http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 34, No. 2, March–April 2022, pp. 807–816
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

JANOS: An Integrated Predictive and Prescriptive Modeling Framework

David Bergman,[a] Teng Huang,[b,*] Philip Brooks,[c] Andrea Lodi,[d] Arvind U. Raghunathan[e]

[a] Department of Operations and Information Management, School of Business, University of Connecticut, Storrs, Connecticut 06268; [b] Lingnan (University) College, Sun Yat-sen University, Guangzhou 510275, China; [c] Optimized Operations, LLC, Boston, Massachusetts 02116; [d] CERC and MAGI, Polytechnique Montréal, Montréal, Québec H3C 3A7, Canada; [e] Mitsubishi Electric Research Laboratories, Cambridge, Massachusetts 02139
*Corresponding author

Contact: david.bergman@uconn.edu, https://orcid.org/0000-0002-5566-5224 (DB); huangt258@mail.sysu.edu.cn, https://orcid.org/0000-0001-5718-2333 (TH); phil.brooks24@me.com, https://orcid.org/0000-0003-1220-9349 (PB); andrea.lodi@polymtl.ca, https://orcid.org/0000-0001-9269-633X (AL); raghunathan@merl.com, https://orcid.org/0000-0003-3173-3875 (AUR)

Received: November 20, 2019
Revised: May 3, 2020; August 2, 2020
Accepted: September 4, 2020
Published Online in Articles in Advance: September 28, 2021

https://doi.org/10.1287/ijoc.2020.1023

Abstract. Business research practice is witnessing a surge in the integration of predictive modeling and prescriptive analysis. We describe a modeling framework JANOS that seamlessly integrates the two streams of analytics, allowing researchers and practitioners to embed machine learning models in an end-to-end optimization framework. JANOS allows for specifying a prescriptive model using standard optimization modeling elements such as constraints and variables. The key novelty lies in providing modeling constructs that enable the specification of commonly used predictive models within an optimization model, have the features of the predictive model as variables in the optimization model, and incorporate the output of the predictive models as part of the objective. The framework considers two sets of decision variables: *regular* and *predicted*. The relationship between the regular and the predicted variables is specified by the user as pretrained predictive models. JANOS currently supports linear regression, logistic regression, and neural network with rectified linear activation functions. In this paper, we demonstrate the flexibility of the framework through an example on scholarship allocation in a student enrollment problem and provide a numeric performance evaluation.

Summary of Contribution. This paper describes a new software tool, JANOS, that integrates predictive modeling and discrete optimization to assist decision making. Specifically, the proposed solver takes as input user-specified pretrained predictive models and formulates optimization models directly over those predictive models by embedding them within an optimization model through linear transformations.

History: Accepted by Ted Ralphs, Area Editor for Software Tools.

Supplemental Material: The software that supports the findings of this study is available within the paper and from the IJOC GitHub software repository (https://github.com/INFORMSJoC) at [https://doi.org/10.5281/zenodo.4017796].

Keywords: predictive modeling • prescriptive analysis • discrete optimization • solver

1. Introduction

There has been significant proliferation of research in and application of machine learning and discrete optimization. These two analytical domains have frequently been used in a single business decision-making problem but for different purposes. Machine learning techniques have typically been used to predict what is likely to happen in the future, while optimization methods have been used to strategically search through feasible solutions.

However, the integration of the two analytics streams is sometimes handled sequentially without full integration (Deng et al. 2018), with the outputs of machine learning models only used as parameters in an optimization model. As an example, suppose a neural network is built to predict customer churn of a telecommunication company, with features consisting of demographic information together with service price to the customer (the lower the price, the lower the probability of customer churn). How then should the company set service price in order to maximize revenue? The features of the predictive model are variables in the decision problem, and the output of the predictive model for any choice of the decision variables is a part of the objective function. This makes the optimization problem of maximizing revenue particularly challenging. There are few, if any, techniques, much less tools, available for solving such an optimization problem in the way proposed. This paper seeks to fill that void by introducing JANOS,[1] a modeling

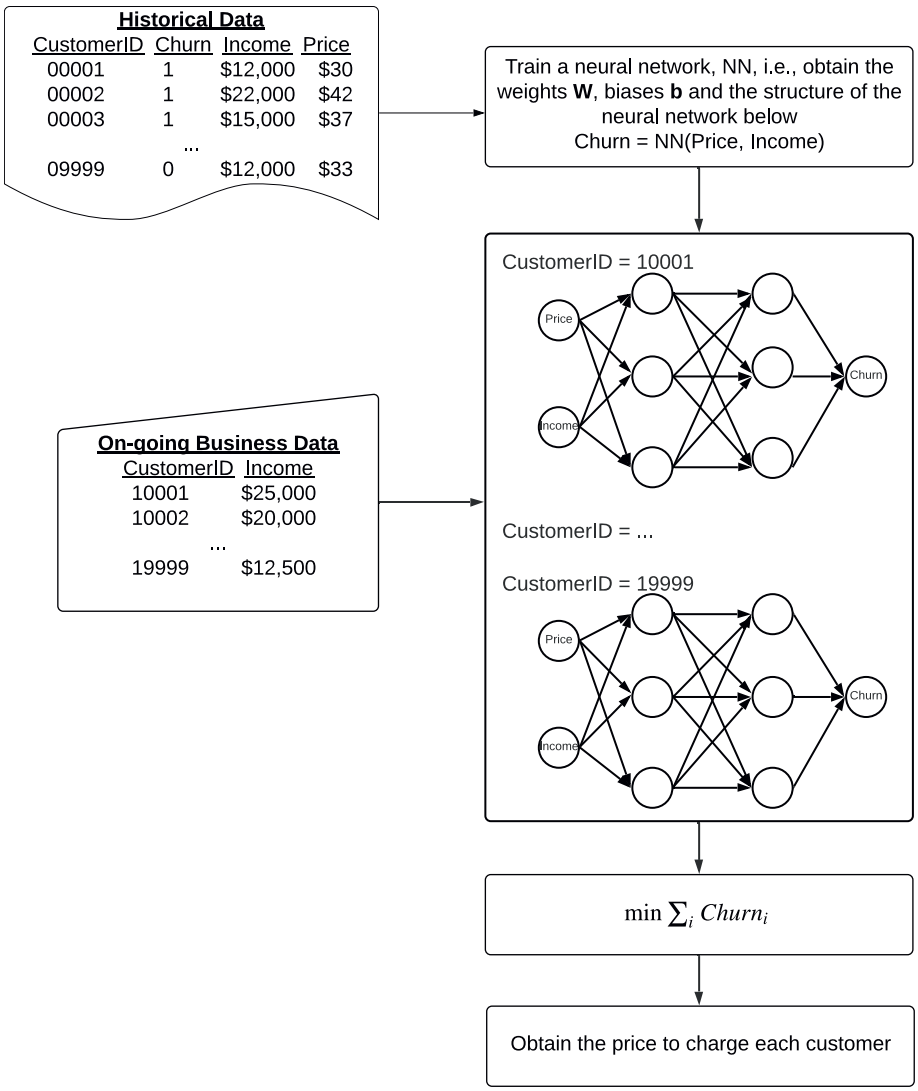
framework for joint predictive-and-prescriptive analytics. Figure 1 depicts how one could use JANOS for a similar business problem of minimizing customer churn. First, the user would train a machine learning model, in this case a neural network, for predicting customer churn. Then, the trained model would be replicated any number of times to represent either each customer or each segment of customers, and an optimization model where the neural network is embedded is formulated and solved.

