Received: 27 November 2018 | Revised: 12 June 2020 | Accepted: 13 June 2020 | Published on: 9 August 2020

DOI: 10.1002/smj.3215

RESEARCH ARTICLE

STRATEGIC MANAGEMENT JOURNAL WILEY

Machine learning for pattern discovery in management research

Prithwiraj Choudhury[1] | **Ryan T. Allen**[1] | **Michael G. Endres**[2]

[1]Technology and Operations Management Unit, Harvard Business School, Boston, Massachusetts

[2]Data Scientist, Laboratory for Innovation Science at Harvard, Harvard University, Cambridge, Massachusetts

Correspondence
Prithwiraj Choudhury, Technology and Operations Management Unit, Harvard Business School, Boston, MA.
Email: pchoudhury@hbs.edu

Abstract

Research Summary: Supervised machine learning (ML) methods are a powerful toolkit for discovering robust patterns in quantitative data. The patterns identified by ML could be used for exploratory inductive or abductive research, or for post hoc analysis of regression results to detect patterns that may have gone unnoticed. However, ML models should not be treated as the result of a deductive causal test. To demonstrate the application of ML for pattern discovery, we implement ML algorithms to study employee turnover at a large technology company. We interpret the relationships between variables using partial dependence plots, which uncover surprising nonlinear and interdependent patterns between variables that may have gone unnoticed using traditional methods. To guide readers evaluating ML for pattern discovery, we provide guidance for evaluating model performance, highlight human decisions in the process, and warn of common misinterpretation pitfalls. The Supporting Information section provides code and data to implement the algorithms demonstrated in this article.

Managerial Summary: Supervised machine learning (ML) methods are a powerful toolkit that might help managers and researchers discover interesting patterns in large and complex data. We demonstrate this by using several ML algorithms to investigate the drivers of employee turnover at a large technology company. We evaluate the performance of the models, and use visual tools to interpret the patterns revealed. These patterns can be useful in understanding turnover, but

we caution not to confuse correlation with causation. These methods should be viewed as "exploratory" and not conclusive proof of relationships in the data. Our guidance can be helpful for managers evaluating analysis conducted by data scientists in their organizations.

KEYWORDS

abduction, decision trees, exploratory data analysis, induction, machine learning, neural networks, partial dependence plots, pattern discovery, random forests, ROC curve, supervised machine learning

1 | INTRODUCTION

Machine learning (ML) methods represent an exciting but underutilized toolkit for strategy and management researchers.[1] Greater adoption of these methods could be facilitated by illustrating relevant applications of ML to research in our fields. This article attempts to make progress in this direction by using real-world data to demonstrate a specific application of supervised ML methods: *as an exploratory tool to discover robust patterns in quantitative data*. These patterns can be used as an "observation" for further exploratory inductive or abductive research. These observations can help researchers formulate better hypotheses grounded in data, which can later be deductively tested using traditional econometric tools. The pattern discovery capabilities of ML could also be helpful during post hoc analysis of traditional regression results to detect patterns that may have gone unnoticed.

In addition to demonstrating the application of ML as a useful tool for pattern discovery, this article also provides guidance for readers to evaluate the work of researchers who use this tool. As a new methodological framework in our fields, it is important to establish a basis for how to evaluate whether researchers made appropriate choices when applying machine learning methods. In other words, what should readers look for in the methods section of the paper that uses ML methods for pattern discovery? In summary, while the first goal of this article is to illustrate the use of machine learning for exploratory pattern discovery, the second goal is to provide guidance for readers to evaluate such work.

Using ML for pattern discovery should be viewed as a complement (not a substitute) to traditional econometric hypothesis testing (Mullainathan & Spiess, 2017). In the traditional econometric approach, researchers typically specify a model, which yields coefficients that represent the best-fitting relationship between y and X given the specified model structure. This procedure imposes strict functional form assumptions, but yields statistically consistent, interpretable coefficients that can be used to test hypotheses (e.g., rejecting a null hypothesis). This is the preferred approach when researchers can pre-specify clear hypotheses and an appropriate model to test.

[1]To date, ML has mostly been used to classify meaning embedded in unstructured text and image data to use as variables in traditional econometric models (Choudhury, Wang, Carlson, & Khanna, 2019; Furman & Teodoridis, 2020; Gross, 2018; Kaplan & Vakili, 2015; Menon, Choi, & Tabakovic, 2018).

