algorithm over 100 simulated data sets of 1000 students each. The black line in each box is the median, and the average and standard deviation of the optimal bonus points found across the simulated data sets is presented above the box plot. In each graphic, the true bonus point for each feature is presented by the horizontal red line. For scenario one, the true bonus points are 15 for local students and five for first-generation college students; for scenario two, the true bonus points are 15 for local students, five for first-generation college students, and 10 for students receiving free-and-reduced lunch; for scenario three, the true bonus points are 15, 5, 10, 10, and 20 for features one through five, respectively.

across 100 simulated data sets. The average bonus points across the data set replications is 16.4 for local students, 6.9 for first-generation college students, and 11.6 for students receiving free-and-reduced lunch; the average bias is within 2 points and the standard deviation is within 2 points. We note that adding more population features may lead to variations in the simulated annealing process making the simulated annealing algorithm less accurate.

In the third scenario, we include five features to examine the admission structure we proposed. As there are more bonus points, we need to expand the range of the admission score. The data generation process is as follows:

1. Generate student admission scores (including bonus points) uniformly from 75 to 200.
2. Select the final admission pool as students with admission score ≥ 140; record the number of admitted students to be the target admission number.
3. Randomly select student characteristics relative to each of the five features: for students that are not in the final admission pool use the probabilities 0.1, 0.2, 0.1, 0.1, and 0.1, respectively; for students that are in the final admission pool use the probabilities 0.4, 0.3, 0.2, 0.25, and 0.3, respectively.
4. Remove bonus points from the student admission scores (subtract 15, 5, 10, 10, and 20 for each of the five features, respectively).
5. Set show rates for all students to be 0.25.

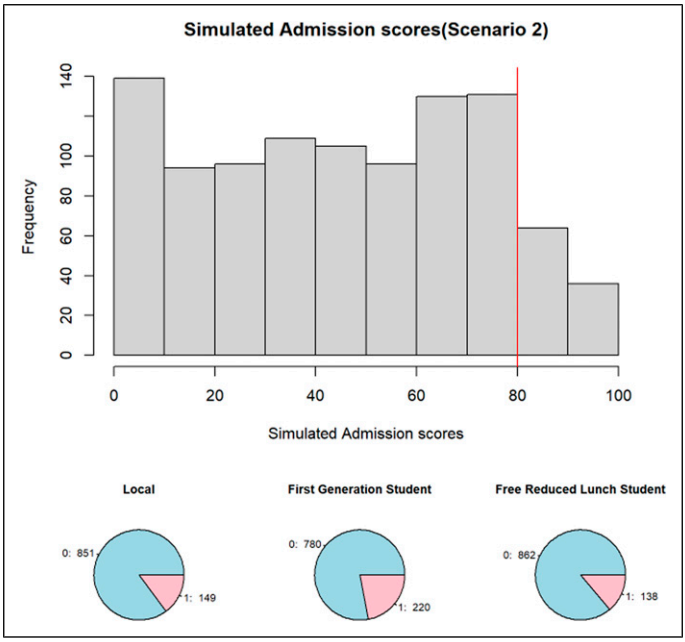

Figure 6. Simulation data illustration for scenario 2. The admissions scores are the initial scores without the bonus points. The red line identifies the admission cut-off score of 80.

6. Run the simulated annealing process to identify appropriate bonus points for each of the five features.
7. Repeat Steps 1–6 one hundred times.

Figure 7 presents a distribution of the admission score and the five features for the simulated data set for scenario 3. Figure 5 again presents the results from the experiment over 100 simulated data sets of 1000 students each. The graphic shows that the simulated annealing algorithm captures the correct bonus points on average, with small standard deviations on each.

Appendix B

Machine Learning Background

In the *Methods* section, we elude to a number of concepts and techniques currently common in analytics work-ups: machine learning models, classification and regression problems, training/testing, and cross-validation.