JANOS allows a user to specify portions of an objective function as commonly utilized predictive models, whose features are fixed or are variables in the optimization model. JANOS currently supports predictive models of the following three forms: linear regression, logistic regression, and neural networks (NNs) with rectified linear (ReLU) activation functions. These models are commonly used in practice, and the framework is easily extensible to other predictive models.

We decided to only include the output of the predictive models in the objective function because the typical setting is that a predictive model provides just an expected value for the dependent variable. Adding the outputs in the constraints is possible, but it would result in satisfying constraints only in expectation. Constrained stochastic optimization problems sometimes require chance constraints or other considerations, so in this iteration of the solver we opted for a simplistic setting.

To embed these predictive models in an optimization model, we utilized linearization techniques. For linear regression this is straightforward. For logistic regression, we employ a piecewise linear approximation. Details are provided in Section 4.2. For NNs, we make use of recent work that formulates an NN as a *mixed integer programming* (MIP) problem (Fischetti and Jo 2017, Serra et al. 2017, Bienstock et al. 2018, Anderson et al. 2020); we do not use the MIP to train

Figure 1. A Schematic of How JANOS Works



the NN, but rather utilize the network reformulation to produce outputs of the NN based on the input features. Details are provided in Section 4.1.

A key advantage of JANOS is that it automates the transcription of common predictive models so that they can be handled by MIP solvers. Thus, researchers and practitioners are relieved of this onerous task of reformulating the predictive models into tractable constraints. A different model for predictive variables can quickly be substituted out without much effort from the modeler, enabling the user to quickly compare the optimal decisions when different predictive models are used. Finally, a researcher or practitioner looking to apply an algorithmic approach to the type of decision-making problem discussed in this paper would not have to add all parameters of a predictive model with an optimization model; for example, a neural network with three layers of 10 nodes each would have hundreds of parameters, and JANOS automates that transformation. In future releases, additional predictive models will be added as well as more advanced reformulations and algorithmic implementations.

1.1. Contributions

JANOS is built in Python and calls Gurobi (Gurobi Optimization 2019) to solve MIPs. A user specifies an optimization model through standard modeling constructs that share similarities to those in other common optimization modeling languages, for example, Gurobi's Python interface, Pyomo Gurobi (Hart et al. 2011) or JuMP (Dunning et al. 2015) in Julia (Bezanson et al. 2017).

We partition the variables in the model into two sets—the *regular variables* and the *predicted variables*. The regular variables are used to model operational constraints and objective function terms as typical variables in MIP. The predicted variables are specified through predictive models wherein some of the features depend on regular variables. The predictive models are pretrained by the user, who can load any of the three permissible predictive model forms, together with a *mapping* between the regular variables and the features. We eventually plan to integrate automated machine learning (Feurer et al. 2015) and have JANOS determine the best predictive model to associate with a given data frame. To exhibit how the framework can be used, we present as an example the allocation of scholarship offers to admitted students to optimize enrollment.

The impact of the JANOS solver on research is twofold. First, framing a problem as optimizing over standard predictive models where the decision variables are features of the predictive models requires much attention. Suppose the predictive model is a support vector regression with a radial kernel. How does one solve the optimization problem? Each predictive model added to JANOS will require research effort to investigate linearizations or advanced optimization methodologies that lead to efficient solution times.

More broadly, we envision that JANOS will be used by researchers in fields other than optimization to solve the problems they are interested in to derive the insights they desire. Currently, a researcher familiar with optimization who has a machine learning model that they would like to optimize over is unable to solve the problem of interest. Contrast that with a researcher in logistics that requires the solution of a routing problem to identify how many trucks are needed by the company. This researcher can use a standard commercial integer programming solver, solve the routing problem to identify the number of trucks that are needed by the company, and then make decisions based on that output. The logistics researcher need not know how the optimization model is solved, just that the problem of interest can be solved. We envision the same thing for the former researcher. JANOS allows the researcher to solve an optimization problem with machine learning models so that optimal decisions can be identified for real-world decision-making problems that previously could not.

The rest of the paper is organized as follows. We first review the literature related to the joint reasoning in predictive and prescriptive analysis in Section 2. The general problem addressed by JANOS is provided in Section 3. The algorithmic details of how we optimize over linear regression models, logistic regression models, and NNs are given in Section 4. The student enrollment example and a collection of experiments designed to test the efficiency of JANOS are described in Section 5. We then describe how JANOS can be downloaded and installed in Section 6. We conclude in Section 7.