In contrast, ML methods can be used for discovery-driven (e.g. inductive or abductive) research. This is because, unlike traditional methods, ML algorithms can reveal complex patterns in X that relate to y using structure that was not specified *a priori*. Unlike econometric hypothesis testing, ML algorithms build models with flexible functional forms that maximize a model's performance using explanatory variables (X) to predict an outcome ($\hat{y}$). The resulting functional forms of the models can highlight surprising underlying relationships in the data. In other words, rather than deductively testing a model specified *ex ante* by the researcher (as is the case with traditional econometric analysis focused on inference), ML algorithms inductively build a model from the data to reveal patterns. These properties also make ML a useful tool in post hoc analysis of traditional regression results to detect patterns that may have gone unnoticed.

Thus, ML methods can potentially bring quantitative empirical researchers closer to the tradition of grounded theory, in which researchers identify patterns in the data to build theory based on data (Bamberger & Ang, 2016; Eisenhardt, 1989; Glaser & Strauss, 1967). In the broader literature in organization science, Mantree and Ketokivi (2013) state the act of reasoning on the part of managers and researchers alike takes three forms: deduction, induction, and abduction. Deductive reasoning takes the *rule* and the *explanation* as premises and derives the *observation*. Inductive reasoning combines the *observation* and the *explanation* to infer the *rule*, thus moving from the particular to the general. Abduction begins with the *rule* and the *observation*; the *explanation* is inferred if it accounts for the observation in light of the rule. For example, if marbles in a bag are white (*rule*) and I am given a white marble (*observation*), then perhaps the marble came from the bag (*explanation*) (Mantree & Ketokivi, 2013). We argue that ML methods could provide researchers with a novel and robust *observation*. The ML methods do not build theory itself—rather they represent tools which can generate an observation that aids the process of building theory. The process may be inductive or abductive, depending on which is taken as given—the *explanation* or the *rule*.

To illustrate the use of ML for pattern discovery, we implement several ML algorithms using employee turnover data from a large technology company. Most ML-built models do not yield familiar coefficients, so interpretation can be difficult. Fortunately, regardless of the algorithm used to build a model, we can visualize the relationship between $y$and X by using *partial dependence plots* (Friedman, 2001; Zhao & Hastie, 2019). This tool displays how the predicted outcome changes in response to a variable, conditional upon all other variables in the model. For our dataset, partial dependence plot visualizations of the models uncover an interesting pattern in the data that is robust across algorithms. A small group of employees who scored poorly in onboarding training were dramatically more likely to leave in their first 6 months at the company. If we had estimated a naïve global linear fit (e.g., using a logistic regression with a coefficient for each explanatory variable), we would have found a statistically significant *negative* relationship between training performance and turnover probability. In fact, only the small subset of employees who scored poorly during training was more likely to leave, and only during the first 6 months. This effect was large enough to drive a negative global effect at odds with the true *positive* effect for the majority of employees. A well-trained econometrician might discover these or similar patterns in the data without ML methods, but it would be difficult and time consuming to do so in a systematic way, especially with a larger number of covariates and a large set of possible interactions or nonlinearities. This example serves as a proof-of-concept that ML can be useful for discovering meaningful patterns in the data that may have gone unnoticed—potentially leading to imprecise measurement and incomplete views of empirical relationships.

We also provide guidance on evaluating empirical work that uses ML methods for pattern discovery. First, we summarize and provide guidance on making effective decisions at each step of the ML process, from selecting covariates to evaluating model performance. These guidelines can be helpful to both researchers and readers of work that uses ML methods for pattern discovery. Second, to evaluate performance of models, we discuss various metrics and illustrate the use of two plots: (a) a plot that compares training and validation loss (i.e., error); and (b) a receiver operating characteristic (ROC) plot, which is a graphical comparison of the rate of true and false positives. Third, we warn of pitfalls that often lead to misinterpretation of ML results and emphasize throughout the paper that ML is not a license to bypass rigorous causal thinking. ML analysis should be considered exploratory rather than as a result of a causal test. Furthermore, we caution against testing ML-identified patterns using the same dataset as though they were prespecified hypotheses. This would be a form of hypothesizing after the results are known (HARKing), which is a violation of the assumptions of deductive hypothesis testing.