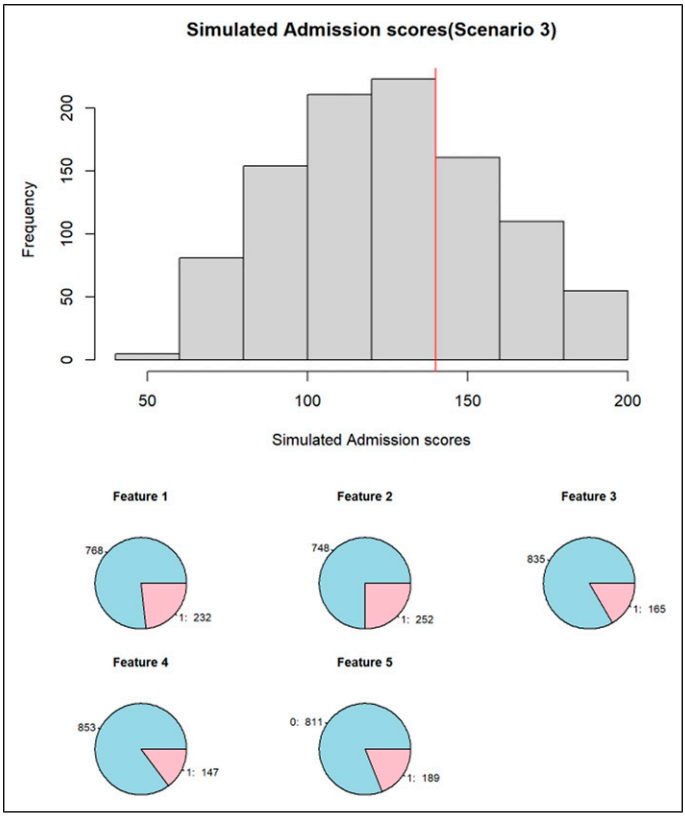

Figure 7. Simulation data illustration for scenario 3. The admissions scores are the initial scores without the bonus points. The red line identifies the admission cut-off score of 140.

Within these concepts are ideas of supervised and unsupervised learning, overfitting and underfitting, input data, output data, out-of-bag sample, and ROC curve. In this Appendix section, we provide machine learning background for the interested reader.

Machine learning models include supervised learning and unsupervised learning. A supervised learning algorithm teaches a computer to make predictions for future observations based on historical data (training data). A model is just a mathematical function fit to data using algorithms. Algorithms are predefined steps that take data (in our case, student characteristics and academic performance) as input and then transform it into outputs (predictions) through mathematical operations. There are an overwhelmingly large number of machine learning methods/algorithms available to the practitioner;

for example, the statistical learning textbook by James et al. (2021) presents multiple approaches, which in itself is just touching the surface. That said, each algorithm will have distinct advantages and disadvantages, in processing inputs into predictions.

Classification problems entail classifying subjects into a binary 0/1 outcome or into categories. Regression problems entail predicting a continuous outcome. Predictive performance in either problem setting typically weighs challenges in overfitting and underfitting the input data. Overfitting occurs when a model does not successfully pull signal from noise, following input details for the given training data set too closely. On the other end, underfitting occurs when a model does not learn enough of the pattern in the training data. In each case, the model will poorly predict output for an external, independent, or new set of inputs, the overfit model capturing too much training data noise, the underfit model unsuccessfully capturing the training data signal.

Evaluation of a learner typically involves a training and testing process. The goal is to build or train our model on historical data where the true outputs are known and used in the model fitting; this is the training data set. We then evaluate the model fit on an independent testing data set. We also need to know the outputs in the training data set to assess model predictions against the truth; this is called model validation. The testing set is thus thought of as a targeted future prediction set. However, the testing set is always assumed to be unknown during the learning process so we can mimic the real-world problem as much as possible when evaluating the predictive performance. Ideally, we have two independent data sets to work with for training and testing. For example, in our analysis, our training set is based on historical student information from 2018 and 2019, and our testing set includes all the students from 2020 for which we want to predict the show rate. The testing set error rate is typically more reliable than the training set error rate.

In practice, we often have a single data set and thus create training and testing sets by splitting the data. We can randomly split the data set or use so-called cross-validation. The SuperLearner algorithm uses cross-validation. Generally speaking, K-fold cross-validation first divides the data into K groups and then takes one group as the testing data set and the other $K - 1$ groups as the training data. We often call the small group of $1/K$ of the data as the "out-of-bag (held-out) sample" as it is removed from and independent of the learning/training process. The cross-validation routine then loops through each of the K groups in turn as testing data. Cross-validation is potentially computationally intensive as we ultimately train K models. However, the routine is easily run in parallel over the K groups, and at the end, we have predictions, independent of the learning process, for each observation.