2. Literature Review

Existing studies on the combination of predictive and prescriptive analytics take predictions as fixed and then make choices based on fixed predictions, for example, predictions are parameters in an optimization model (Ferreira et al. 2015). Ferreira et al. (2015) first predict sales based on a chosen number of price values and then use those fixed estimates to determine the optimal price. In their application, the predicted sales are parameters in the optimization model. This modeling approach adapts, but still lacks in full integration between the two analytics disciplines. Sales will generally depend on price, and an optimization algorithm for tackling this problem should incorporate such reasoning. Approaches using fixed-point estimates of parameters are feasible when full enumeration or partial enumeration of the collection of feasible solutions is practical. However, in instances where enumeration is not possible, aspects of the optimization algorithm

need to be integrated into the predictive model. For example, Huang et al. (2019) model a real-world problem in such a way, but only propose an exact optimization approach when simple linear regression models are used for prediction.

There are additional streams of research that combine predictive modeling and optimization. First of all, machine learning algorithms are powered by optimization techniques. For example, when fitting a simple linear regression model, the *ordinary least square* method determines the unknown parameters by minimizing the sum of the squares of the difference between the observed dependent variable values and the fitted values. In machine learning, various optimization techniques are applied so that the learning process is efficient and achieves the desired accuracy (Boyd et al. 2011). For example, Koh et al. (2007) propose an interior-point method for solving large-scale ℓ_1-regularized logistic regression problems; ALAMO (Cozad et al. 2014) uses mixed integer optimization to learn algebraic models from data; and linear programming (LP) and integer programming (IP) based methods have been proposed to assist training NNs (Bienstock et al. 2018). Additionally, there have been recent efforts on using machine learning to improve optimization algorithms. For example, Cappart et al. (2018) utilize deep reinforcement learning to improve optimization bounds, and Khalil et al. (2017) propose a generic method for learning combinatorial optimization over graphs, with a plethora of papers arising in this area (Bengio et al. 2018, Nazari et al. 2018, Lemos et al. 2019). These papers either leverage machine learning to improve optimization or leverage optimization to improve machine learning—in our setting, we integrate the two into a unified decision-making framework. There are multiple streams of research that consider machine learning and operations research in one decision-making pipeline. One stream of research is to estimate machine learning models while minimizing the regret of decisions (Elmachtoub and Grigas 2017, Demirović et al. 2019, Mandi et al. 2019, Wilder et al. 2019). Another is a framework for using auxiliary data to prescribe business decisions (Bertsimas and Kallus 2020). In particular, a conditional stochastic optimization problem minimizing uncertain costs is formulated, while our framework does not explicitly model the uncertainty in the predictions.

There appear many recently published and on-going works that optimize over supervised learning models. For example, Paul et al. (2018) optimize over a nonparametric tree choice model; Mišić (2017) and Biggs et al. (2017) optimize over tree ensembles; and Aouad et al. (2019) and Feldman et al. (2019) optimize over multinomial logistic models.

There exist some studies on optimizing over neural network structures. For example, in Lam et al. (2000), a dual-objective optimization algorithm is formulated to optimize two neural-network outputs. They use a heuristic algorithm for finding the optimal values of the decision variables by searching over a small enumerated collection of possible values. Schweidtmann and Mitsos (2019) present a method for a global optimization problem with a single embedded neural network. They utilize the convex and concave envelops of the activation function they employ, the hyperbolic tangent function. In JANOS, we restrict to only ReLU functions as activation functions, enabling us to formulate a network flow model to optimize over any number of neural networks through a *mixed-integer linear programming* (MILP) transformation.

3. Problem Description

JANOS seeks to solve problems formulated as (PROBLEM-ORI):

$$\max_{x} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{k=1}^{n_2} d_k y_k \qquad \text{(PROBLEM-ORI)}$$

$$\text{s.t.} \quad \sum_{j=1}^{n_1} a_j^i x_j \le b_i, \qquad \forall i \in \{1,\dots,m\} \qquad (1)$$

$$y_k = g_k\left(\alpha_1^k,\dots,\alpha_{p_k}^k;\theta_k\right), \qquad \forall k \in \{1,\dots,n_2\}, \qquad (2)$$

$$\alpha_l^k \text{ is given}, \qquad \forall l \in \{1,\dots,q_k\}, \quad \forall k \in \{1,\dots,n_2\}, \qquad (3)$$

$$\alpha_l^k = e_l^k \cdot x, \qquad \forall l \in \{(q_k+1),\dots,p_k\}, \quad \forall k \in \{1,\dots,n_2\}, \qquad (4)$$

$$x_j \in X_j, \qquad \forall j \in \{1,\dots,n_1\}. \qquad (5)$$

The variables $x = (x_1,\dots,x_{n_1})$ are *regular variables* and the variables $y = (y_1,\dots,y_{n_2})$ are *predicted variables*. Each variable x_j belongs to a finite or continuous set X_j and is constrained via linear inequalities. Each predicted variable y_k is associated with a predictive model g_k, with features $\alpha^k = (\alpha_1^k,\dots,\alpha_{p_k}^k)$. Each g_k is assumed to be a pretrained predictive model (a linear regression, logistic regression, or neural network with a rectified linear activation function) so that the parameters θ_k are fit prior to optimization by the user. Model g_k has p_k *features*, $(\alpha_1^k,\dots,\alpha_{p_k}^k)$. The first $q_k, 1 \le q_k \le p_k$, features of predictive model g_k are fixed and given, while each of the remaining $(p_k - q_k)$ features are regular variables, linked through the function e_l^k, an n_1-length binary unit-vector with a 1 in the coordinate of the associated regular variable. Note that a single pretrained model can be used as multiple g_k's.

4. Algorithmic Details

In this section, we summarize how JANOS handles linear regression, logistic regression, and NN models. If y is determined by a linear regression model, the function g

is linear and the construction is straightforward. We construct (PROBLEM-ORI) and feed the model to Gurobi. If y is a predicted value of an NN, y is obtained from a network flow model. The details are in Section 4.1. If y is a predicted value of a logistic regression model, we partition the range of the *log-odds*, that is, $X\beta$ as in $y = (1 + e^{-X\beta})^{-1}$, into a collection of intervals and use the mean of the **sigmoid** value of $X\beta$ within the corresponding interval to approximate y in the objective function. The details are in Section 4.2.