In summary, effective use of ML requires human agency and expertise. The name "machine learning" should not be taken literally—as we illustrate, human researchers using these methods make meaningful decisions in every step of the analysis (summarized in Table 1). The guidelines summarized in our paper can be helpful for researchers attempting to implement these methods, and for readers to hold them to a high standard. We now illustrate the use of ML methods for robust pattern discovery and how to visualize, interpret, and evaluate such patterns. The Supporting Information provides code and simulated data to help readers apply these tools (see footnote 2 for direct links to the code).[2]

2 | GUIDANCE FOR IMPLEMENTING AND EVALUATING ML IN RESEARCH

In this section, we provide general intuition, and a step-by-step framework for understanding ML implementation for pattern discovery. In the next section, we will demonstrate using a concrete example.

2.1 | What is machine learning? Some intuition

First, we provide some intuition on ML methods (for technical foundations of ML, see Hastie, Tibshirani, & Friedman, 2009). Consider the task of identifying chairs in images (example from Autor, 2015). We can feed an ML algorithm thousands of example images marked as "chair" or "not chair". The algorithm discovers complex nonlinear and interdependent relationships in pixel clusters that are correlated with images labeled as a "chair." The algorithm is adjusted to build many different models of these correlations. Finally, the model that performs best on out-of-sample images is selected as the final model.[3]

[2]Python version of code: https://www.kaggle.com/ryanthomasallen/online-appendix-for-cae-2020-python;
R version of code: https://www.kaggle.com/ryanthomasallen/online-appendix-for-cae-2020-r

[3]Contrast that approach with a linear regression, in which we specify the functional form of a model that estimates a linear coefficient for each variable. Regardless of the true relationships in the data, this procedure will find the best fit for the specified model, with no flexibility in functional form. We could explore new model structures by manually adding interaction or polynomial terms, but trying all the possible combinations would be difficult, time consuming, and non-exhaustive.

TABLE 1 ML for pattern discovery: Guidance for evaluating human decisions

Step	Human decisions	Guidance
Step 1: Select data and explanatory variables	Which universe of variables to consider and which to include	Specify which variables were used out of the set of possible variables, and why
Step 2: Select algorithm	Choose a loss function (usually use the statistical package default) Which set of algorithms to try Prioritize objectives—For example, predictive accuracy versus model interpretability	Explicitly state the loss function Disclose all algorithms used Explain objectives and purpose of analysis
Step 3: Set regularization and other hyperparameters	Which set of hyperparameter values to try	State which hyperparameters and hyperparameter values were used
Step 4: Partition data for training, validation, and testing	How to split the training/validation and holdout test sets How many folds for cross-validation	Enough data for reliable validation and testing (perhaps 70% for training/validation and 30% for holdout test). 10-fold cross-validation is common. More folds can yield better results but take longer/more computation.
Step 5: Apply preprocessing steps	Applying "feature engineering" to variables (including scaling) How to treat missing data (e.g., drop observations, impute values, etc.)	Describe manipulations of variable values Report how sensitive results are to different choices of treating missing data
Step 6: Fit model on training set and evaluate predictive performance on validation set	Which metric of predictive performance to use. For example, log-loss score, accuracy, F_1 score, average precision, AUC score, and so on	The output of the loss function is a common evaluation metric (e.g., log-loss for classification or mean-squared error for ML regression). Different metrics can work better for different objectives. For example, when the costs of inaccurate prediction are high (e.g., predicting rare diseases) consider using a metric that prioritizes recall. When they are low (e.g., recommending TV shows), perhaps give more weight to precision. Metrics like the F_1score can balance these considerations.
Step 7: Repeat steps 1–6, varying choices to maximize predictive performance	When to stop repeating the steps—When is good enough?	Ideally, stop when you reach a saturation point—The tweaks only yield very small improvements to performance.
Step 8: Evaluate final predictive performance and interpret model	Predictive performance evaluation: Which metrics and visualizations of predictive performance to use Model interpretation: Which variables and combinations of variables to plot partial dependence	Report comparisons of the performance on the holdout test set. Compare to the same performance metric as training/validation. Can also plot visualizations of predictive performance Start by plotting variables with high "variable importance". For a small number of variables, consider plotting all of them. When presenting results, state which plots you include and do not include, and why (e.g., we presented partial dependence plots for all variables with nonlinear or interactive patterns)