We assess the performance of the SuperLearner fit using ROC curves and area under the ROC curve (AUC). In classification problems, the prediction model must classify a success or failure (1/0 outcome), in our case, whether a

student shows or not. This classification is typically done by setting a probability cut-off; for example, if the probability of showing is greater than 0.5, we declare that the student will show (outcome of 1). However, a 50–50 cut-off may not be the best for predictive accuracy. To assess the impact of the cut-off, we present an ROC curve, which plots the sensitivity (probability of correctly classifying students as showing) and specificity (probability of correctly classifying a student as not showing) over show probability cut-offs from zero to one. The area under the ROC curve (AUC) quantifies predictive performance: the larger the area, the closer we are to perfect sensitivity and specificity and thus the better the prediction. A random guess (coin flip) has an AUC of 0.5 and a perfect prediction has an AUC of 1.

Appendix C

Mathematical Description of the SuperLearner Algorithm

Denote the learning data set $X_i = (Y_i, W_i)$, where Y_i is the outcome variable and W_i the p-dimensional set of covariates. Denote L the library of algorithms {L1, L2, …, $LK_{(n)}$} Denote the predicted response $\widehat{\Psi}_k(W)$ where W is the set of p-dimensional covariates.

Algorithm 4 SuperLearner

1. **procedure** SuperLearner (L, X, W)
2. Fit each $L_k \in L$ on X to estimate $\widehat{\Psi}_k(W)$.
3. Perform a V-fold cross-validation, denote the T(v) to be the v^{th} training data split and V(v) the v^{th} validation data split.
4. For v^{th} fold, fit each algorithm on training data T(v) and save the prediction on V(v), denote the expected loss fitted on T(v) and calculated on V(v) $\widehat{\Psi}_{k,T(v)}(W_{V(v)})$ where $W_{V(v)} = W_i : X_i \varepsilon \mathrm{V(v)}$.
5. Create a V by K matrix Z= $\{\widehat{\Psi}_{k,T(v)}(W_{V(v)}), v = 1,\ldots V, k = 1,,,K\}$
6. let $m(z|a) = \sum_{k=1}^{K} \alpha_k \widehat{\Psi}_{k,T(v)}(W_{V(v)}), \alpha_k \geq 0, \sum_{k=1}^{K} \alpha_k = 1$ where α is a k dimensional weight–vector
7. Solve $\widehat{\alpha} = \mathrm{arg}min_{\alpha} \sum_{i=1}^{n} (Y_i - m(z_i|\alpha))^2$
8. Return the final prediction vector $\alpha_k \widehat{\Psi}_k(W)$

Appendix D

Mathematical Description of Simulated Annealing for Combinatorial Optimization

Let temperature function at iteration j to be $T^j = \alpha(T^j)$, where α is a decreasing function to zero. Denote $m_j = \beta(m_{j-1})$ the number of iterations at each temperature, it should be large and increasing in j. Denote the learning data set $X_i = (Y_i, W_i)$, where Y_i is the outcome variable and W_i the p-dimensional set of covariates. Denote the predicted response $f(x)$.

In classical simulated annealing (SA), the visiting distribution is a Gaussian distribution (a local search distribution) for each temperature. It has been observed that this distribution is not optimal for moving across the entire search space. Generalized simulated annealing (GSA) was developed to overcome this issue by using a distorted Cauchy–Lorentz visiting distribution (Xiang et al. 2017), with its shape controlled by the parameter q_v

$$Gq_v(\Delta x(t)) \propto \frac{\left[T_{q_v}(t)\right]^{-\frac{D}{3-q_v}}}{\left[1+(q_v-1)\frac{(\Delta x(t))^2}{\left[T_{q_v}(t)\right]^{\frac{2}{3-q_v}}}\right]^{\frac{1}{q_v-1}+\frac{D-1}{2}}}.$$

A generalized metropolis algorithm is used for the acceptance probability

$$p_{q_a} = \min\left\{1, [1-(1-q_a)\beta\Delta E]^{\frac{1}{1-q_a}}\right\}.$$

The temperature $T_{q_v}(t)$ is decreased according to

$$T_{q_v}(t) = T_{q_v}(1)\frac{2^{q_v-1}-1}{(1+t)^{q_v-1}-1}.$$

When $q_v = 1$ and $q_a = 1$, GSA resorts to the classical simulated annealing. When $q_v > 2$, GSA converges faster than SA and is also able to escape from a local optimum more easily than SA. In the R package GenSA, the default value of q_v and q_a are set to 2.62 and -5 (Xiang et al. 2013).