4.1. Optimization over Neural Networks

For every predicted variable y that is determined by an NN prediction, we associate a distinct network flow model. An NN can be viewed as a multisource single-terminal[2] arc-weighted acyclic layered digraph $\mathsf{N} = (\mathsf{V}, \mathsf{A})$. The node set V is partitioned into a collection of layers $\mathsf{V}_1 \cup \cdots \cup \mathsf{V}_l$. There is a one-to-one mapping between input features α of g and nodes $\mathsf{v} \in \mathsf{V}_1$. For any node $\mathsf{v} \in \mathsf{V}_1$, we denote the corresponding feature by $\alpha(\mathsf{v})$. Set V_l consists of a single terminal node t. For $j \in \{2, \ldots, l\}$, every node $\mathsf{u} \in \mathsf{V}_j$ has a *bias* $\mathsf{B}(\mathsf{u})$ learned during training.

Each arc $\mathsf{a} = (\mathsf{u}, \mathsf{v}) \in \mathsf{A}$ is directed from a node u in layer V_j to a node v in layer V_{j+1} for some $j \in \{1, \ldots, l-1\}$. Every arc has a *weight* $\mathsf{w}(\mathsf{a})$ learned during training.

Given values $\alpha(\mathsf{v})$ for all nodes $\mathsf{v} \in \mathsf{V}_1$, the prediction from an NN with an ReLU activation function is calculated recursively by assigning a value F_v to all nodes in the NN via the following iterative procedure and returning F_t:

- $\forall \mathsf{v} \in \mathsf{V}_1, \mathsf{F}_\mathsf{v} = \mathsf{G}_\mathsf{v} = \alpha(\mathsf{v})$;
- for $j = 2, \ldots, l-1$, $\forall \mathsf{v} \in \mathsf{V}_j$, $\mathsf{G}_\mathsf{v} = \sum_{\mathsf{u} \in \mathsf{V}_{j-1}} \mathsf{w}((\mathsf{u}, \mathsf{v})) \cdot \mathsf{F}_\mathsf{u} + \mathsf{B}(\mathsf{v})$, $\mathsf{F}_\mathsf{v} = \max\{0, \mathsf{G}_\mathsf{v}\}$; and
- $\mathsf{F}_\mathsf{t} = \mathsf{G}_\mathsf{t} = \sum_{\mathsf{u} \in \mathsf{V}_{l-1}} \mathsf{w}((\mathsf{u}, \mathsf{t})) \cdot \mathsf{F}_\mathsf{u} + \mathsf{B}(\mathsf{u})$.

Here G_v is the input of the ReLU function, and F_v is the output of the ReLU function.

We further define z_u $\forall \mathsf{u} \in \mathsf{V}$ as a binary variable indicating if $\mathsf{G}_\mathsf{u} > 0$. With this interpretation, one can formulate the following model (MOD-NN) to calculate y based on the inputs $\alpha(\mathsf{v})$, $\forall \mathsf{v} \in \mathsf{V}_1$, that are the features, which can be fixed constants or functions of the decision variables. We have

$$y = \mathsf{F}_\mathsf{t} \qquad \text{(MOD-NN)}$$

$$\mathsf{F}_\mathsf{v} = \mathsf{G}_\mathsf{v}, \qquad \forall \mathsf{v} \in \mathsf{V}_1 \cup \mathsf{V}_l \qquad (6)$$

$$\mathsf{G}_\mathsf{v} = \alpha(\mathsf{v}), \qquad \forall \mathsf{v} \in \mathsf{V}_1, \qquad (7)$$

$$\mathsf{G}_\mathsf{v} = \sum_{\mathsf{u} \in \mathsf{V}_{j-1}} \mathsf{w}((\mathsf{u}, \mathsf{v})) \cdot \mathsf{F}_\mathsf{u} + \mathsf{B}(\mathsf{u}), \quad \forall j \in \{2, \ldots, l\}, \ \forall \mathsf{v} \in \mathsf{V}_j, \qquad (8)$$

$$-M \cdot (1 - z_\mathsf{v}) \le \mathsf{G}_\mathsf{v} \le M \cdot z_\mathsf{v}, \qquad \forall \mathsf{v} \in \mathsf{V}_2 \cup \cdots \cup \mathsf{V}_{l-1}, \qquad (9)$$

$$\mathsf{G}_\mathsf{v} - M(1 - z_\mathsf{v}) \le \mathsf{F}_\mathsf{v} \le \mathsf{G}_\mathsf{v} + M \cdot (1 - z_\mathsf{v}), \qquad \forall \mathsf{v} \in \mathsf{V}_2 \cup \cdots \cup \mathsf{V}_{l-1}, \qquad (10)$$

$$0 \le \mathsf{F}_\mathsf{v} \le M \cdot z_\mathsf{v}, \qquad \forall \mathsf{v} \in \mathsf{V}_2 \cup \cdots \cup \mathsf{V}_{l-1}, \qquad (11)$$

$$z_\mathsf{v} \in \{0, 1\}, \qquad \forall \mathsf{v} \in \mathsf{V} \qquad (12)$$

$$\mathsf{G}_\mathsf{v} \text{ unconstrained}, \qquad \forall \mathsf{v} \in \mathsf{V}, \qquad (13)$$

$$\mathsf{F}_\mathsf{v} \text{ unconstrained}, \qquad \forall \mathsf{v} \in \mathsf{V}. \qquad (14)$$

Constraints (6) guarantee that the ReLU activation function is not enforced in the input and output layers. Constraints (7) enforce that the values of the nodes in the input layer are the values of each input variable of the predictive model. These will either be fixed constants or the value determined by the optimization model for a single problem variable. Constraints (8) compute the input of the ReLU function of each node that is not on the first layer. Constraints (9)–(11) enforce the ReLU activation function on each hidden layer.

In (PROBLEM-ORI), $\alpha(v)$ can be a regular variable or a constant, and there will be a separate network flow formulation for each of the predicted variables that are outcomes of neural networks. We test the impact of the size of the neural network and the number of such predicted variables on the performance of `JANOS` in Section 5.