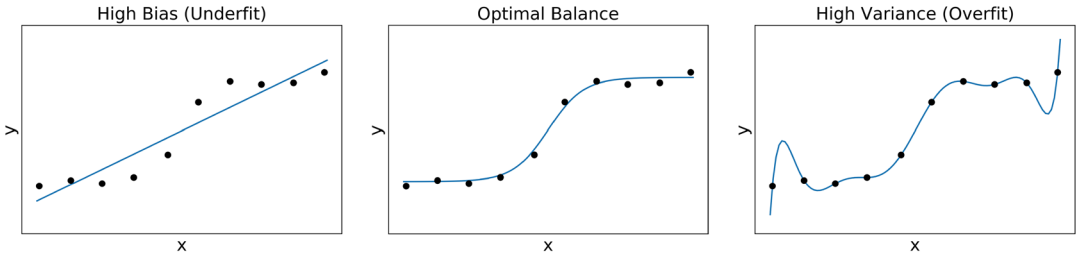

FIGURE 1 Illustrative depiction of bias-variance trade-off. *Note:* Machine learning excels at balancing the bias variance trade-off. The left panel illustrates a high bias/low variance fit of the points, the right panel illustrates a high variance/low bias fit, and the middle panel represents a reasonable balance between bias and variance [Color figure can be viewed at wileyonlinelibrary.com]

[Correction added on 10 December 2020, after first online publication: Figure 1 has been corrected.]

A strength of ML models is the capacity to discover complex relationships, but it is also important that models are generalizable. After all, the best model to predict in-sample outcomes for any dataset would simply specify the actual outcome for each data point. Yet such a model would not be generalizable, and would perform very poorly out-of-sample. Instead, ML algorithms attempt to find a model that best fits a sample dataset without *overfitting*, so the model performs well out-of-sample. The tension between fitting the in-sample data perfectly and generalizing to out-of-sample data is known as the bias-variance trade-off (Figure 1). As a model is overfit (i.e., relies on idiosyncrasies of in-sample data for prediction), its bias decreases but its variance increases, making it less generalizable. An underfit model is biased because it is too simple to describe the data.

The ML approach includes an arsenal of techniques such as cross-validation and regularization (discussed later) that limit a model's capacity to describe in-sample data. These limitations are necessary for the model to perform well out-of-sample. An important idea in implementing ML is experimenting with different constraints on model descriptive capacity to find the model that performs the best out-of-sample.[4]

Showing the actual mathematical details of ML implementation is beyond the scope of this article. However, it is helpful to understand a few terms for the sake of intuition. The mathematical objective of any ML algorithm is to build a model that minimizes a loss function (aka objective function or error function). *The loss function is simply a way to measure the error of a model*—to punish models for predictions that do not match the observed data. Certain loss functions are better for certain tasks. For example, we might want the loss function for a medical diagnosis algorithm to punish models for false negatives more than false positives. In practice, the default loss functions provided by statistical packages are usually sufficient. Throughout this article, we use a loss function foundational for many classification problems: the log-loss function.[5]

[4]Once we have a model built by an ML algorithm, why cannot we use the model to make inferences about underlying relationships in our data? The reason is that ML algorithms build a model based on how well it predicts the outcome, not whether the model is "correct". The algorithm may substitute the true explanatory variable with a highly correlated variable that has no effect on the outcome in the real world. There are other issues as well, such as the fact that it is difficult to calculate standard errors that account for how the model was selected. Thus ML algorithms' strength (flexibility fitting many different functional forms) can be an "Achilles' heel" for inference (Mullainathan & Spiess, 2017). Although causal inference is not the focus of this article, there has been some notable progress in designing ML methods that can be helpful for causal inference under certain conditions. (Athey & Imbens, 2015; Zhao & Hastie, 2019)

[5]The log-loss function is $\mathcal{L}(\theta) = -\frac{1}{n}\sum_i y_i \log[h_\theta(x_i)] + (1-y_i)\log[1-h_\theta(x_i)]$. The symbol θ represents the model parameters. The symbol $h_\theta(x_i)$, the "hypothesis", represents the predicted probabilities of the model given an observation x_i. The symbols y_i and n represent the outcome variable and the number of observations.

It is also important to clarify differences between the following terms: "algorithm", the "model," and the "loss function". In this article, we refer to the "algorithm" as the computational procedure that is used to build the "model". The "model" is simply a function that produces a prediction when given an input of observed data. The "loss function" is used to evaluate the performance of the "model". The online Supporting Information Appendix S1 provides greater detail for conceptually understanding the loss function.