Appendix E

A Little More on XGBoost

As mentioned in the *Examples of Applications in Admissions Practice* section, XGBoost is a very flexible boosting algorithm. One of the advantages of XGBoost, compared to random forest, is regularization, a technique used to avoid overfitting. Regularization is turned through parameters alpha and lambda. These regularization terms help control overfitting. Alpha controls the L1 regularization, and lambda controls the L2 regularization. In other words, alpha tries to estimate the median for regularization, and lambda tries to estimate the mean for regularization. Another difference is that alpha eliminates unwanted features, but lambda only shrinks the importance of that unwanted feature to a very small coefficient. If one is interested in tuning these parameters, the process is usually as follows. The range of alpha and lambda is zero to infinity. When tuning alpha and lambda, always start with 0 and add up. A value of 20 is considered an extremely high value of the regularization constant. The idea is that when the training and testing cross-validated prediction results are very similar, the model may be overfit and the values of alpha or lambda should be cut down, say by 10% or 20%.

If a user is experiencing poor performance, the overall tuning parameter strategy usually follows the following structure. We first choose a small eta of 0.1, and then use cross-validation to select the optimal number of trees to grow while keeping all other parameters as default. We then perform a grid search on the remaining booster parameters. After obtaining the best booster parameters over this grid search, we can tune the regularization constants (alpha and lambda). At last, we adjust eta, the learning rate, to guarantee the efficiency of our model. As stated in the *Examples of Applications in Admissions Practice* section, a lower learning rate improves prediction accuracy at the cost of greater computational expense. A suggested strategy is to choose eta as small as possible during training. Complete details can be found at (Saraswat n.d.).

Appendix F

R Code

Listing 1 SuperLearner Sample Code

```
##################
  # Setting up an example of data to feed into the model.
  # All variables simulated, # but we give specific variable names
```

```
# we may see in practice for illustration.

# Make sure to install packages before
# calling individual learners.
# Libraries used:
# SuperLearner
# ranger
# xgboost
# pROC
#################

gender=rbinom(n=1000,prob=0.4,size=1)
local=rbinom(n=1000,prob=0.1,size=1)
first_gen=rbinom(n=1000,prob=0.1,size=1)

admit=rbinom(n=1000,prob=0.2,size=1)
APS=rnorm(n=1000,mean = 3000, sd=500)
APS[APS>4000]=4000
Enroll=rep (0,1000)
Enroll[admit==1]=rbinom(n=sum(admit==1),prob=0.5,size=1)

dat=as.data.frame(
        cbind(gender,local,first_gen,admit,Enroll,APS))
train=dat [1:500,]
test=dat [501:1000,]
preds=c("gender", "local")
y <- as.numeric(train[,"Enroll"])

ytest <- as.numeric(test[,"Enroll"])

x <- subset(train,select=preds, drop = FALSE).
xtest <- subset(test, select=preds, drop = FALSE)

#check and see what individual learners are available
listWrappers ()
library("SuperLearner")
library("ranger")
library("xgboost")

# Call to SuperLearner
# x: covariates
# y: outcome (for our application,
# whether a student accepts admission)
# Fit model with training set
m=SuperLearner(y, x, family=binomial(),
        SL.library=list("SL.ranger", "SL.xgboost"))
```

```
# Make predictions for the test set
predictions <− predict.SuperLearner (m,
        newdata=xtest, onlySL = TRUE)

# Store predictions to compute the show rate
test$show_rate=predictions$pred

#roc, auc, confusion matrix codes.

library(pROC)
myroc=roc(predictor=(predictions$pred), response=(test$Enroll),auc= TRUE)
plot(myroc,main=" New Show Rates ROC curve")
myroc$auc

# Confusion matrix and AUC ROC analysis
best_thresh=coords(myroc, "best", "threshold",
        transpose = TRUE)[1]
cm < − confusionMatrix(as.factor (
        ifelse(predictions$pred>=best_thresh,1,0)),
cm
```