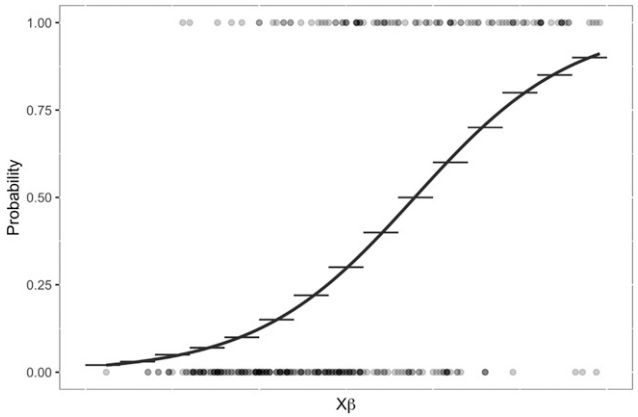
4.2. Optimization over Logistic Regression Models

`JANOS` provides a parameterized discretization for handling logistic regression prediction. Specifically, it represents the nonlinear function of a logistic regression model using a piecewise linear approximation, partitioning the range of the *log-odds*, that is, $X\beta$ as in $y_k = (1 + e^{-X\beta})^{-1}$, into a collection of mutually exclusive and collectively exhaustive intervals. The number of intervals is a parameter that can be specified by users. This is done by taking any number of intervals and forcing the output of the logistic regression model to take a fixed value within a small window of what the actual predicted value is. This idea is illustrated in Figure 2.

We partition the range of the *log-odds* into Δ intervals, $[L_\delta, U_\delta]$, for $\delta \in \{1, \ldots, \Delta\}$, where $L_{\delta+1} = U_\delta$, and we assume that the length of each interval [**sigmoid**(L_δ), **sigmoid**(U_δ)], $\forall \delta \in \{1, \ldots, \Delta\}$, is uniform. The range of the *log-odds* is computed based on the bounds of the features. We use the mean of the **sigmoid** function value within an interval to serve as a piecewise linear approximation of the actual predicted value of the logistic regression model. Specifically, for interval δ, let

$$\mathcal{V}_\delta = \frac{\log(1 + e^{U_\delta}) - \log(1 + e^{L_\delta})}{U_\delta - L_\delta}.$$

Figure 2. A Logistic Function Curve for a Collection of Points



Notes. Each point represents an element of a training set, with coordinates given by the dot product of the coefficients and their features, and the dependent variable. The curve shows a fit logistic regression, with confidence interval in estimation around the points. The horizontal lines depict the linearization currently implemented in JANOS, which associates the mean value of the function with every interval in the discretization.

The value $\mathcal{V}_\delta$ is the average value of the **sigmoid** function over all values between L_δ and U_δ, where

$$\textbf{sigmoid}(\mathbf{a}) = \frac{e^{\mathbf{a}}}{1+e^{\mathbf{a}}}.$$

We define z_δ, $\forall \delta \in \{1,\dots,\Delta\}$, as a binary variable indicating if we select a value for y_k in interval δ. Let F be the vector of features. Let β be the vector of estimated coefficients in the logistic regression model. With this interpretation, one can formulate (MOD-LOGIT) to maximize over a logistic regression model *approximately*, using the following transformation:

$$\max \quad y \qquad \text{(MOD-LOGIT)}$$

$$\text{s.t.} \quad \sum_{\delta=1}^{\Delta} z_\delta = 1, \qquad (15)$$

$$(L_\delta - L_1)\cdot z_\delta + L_1 \le F\beta, \qquad \forall \delta \in \{1,\dots,\Delta\} \qquad (16)$$

$$(U_\delta - U_\Delta)\cdot z_\delta + U_\Delta \ge F\beta, \qquad \forall \delta \in \{1,\dots,\Delta\} \qquad (17)$$

$$\textbf{sigmoid}(L_1) + (\mathcal{V}_\delta - \textbf{sigmoid}(L_1))\cdot z_\delta \le y, \qquad \forall \delta \in \{1,\dots,\Delta\} \qquad (18)$$

$$\textbf{sigmoid}(U_\Delta) + (\mathcal{V}_\delta - \textbf{sigmoid}(U_\Delta))\cdot z_\delta \ge y, \qquad \forall \delta \in \{1,\dots,\Delta\}, \qquad (19)$$

$$z_\delta \in \{0,1\}. \qquad \forall \delta \in \{1,\dots,\Delta\}, \qquad (20)$$

The value assumed by y will be approximately equal to **sigmoid**$(F\beta)$, used as the value predicted by a logistic regression model.

Constraint (15) ensures that only one interval is selected. Constraints (16) to (17) select the interval that contains the linear combination $F\beta$. Constraints (18) to (19) make sure that y equals the mean outcome value of the selected interval. Recall that F will be determined partially through fixed features and partially through decision variables.

5. Example Applications

In this section, we explore an example of allocating offered scholarships to admitted college students to exhibit the capability and flexibility of JANOS v0.0.9. All predictive models were built in Python3.6.9 using scikit-learn 0.22.2 (Pedregosa et al. 2011), and all optimization models were solved with Gurobi Optimizer v9.0.2 (Gurobi Optimization 2019) setting the MIPGap parameter to 0.1%. All experiments were run in Ubuntu 18.04.4 LTS on an Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz processor with 16 GB RAM. Data, code, and results are available on GitHub (Bergman et al. 2020).

5.1. Problem Description

The admission office of a university wants to offer scholarships to its admitted students in order to bolster the class profile, such as academic metric (e.g., GPA), and often to simply maximize the expected class size (Maltz et al. 2007).

The admission office has collected from previous enrollment years the applicants' SAT, GPA, scholarship offer, and matriculation results, that is, whether the student accepted the offer or not. This year, suppose the school is issuing N offers of admission. Moreover, suppose the budget available for offered scholarships is $\$0.2 \cdot N \cdot 10^4$, denoted by **BUDGET** henceforth. The amount of scholarships that can be assigned to any particular applicant is between \$0 and \$25,000. The admission office wants to maximize the incoming class size.

To solve this optimization problem using JANOS, one can pretrain a model to predict the probability of a candidate accepting an offer given this student's SAT, GPA and scholarship offered. The decision to make is the third feature *for each student*: the amount of scholarship to offer to each student.