2.2 | Step-by-step implementation framework

Armed with some foundational intuition, we now provide a step-by-step framework designed to guide researchers implementing or evaluating ML. These steps outline a structured process for adjusting algorithms until they produce models that perform well in out-of-sample data. Although the term "machine" learning evokes images of machine autonomy, each step of the process requires considerable human input. Table 1 summarizes the steps of the ML process, human decisions required at each step, and guidance for how to evaluate the decisions that have been made.

2.2.1 | Step 1: Select data and explanatory variables

Arguably, the first step of any empirical analysis is to select a dataset and the set of variables to consider. As with other forms of empirical analysis, ML researchers are guided by prior literature in selecting the universe of variables to analyze. This decision is affected by the same sets of considerations and biases affecting researchers of prior methods, even in highly qualitative inductive research. As Suddaby (2006) states, "grounded theory is not an excuse to ignore the literature...constantly remind yourself that you are only human and what you observe is a function of both who you are and what you hope to see" (Suddaby, 2006; pp. 634–635). Researchers using ML for pattern discovery should heed the same caution in selecting variables, and should document and motivate which variables selected. While the set of variables chosen can be motivated by prior literature, the patterns (i.e., relationships between the variables) illuminated in the inductive exercise may be novel.

2.2.2 | Step 2: Select an algorithm

The next step is to select an algorithm that will build the predictive model. The algorithm attempts to find a model that minimizes error (i.e., the output of the loss function). In this article, we will implement three ML algorithms: decision tree, random forest, and neural network. Each algorithm uses a different computation procedure to build models for predicting $\hat{y}$ and has unique strengths and weaknesses.

There is no secret recipe for selecting the algorithm that best fits a particular situation. In practice, even experienced data scientists do not know *ex ante* which algorithm to use. Nevertheless, in this step, we provide rule-of-thumb guidance. Table 2 lays out the strengths and typical uses of some of the most popular ML algorithms. Considerations for algorithm selection include:

Regression or classification?

It is important to distinguish between regression problems (a continuous real-number dependent variable, such as stock price) and classification problems (a categorical dependent variable, such as filing for bankruptcy).[6] Some algorithms are suitable for both types of problems; they are simply used to minimize different loss functions. For example, a decision tree classifier minimizes the log-loss function, but a decision tree regression minimizes mean squared error.[7] However, using a decision tree classifier for a continuous dependent variable or a decision tree regression for a categorical dependent variable will not produce optimal results.

(For classification problems) Linear separability

Some algorithms (e.g., support vector machines) are designed to isolate data points of one classification from data points of another classification by the widest possible margin. This method is used when classifications of data points are linearly separable—that is, in which it is possible to draw a line (or a plane/hyperplane) that separates the classes. In contrast, algorithms that use the log-loss function may perform better when classes are not linearly separable. For example, a plot with overlapping points labeled $y = 1$ and $y = 0$ with no clear line of separation between the two is not linearly separable.

(For classification problems) Labels or predicted probabilities?

Predicting actual labels is a common task in practice (e.g., loan will default). However, researchers using ML classification for exploratory pattern discovery may be more interested in the model's predicted probabilities(e.g., loan has 0.26 probability of defaulting). Predicted probabilities are translated into a label using a decision threshold (e.g., loans with predicted probabilities >0.1 are labeled"default"). Algorithms that use the log-loss function are more appropriate for probabilistic interpretation than, for example, support vector machine (SVM) algorithms (which excel at hard categorization between distinct classes). Later, in this article's demonstration, we will stop one step before assigning a label using a decision threshold. This is because it is nearly impossible to predict the exact time period when a specific person will leave a company. Instead, we compare what drives the relative probabilities of turnover in each time period.

Consideration: Model capacity versus number of observations in data

With more data observations, we can use algorithms with greater "model capacity". Model capacity means fitting highly flexible functional forms to achieve higher predictive performance. For example, a neural network algorithm has high potential model capacity, and can theoretically be used to represent any nonlinear relationship. However, algorithms like neural networks with higher model capacity require more data and expertise to prevent overfitting. A highly complex neural network trained on a few hundred observations would inevitably overfit the data. In fact, high capacity models like neural networks can underperform other algorithms unless they are trained using large amounts of data.

[6]Classification problems may entail two categories (binomial classification) or more than two categories (multiclass classification).