Listing 2 Simulated Annealing Sample Code

```
##################
# Setting up an example of data to feed into the model.
# All variables simulated, # but we provide labels to coincide with
# variables we may see in practice for illustration.

# Make sure to install needed packages
# Libraries used: GenSA
#################

par1=c(200,200,200)#starting point
lower=c(0,0,0)#bonus point lower bound.
upper=c(200,200,200)#bonus point upper bound

summary(train$APS) #academic performance score
test$fake_admit=0 #the variable to store admission decision.
targets=200 #suppose we want to admit 200 people.
feature_targets=c(0.3,0.3,0.3)
x=c(1,1,1)
feature_vectors=test[,c("gender","local","first_gen")].

Minimize_rank_difference= function(x) {
  test$fake_admit=0

  #new academic performance score
```

```
    y=test$APS+unlist(rowSums(feature_vectors*x))
    test$fake_admit=0
    n=nrow(test)
# compute error rate
    error=−abs(cor.test (y, test$APS,
          method = "spearman")$estimate)
       i=1
       while(sum(test$showrate [test$fake_admit==1]) < targets){.
          test$fake_admit [order(y, decreasing = TRUE)][i]=1
          i=i+1
          }


perc_local=sum(test$fake_admit [test$local==1] *
                          test$show_rate[test$local==1]) /.
                          sum(test$fake_admitast test$show_rate)
perc_first=sum(test$fake_admit[test$first_gen==1] *
                          test$show_rate[first_gen==1]) /
                          sum(test$fake_admit*test$show_rate)
perc_gender=sum(test$fake_admit [test$gender==1] ast
                          test$show_rate[test$gender==1]) /
                          sum(test$fake_admit*test$show_rate)
#average_GPA=(sum(train$Hsgpa [train$fake_admit==1] ast
# train$showrate.y[train$fake_admit==1]) +
# sum(ott [ott$Period==20184 & ott$Enroll==1, ]$local)) /
# (sum(targets)+
# nrow(ott[ott$Period==20,184 & ott$Enroll==1, ]))

#use show rate to compute
#the percentage for enrolled student
#weighted proportion

if(perc_local<feature_targets [1] vert
       perc_first<feature_targets[2] vert
       perc_gender<feature_targets[3]){
    error=0
}

error

}

tol <− 1e−13
```

```
library (GenSA)
out <– GenSA(lower = lower, upper = upper,
                    fn = Minimize_rank_difference,par=par1,
                    control=list(maxit=10, verbose=TRUE))
out [c("value","par","counts")]
```

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the NSF grant 1633130.

References

Aksenova, S., Zhang, D., & Lu, M. (2006). "Enrollment Prediction through Data Mining." In 2006 IEEE International Conference on Information Reuse & Integration, Waikoloa, HI, USA, 16–18 Sept. 2006, 510–515. https://doi.org/10.1109/IRI.2006.252466.

Breiman, L. (2001). "Random Forests." *Machine Learning*, *45*, 5–32. https://doi.org/10.1023/A:1010933404324.

Bruggink, T. H., & Gambhir, V. (1996). "Statistical models for college admission and enrollment: A case study for a selective liberal arts college." *Research in Higher Education*, *37*(2), 221–240. http://www.jstor.org/stable/40196173.

Chen, T., & Guestrin, C. (2016). "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, New York, NY, USA, August 2016, 785–794, Association for Computing Machinery. https://doi.org/10.1145/2939672.2939785.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., & Li, Y. (2021). *Xgboost: Extreme Gradient Boosting [Computer Software Manual]*. Retrieved from https://CRAN.R-project.org/package=xgboost (R package version 1.4.1.1.

Dreo, J., Siarry, P., Petrowski, A., & Taillard, E. (2006). *Metaheuristics For Hard Optimization*. 1st ed. Springer.