We model this problem as follows (STUDENT-ENROLL):

$$\max \quad \sum_{i=1}^{N} y_i \qquad \text{(STUDENT-ENROLL)}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} x_i \le \textbf{BUDGET}, \qquad (21)$$

$$y_i = g(\mathbf{s}_i, \mathbf{g}_i, x_i; \theta), \qquad \forall i \in \{1,\dots,N\}, \qquad (22)$$

$$0 \le x_i \le 25{,}000, \qquad \forall i \in \{1,\dots,N\}. \qquad (23)$$

where

- x_i is the decision variable, that is, the amount of scholarship assigned to each student accepted;
- $\mathbf{s}_i$ is the SAT score of applicant i (standardized using the z-score);
- $\mathbf{g}_i$ is the GPA score of applicant i (standardized using the z-score);

• y_i is a predicted variable per admitted student, the outcome of a predicted model representing the probability of a candidate accepting the offer; and

• g is a predicted model pretrained to predict any candidates' probabilities of accepting an offer. The parameters $\mathbf{s}_i$, $\mathbf{g}_i$, and x_i are the predictive model's inputs. The vector θ represents the parameters of the predictive model, which we assume to be the same for each applicant. The function g can be any of the permissible predictive models with θ determined prior to optimization.

5.2. Experimental Results

We utilize randomly generated realistic student records to train predictive models and test the efficiency of the solver when solving different-sized problems with variations in parameters as well. We build the three permissible models (linear regression, logistic regression, and neural networks) with various parameter settings, that is, the number of intervals for logistic regression models and the hidden layer sizes for neural networks. Each of the models is trained on 20,000 randomly generated student records.

We then generate sets of random student records of different sizes to test the scalability of JANOS, namely, 50, 100, 500 and 1,000 admitted students. These experiment instances are produced using the same data-generating scheme as was used for building the training set. In the test data sets, each record contains only an SAT score and a GPA score, where the decision is to assign scholarships based on prediction of enrollment. We document how long it takes JANOS to solve problems of different sizes with different predictive models.

We first provide an analysis of the total runtime using various parameters for each of the models. For the linear regression there are no configurable parameters, and so we have only one setting, **LinReg**. For the logistic regression model, the main parameter of interest is the number of intervals in the discretization. For a fixed number of intervals Δ, let **LogReg**(Δ) refer to solving the model with logistic regression prediction with Δ intervals. For neural networks, there are several parameters one might tune, most apparent being the configuration of the neurons. We fix 10 nodes per hidden layer and vary the number of hidden layers to 1, 2, or 3. Let **NN**(h) refer to solving the model with neural networks with h hidden layers.

For each of the predictive model specifications, we run five instances and take the mean runtime. Figure 3 depicts the runtimes. On the x-axis we indicate the number of admitted students in the admitted pool. On the y-axis we report the runtime in seconds. **LinReg** yields the most efficient model, taking up to a second to solve. **LogReg**(Δ) takes an increasing amount of time as Δ grows, as does **NN**(h) for increasing h, but the runtimes are not prohibitively large. Note that largest instances have 1,000 logistic regression approximations or 1,000 neural network flow models.

Figure 3. (Color online) The Average Runtimes of Linear Regression Models, Logistic Regression Models, and Neural Networks with Different Scales

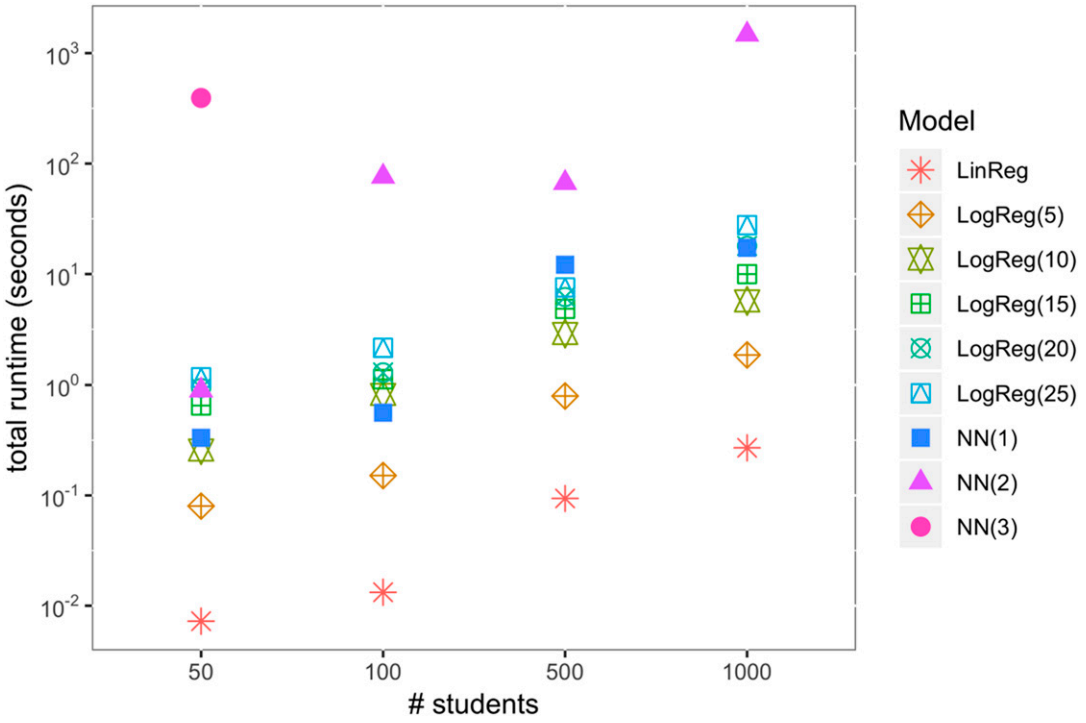
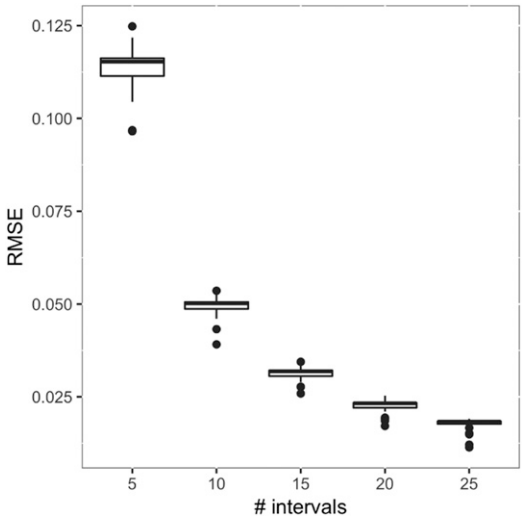