[7]Mean squared error is $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. It is technically possible to define custom loss functions, but that topic is beyond the scope of this article. We merely want to clarify that many models can be used for both regression and classification, depending on the loss function in question. In software implementations written in Python and R, implementing a model automatically minimizes a default loss function (often log-loss for classification and mean-squared error for regression).

TABLE 2 Guidance for selecting supervised machine learning algorithms

Algorithm	(R)egression or (C)lassification?	Scale features?	Capacity	Interpretability	Speed	Ease of tuning	Notes	Common uses
Decision tree[a]	Both		◐	●	●	◐	Highly interpretable due to visualization of tree and variable importance	Quick understanding of important features and partitions in data
Random Forest[a]	Both		●	◐	◐	◐	Versatile and generally performs better than decision tree. Relative to gradient boosted trees, it is easy to tune and has a low memory footprint. Can also estimate trees in parallel.	General purpose
Neural network[a]	Both	✓	●	○	○	○	Highly flexible functional form; difficult to tune. More reliable and useful with big data. Generally harder to interpret.	Image recognition, language processing, forecasting, and more
K-nearest neighbors (KNN)	Both	✓	◐	◐	◐	◐	Lazy nonparametric estimation based entirely from values of K neighboring observations; high memory requirements.	Useful when little is known about the distribution and structure of the data
Gradient boosted tree	Both		●	◐	○	○	Estimates trees sequentially; often outperforms random forest but harder to tune, slower, and more memory needed	General purpose high performance; especially good for unbalanced data
Support vector machine	Both	✓	◐	◐	●	●	Uses hinge loss function—Good for drawing optimal boundaries between linearly separable classes; reliable with relatively few observations and many features	Image recognition (for example, character recognition) and text categorization

TABLE 2 (Continued)

Algorithm	(R)egression or (C)lassification?	Scale features?	Capacity	Interpretability	Speed	Ease of tuning	Notes	Common uses
LASSO or ridge	Both	✓	◐	●	●	◐	Easy to understand and interpret for those with econometrics background. Highly interpretable coefficients. For classification, use generalized linear model (e.g., logit) rather than OLS.	Simple methods for reducing overfitting and complexity for linear models.
Naïve Bayes	C		◐	◐	●	●	Minimal structure; strongly assumes independence of features so cannot exploit interactions; scalable for large data and reliable with few observations	Multiclass classification; text classification, such as assigning emails to "spam" or "not spam"

Note: ○, Low; ◐, medium; ●, High. These comparisons are broad generalizations that may change frequently according to circumstance. The meaning of each column in the table is as follows: *(R)egression or (C)lassification*: the "R" indicates that the algorithm is used for regression and "C" indicates the algorithm is used for classification. Many of these algorithms can be used for both. For example, when used for regression, the decision tree may be used to minimize the squared error (or "mean squared error") loss function; when used for classification, it may be used to minimize the log-loss function. It is also possible to use other loss functions or even customize loss functions. *Scale Features*: the check indicates that the algorithm is sensitive to feature scaling—that is, features (variables or functions of variables) should be transformed to a standardized scale either by using "*z*-score" or "min–max." *Capacity*: refers to the algorithm's capacity to achieve high predictive performance, usually by building a nuanced model with highly flexible functional form. Note this is highly dependent on the specific dataset. Simple algorithms can outperform complex ones on certain datasets. *Interpretability*: how easy is it to conceptualize or interpret the algorithm's resulting predictive model. *Speed*: refers to computational speed of training a model (though, of course, these comparisons depend heavily on the data being used). *Ease of tuning*: generally corresponds to fewer hyperparameter choices and models that do not require as much nuanced expertise to avoid overfitting. For similar resources, see Microsoft (2019) and Scikit Learn (2018).

[a]Demonstrated in this article.

2.2.3 | Step 3: Choose regularization and other hyperparameters

For each algorithm, we will set algorithm-specific constraints on the models it can build. *Regularization* is any constraint that restricts the descriptive capacity of a model—essentially smoothing the functional form to prevent overfitting (recall Figure 1). This is done by tuning (i.e., adjusting the values of) the regularization hyperparameters of the algorithm. A *hyperparameter* is any parameter of the algorithm that is set *before* estimating the model. Hyperparameters are not learned from the data; they are assigned to the model by the researcher. The ML algorithms we implement later in this article all have specific hyperparameters, which can be "tuned" (i.e., adjusted) to avoid underfitting and overfitting. For example, decision trees have "stopping rules" that limit the growth of the tree.