Erana-Diaz, M. L., Cruz-Chavez, M. A., Rivera-Lopez, R., Martinez-Bahena, B., Avila-Melgar, E. Y., & Heriberto Cruz-Rosales, M. (2020). "Optimization For Risk Decision-Making Through Simulated Annealing." *IEEE Access*, *8*, 117063–117079. https://doi.org/10.1109/ACCESS.2020.3005084.

Ghorbani, A., & Akbari Jokar, M. R. (2016). "A hybrid imperialist competitive-simulated annealing algorithm for a multisource multi-product location-routing-inventory problem." *Computers & Industrial Engineering*, *101*, 116–127.

Guilmeau, G., Chouzenoux, E., & Elvira, V. (2021). "Simulated Annealing: A Review And A New Scheme." In SSP 2021 - IEEE Statistical Signal Processing Workshop, Brazil, 11–14 July 2021.

Haddou Benderbal, H., Dahane, M., & Benyoucef, L. (2018). "Modularity Assessment In Reconfigurable Manufacturing System (Rms) Design: An Archived Multi-Objective Simulated Annealing-Based Approach. *The International Journal of Advanced Manufacturing Technology*, *94*, 729–749.

Haris, N. A., Abdullah, M., Hasim, N., & and Rahman, F. A. (2016). "A Study On Students Enrollment Prediction Using Data Mining. In: Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, Da Nang Vietnam, 4–6 January, 2016, 1–5. https://doi.org/10.1145/2857546.2857592.

Hossler, D., & Bontrager, B. (2015). *Handbook Of Strategic Enrollment Management*. Jossey-Bass.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction To Statistical Learning*. 2nd ed. Springer.

Mamano, N., & Hayes, W. B. (2017). Sana: Simulated Annealing Far Outperforms Many Other Search Algorithms For Biological Network Alignment. *Bioinformatics*, *33*, 2156–2164.

Martínez, C. M., & Cao, D. (2019). Integrated Energy Management For Electrified Vehicles. In *Ihorizon-enabled energy management for electrified vehicles*, edited by C. M. Martinez, & D. Cao. Butterworth-Heinemann. 15–75. Retrieved from https://www.sciencedirect.com/science/article/pii/B9780128150108000028 doihttps://doi.org/10.1016/B978-0-12-815010-8.00002-8.

Mengash, H. A. (2020). Using Data Mining Techniques To Predict Student Performance To Support Decision Making In University Admission Systems. *IEEE Access*, *8*, 55462–55470. https://doi.org/10.1109/ACCESS.2020.2981905.

Moon, H., Ahn, H., Kodell, R. L., Baek, S., Lin, C.-J., & Chen, J. J. (2007). Ensemble Methods For Classification of Patients For Personalized Medicine With High-Dimensional Data. *Artificial Intelligence in Medicine*, *41*, 197–207.

Mullen, K. M., & van Stokkum, I. H. M. (2012). *nnls: The lawson-hanson algorithm for non-negative least squares (nnls) [Computer software manual]*. Retrieved from https://CRAN.R-project.org/package=nnls (R package version 1.4).

Piryonesi, S. M., Nasseri, M., & Ramezani, A. (2019). Resource Leveling In Construction Projects With Activity Splitting And Resource Constraints: A Simulated Annealing Optimization. *Canadian Journal of Civil Engineering*, *46*, 81–86. https://doi.org/10.1139/cjce-2017-0670.

Polley, E., LeDell, E., Kennedy, C., & van der Laan, M. (2021). *Superlearner: Super Learner Prediction [Computer Software Manual]*. Retrieved from https://CRAN.R-project.org/package=SuperLearner (R package version 2.0-28).

Polley, E. C., Rose, S., & van der Laan, M. J. (2011). Super learning. In *Targeted learning: Causal inference for observational and experimental data*, edited by M. van der Laan, & S. Rose. NY: Springer Science and Business. pp. 43–66.

Rigor, G. W. (2003). *Admissions Decision-Making Models*. College Entrance Examination Board.

Rokach, L. (2019). *Ensemble learning*. 2nd ed. World Scientific.

Sagi, O., & Rokach, L. (2018). "Ensemble learning: A survey." *WIREs Data Mining and Knowledge Discovery*, *8*(4), e1249. https://doi.org/10.1002/widm.1249.