Figure 4. The Quality of the Linear Approximation of the Logistic Regression Function at Optimal Solutions



We also evaluate how well the approximation of the logistic regression performs at obtaining optimal solutions. We apply the logistic regression approximation for 10 instances with 50 students. The results are reported in Figure 4. On the x-axis we indicate the number of intervals in the approximation, and on the y-axis we report the *root mean squared error* (RMSE) of the probability estimates given by the approximation and the actual learned logistic regression evaluation at the optimal solutions obtained by the approximation. As the number of intervals increases, the approximation becomes stronger, but as discussed earlier, this increases runtime.

In order to evaluate the expected improvement in solutions obtained from JANOS over what might be done in practice, we evaluate the solution obtained by JANOS and compare it with the following heuristic that can be employed for any predictive model g for this application:

1. Sort the accepted students in nonincreasing order of

$$g(\mathbf{s}_i, \mathbf{g}_i, \$25,000 \mid \theta) - g(\mathbf{s}_i, \mathbf{g}_i, \$0 \mid \theta).$$

2. Following the above order, allocate the maximum permissible aid (i.e., \$25,000) until the maximum budget is reached.

This is a realistic heuristic because it greedily assigns scholarship to the students in the order of those that are most sensitive to scholarship. Results obtained using this heuristic are listed under "Heuristic" in Table 1.

There are other ways that a practitioner who is well-versed in both optimization and predictive modeling might address a decision problem of this sort. For example, in this application, one could discretize the domain of the decision variables to take values $D = \{0, 1, \ldots, 25,000\}$ and then evaluate the predictive model $g(\cdot)$ at each value for each admitted student. However, such a transformation results in a pseudo-polynomial size model and also admits only an approximation. Note that if desired, one can model the problem in this way directly in JANOS by declaring the decision variables as discrete and setting their domain to D. So we provide results evaluated using JANOS by setting the regular variables as discrete in $\{\$0, \$5,000, \$10,000, \$15,000, \$20,000, \$25,000\}$, listed under "JANOS_Discrete" in Table 1. Results obtained by solving (STUDENT-ENROLL) are listed under "JANOS_Continuous" in Table 1.

Table 1 reports results from the experimental evaluation. In particular, for logistic regression (**LogReg**(Δ)) and neural network prediction (**NN**(1)) models, we report for 500 and 1,000 admitted students the expected number of enrolled students based on the allocation determined by JANOS and the heuristic described above. We also report for both models and for each N the percent reduction in admitted student declination of admission.

The results indicate that, simply by a more careful assignment of scholarship and making no other changes, JANOS can provide a substantial improvement in expected matriculation rates. In particular, we

Table 1. Improvement by JANOS over a Basic Heuristic

				Expected % Reduction in Declination	
LogReg	Heuristic	JANOS_Discrete	JANOS_Continuous	JANOS_Discrete vs. Heuristic	JANOS_Continuous vs. JANOS_Discrete
500	243.17	297.45	308.59	21.13%	5.50%
1,000	467.16	576.35	599.95	20.49%	5.57%
				Expected % Reduction in Declination	
NN	Heuristic	JANOS_Discrete	JANOS_Continuous	JANOS_Discrete vs. Heuristic	JANOS_Continuous vs. JANOS_Discrete
500	243.28	299.67	311.86	21.97%	6.08%
1,000	467.44	580.66	606.57	21.26%	6.18%

see that the heuristic described above would yield under a 50% expected matriculation rate on average, while the solutions obtained using JANOS yield approximately 60% expected matriculation. Using a discretization of monetary increments within JANOS yields slightly lower expected matriculation rates than a continuous domain, but is a realistic setting for a university where giving scholarships in fixed increments is more common, even though more expected enrollments can be realized from allowing the university any level of scholarship for any student, that is, allowing continuous assignment of scholarships. In summary, going from the heuristic to using a discretization can reduce the number of declined admissions by about 20%, and moving from the discretized model to the continuous model can reduce declined admissions by an additional 6%. This example exemplifies the improvement in decision-making capability that JANOS can obtain.

6. Accessing the Solver

JANOS works with Python3 and currently requires Gurobi for optimization and sklearn for predictive modeling. You also must have numpy and matplotlib installed. Please refer to JANOS's website (http://janos.opt-operations.com) for more information, where a user manual, quick start guide, and examples are provided, including all data used for the experimental analysis in this paper.

7. Conclusions and Future Work

We propose a modeling framework JANOS that integrates predictive modeling and prescriptive analytics. JANOS is a useful tool for both practitioners and researchers who are seeking to integrate machine learning models within a discrete optimization model. JANOS can be expanded in many directions. Adding the capability to handle additional predictive models like random forests will greatly expand the potential for use in industry. Investigating how other predictive models, like support vector regressions with an arbitrary kernel function, can be incorporated is also an interesting avenue for research. Another direction is to explore alternative MIP formulations for existing predictive models, such as the MIP formulations for trained NNs in Anderson et al. (2020). Incorporating uncertainty in the predictions and making decisions that account for their impact as suggested in Bertsimas and Kallus (2020) is a further direction to explore.

Endnotes

[1] A play on Janus, who according to ancient Roman mythology is the god of beginnings, gates, transitions, time, duality, doorways and is usually depicted with two faces, one looking to the past (predictive) and one to the future (prescriptive) (Source: Wikipedia).

[2] The NN does not always have a single terminal. In our case, we only have one output, and so the output layer in our trained NN only has one node.

References

Anderson R, Huchette J, Ma W, Tjandraatmadja C, Vielma JP (2020) Strong mixed-integer programming formulations for trained neural networks. *Math. Programming* 183(1–2):3–39.

Aouad A, Feldman J, Segev D, Zhang D (2019) Click-based mnl: Algorithmic frameworks for modeling click data in assortment optimization. Preprint, submitted February 23, https://ssrn.com/abstract=3340620.

Bengio Y, Lodi A, Prouvost A (2018) Machine learning for combinatorial optimization: a methodological tour d'horizon. Preprint, submitted November 15, https://arxiv.org/abs/1811.06128.

Bergman D, Huang T, Brooks P, Lodi A, Raghunathan A (2020) JANOS version v2020.1023.

Bertsimas D, Kallus N (2020) From predictive to prescriptive analytics. *Management Sci.* 66(3):1025–1044.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017) Julia: A fresh approach to numerical computing. *SIAM Rev.* 59(1):65–98.

Bienstock D, Muñoz G, Pokutta S (2018) Principled deep neural network training through linear programming. Preprint, submitted October 7, https://arxiv.org/abs/1810.03218.

Biggs M, Hariss R, Perakis G (2017) Optimizing objective functions determined from random forests. Preprint, submitted June 16, https://ssrn.com/abstract=2986630.

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations Trends Machine Learn.* 3(1):1–122.

Cappart Q, Goutierre E, Bergman D, Rousseau LM (2018) Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. Preprint, submitted September 10, https://arxiv.org/abs/1809.03359.

Cozad A, Sahinidis NV, Miller DC (2014) Learning surrogate models for simulation-based optimization. *AIChE J.* 60(6):2211–2227.

Demirović E, Stuckey PJ, Bailey J, Chan J, Leckie C, Ramamohanarao K, Guns T (2019) An investigation into prediction+ optimisation for the knapsack problem. Rousseau L-M, Stergiou K, eds. *Internat. Conf. Integration Constraint Programming, Artificial Intelligence, Operations Res.* (Springer, Cham), 241–257.

Deng Y, Liu J, Sen S (2018) Coalescing data and decision sciences for analytics. *Recent Advances in Optimization and Modeling of Contemporary Problems*, TutORials in Operations Research (INFORMS, Catonsville, MD), 20–49.

Dunning I, Huchette J, Lubin M (2015) JuMP: A modeling language for mathematical optimization. *SIAM Rev.* 59(2):295–320.

Elmachtoub AN, Grigas P (2017) Smart "predict, then optimize." Preprint, submitted October 22, https://arxiv.org/abs/1710.08005.

Feldman J, Zhang D, Liu X, Zhang N (2019) Customer choice models vs. machine learning: Finding optimal product displays on Alibaba. Preprint, submitted August 15, https://ssrn.com/abstract=3232059.

Ferreira KJ, Lee BHA, Simchi-Levi D (2015) Analytics for an online retailer: Demand forecasting and price optimization. *Manufacturing Service Oper. Management* 18(1):69–88.

Feurer M, Klein A, Eggensperger K, Springenberg J, Blum M, Hutter F (2015) Efficient and robust automated machine learning. Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, eds. *Advances in Neural Information Processing Systems*, vol. 28 (Curran Associates, Inc.).

Fischetti M, Jo J (2017) Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. Preprint, submitted December 17, https://arxiv.org/abs/1712.06174.

Gurobi Optimization LLC (2019) Gurobi optimizer reference manual. Accessed June 12, 2021, http://www.gurobi.com.

Hart WE, Watson JP, Woodruff DL (2011) Pyomo: Modeling and solving mathematical programs in Python. *Math. Programming Comput.* 3(3):219–260.

Huang T, Bergman D, Gopal R (2019) Predictive and prescriptive analytics for location selection of add-on retail products. *Production Oper. Management* 28(7):1858–1877.

Khalil EB, Dilkina B, Nemhauser GL, Ahmed S, Shao Y (2017) Learning to run heuristics in tree search. Sierra C, ed. *IJCAI (U. S.)*, 659–666.

Koh K, Kim SJ, Boyd S (2007) An interior-point method for large-scale l_1-regularized logistic regression. *J. Machine Learn. Res.* 8 (Jul):1519–1555.

Lam SS, Petri KL, Smith AE (2000) Prediction and optimization of a ceramic casting process using a hierarchical hybrid system of neural networks and fuzzy logic. *IIE Trans.* 32(1):83–91.

Lemos H, Prates M, Avelar P, Lamb L (2019) Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. Preprint, submitted March 11, https://arxiv.org/abs/1903.04598.

Maltz EN, Murphy KE, Hand ML (2007) Decision support for university enrollment management: Implementation and experience. *Decision Support Systems* 44(1):106–123.

Mandi J, Demirović E, Stuckey P, Guns T (2019) Smart predict-and-optimize for hard combinatorial optimization problems. Preprint, submitted November 22, https://arxiv.org/abs/1911.10092.

Mišić VV (2017) Optimization of tree ensembles. Preprint, submitted May 30, https://arxiv.org/abs/1705.10883.

Nazari M, Oroojlooy A, Snyder L, Takác M (2018) Reinforcement learning for solving the vehicle routing problem. Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems*, vol. 31 (Curran Associates, Inc.).

Paul A, Feldman J, Davis JM (2018) Assortment optimization and pricing under a nonparametric tree choice model. *Manufacturing Service Oper. Management* 20(3):550–565.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M (2011) Scikit-learn: Machine learning in Python. *J. Machine Learn. Res.* 12:2825–2830.

Schweidtmann AM, Mitsos A (2019) Deterministic global optimization with artificial neural networks embedded. *J. Optim. Theory Appl.* 180(3):925–948.

Serra T, Tjandraatmadja C, Ramalingam S (2017) Bounding and counting linear regions of deep neural networks. Preprint, submitted November 6, https://arxiv.org/abs/1711.02114.

Wilder B, Dilkina B, Tambe M (2019) Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. *Proc. of the 33rd AAAI Conf. Artificial Intelligence*, vol. 33 (AAAI Press, Palo Alto, CA), 1658–1665.