Saini, P., & Kumar Jain, A. (2013). "Prediction using Classification Technique for the Students' Enrollment Process in Higher Educational Institutions." *International Journal of Computer Applications*, *84*, 37–41.

Saraswat, M. (n.d.) *Beginners tutorial on xgboost and parameter tuning in r*. Accessed 2021. https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/tutorial/.

Selingo, J. (2020). *Who gets in and why-a year inside college admissions*. Scribner.

Shahandashti, S. M., & Pudasaini, B. (2019). "Proactive Seismic Rehabilitation Decision-Making For Water Pipe Networks Using Simulated Annealing." *Natural Hazards Review*, *20*. https://doi.org/10.1061/(ASCE)NH.1527-6996.0000328.

Soltys, M., Dang, H., Reilly, G. R., & Soltys, K. (2021). "Enrollment predictions with machine learning." *Strategic Enrollment Management Quarterly*, *9*(2), 11–18.

Waters, A., & Miikkulainen, R. (2013). "Grade: Machine Learning Support For Graduate Admissions." Proceedings of the 25th conference on innovative applications of artificial intelligenceBellevue, Washington, 14–18 July, 2013. Retrieved from http://nn.cs.utexas.edu/?waters:iaai13.

Xiang, Y., Sylvain, G., & Florian, M. (2017). "Generalized Simulated Annealing." In *Computational optimization in engineering, paradigms and applications*, edited by H. Peyvandi. SI: IntecOpen. 25–46.

Xiang, Y., Sylvain, G., Suomela, B., & Hoeng, J. (2013). "Generalized simulated annealing for efficient global optimization: the GenSA package for R." *The R Journal*, *5*(1), 13–28. Retrieved from https://journal.r-project.org/archive/2013/RJ-2013-002/index.html.

Zhang, W., Maleki, A., Rosen, M. A., & Liu, J. (2018). "Optimization with a simulated annealing algorithm of a hybrid system for renewable energy including battery and hydrogen storage." *Energy*, *163*, 197–207.

Zhao, Y., Lackaye, B., Dy, J. G., & Brodley, C. (2020). "A quantitative machine learning approach to master students admission for professional institutions. Paper presented at the International Conference on Educational Data Mining (EDM), MA, 10–13 July, 2020.

Author Biographies

Lucy Shao is a Ph.D. student in Biostatistics at the University of California, San Diego. Previously, she worked at Analytic Studies & Institutional Research at San Diego State University on student learning and institutional operation. Currently, her research focuses on causal inference and high-dimensional statistics. Lucy received an M.S. in Statistics from UC San Diego.

Richard A. Levine is Professor of Statistics at San Diego State University and Faculty Advisor overseeing the Statistical Modeling Group in SDSU Analytic Studies & Institutional Research. He is former Chair of the SDSU Department of Mathematics and Statistics and past Editor of the *Journal of Computational and Graphical Statistics*. He is Associate Editor for Statistics of the *Notices of the American Mathematical Society* and is a fellow of the American Statistical Association. Professor Levine received his Ph.D. in Statistics from Cornell University.

Stefan Hyman is the Associate Vice President for Enrollment Management. In this role, he provides data-informed oversight on enrollment, retention and graduation to support both student and institutional success. He works collaboratively with campus partners to set and meet enrollment targets, and to execute the university's enrollment management goals. Stefan received his Bachelor's in Musical Studies from the State University of New York College at Potsdam, and his Master's and Ph.D. (ABD) in Musicology from Stony Brook University.

Jeanne Stronach is an experienced and effective higher education leader with 20 years of experience in institutional research. Specializing in successful team-building with internal and external partners, innovative resource management and effective strategic planning. Achievements include building self-service visualizations to support data-informed decision-making as well as spearheading a cross-divisional Data Champions program to build data community and boost data literacy. Jeanne earned her M.A. at DePaul University.

Juanjuan Fan is a Professor of Statistics and Data Science in the Department of Mathematics and Statistics, and serves as a Faculty Advisor at the Analytic Studies & Institutional Research (ASIR), at San Diego State University. Her research interests include survival analysis, decision trees and random forests, and observational study data. Working with her students and collaborators, she has published many papers assessing student success studies and solving various problems in educational data mining. Professor Fan received her Ph.D. in Biostatistics from the University of Washington.