For some algorithms, an important hyperparameter is the choice of a "regularization term" (or "penalty term") to add to the loss function. Adding regularization terms to a loss function controls for overfitting by punishing the loss function for putting too much predictive weight on a variable. An example perhaps familiar to some researchers is the LASSO regression. Supporting Information section 1 describes how a regularization term in the loss function prevents overfitting.

What are the optimal values for the hyperparameters? Tuning hyperparameters is a delicate balancing act between bias (underfitting) and variance (overfitting). To find this balance, we try many hyperparameter values and see which combination produces a model that performs best out of sample.

2.2.4 | Step 4: Partition the data for out-of-sample model evaluation (training, validation, and testing)

To evaluate out-of-sample performance, we see how well the model performs on a "validation sample" distinct from the "training sample" used to train (i.e., estimate) the model. We tune the hyperparameters of the algorithm that is "learning" from training data until its predictive performance *on the validation data* is optimized. A final sample of data, the holdout test set, is kept separate from both the training set and the validation set. We use this sample to get a final estimate of predictive performance on data that were not used to train or validate the model (Step 7). A reasonable rule of thumb is to partition the data randomly into either three subsets (e.g., ∼60% training, ∼20% validation, and ∼20% holdout test) or into two subsets (e.g., ∼70% training-validation and ∼30% holdout test) to be used for k-folds cross-validation.[8]

Throughout this article, we use the second option, k-folds cross-validation. This method of cross-validation is less sensitive to the idiosyncrasies of training and validation set selection, though it is more computationally intensive. In k-folds cross-validation, the training-validation data are split randomly into k equal-sized subsets of data. One by one, each of the k subsets is used as the validation data; the other $k - 1$ subsets are used to train the model. The resulting k estimates of the validation error (i.e., output of loss function) from each model are averaged for the measurement of model performance. Taking an average is what makes model performance evaluation less subject to idiosyncrasies in any single split of the data. Throughout this article, we use 10-fold cross-validation ($k = 10$), a common choice for k.

[8]The relative size of the validation and test data can be smaller for large datasets. The key point is that the size of the validation/test set is large enough to give reliable estimates of model performance. For example, if my dataset has a billion observations, I may only need a thousand data points as a holdout test—much smaller than 20%.

2.2.5 | Step 5: Apply preprocessing steps

Preprocessing the data—including "feature engineering"[9] and handling missing data[10]—is also important for model predictive performance. It can be necessary to scale variables (i.e., features) for algorithms that calculate distance between points (e.g., neighbor methods like KNN or support vector machines) or for algorithms that use a regularization term (e.g., neural networks and LASSO). If not scaled, variables with larger magnitudes will overwhelm variables with smaller magnitudes as the algorithm assigns weights. Variables are commonly divided by "*z*-scores" or "min–max" scores, which strip units so that all numerical magnitudes are comparable across variables.

These preprocessing steps (e.g., normalizing values or missing value imputation) can *leak* information from the validation or test data into the training data. Leakage causes the out-of-sample evaluation metrics to be overly optimistic about the performance of the model. Therefore, these preprocessing steps should be done *after* splitting the data into training/validation/test partitions. Each preprocessing step should be "learned" from the training data, then applied to the validation and test data. For example, if a variable is to be normalized by a "*z*-score", then for each observation in the training, validation, and test set, one should apply the following calculation: $\frac{x - \text{mean}\left(x_{\text{training}}\right)}{SD\left(x_{\text{training}}\right)}$ (where *SD* refers to standard deviation). In the companion code for this article,[2] we demonstrate how to preprocess the data in a *pipeline*, which makes preprocessing implementation simple.

2.2.6 | Step 6: Fit the model on the training set and evaluate predictive performance on the validation set

Finally, we can use the algorithm to fit (i.e., estimate) a model. Statistical software written in R and Python make it relatively easy to fit the model (see the code in Supporting Information section 3 as an example). Under the hood, an optimization algorithm finds the function that minimizes error (output of loss function) in the training data, subject to the hyperparameter choices and model constraints.

In-depth discussion of optimization algorithms is beyond the scope of this article—we leave that to the statistical software. However, it is useful to be aware that some complex models (e.g., neural networks), may locate local rather than global optima.[11] A signal of the presence of

[9]"Feature" is another word for variables or functions of variables. "Feature engineering" refers to scaling, creating, or modifying features (e.g., bucketing a continuous variable or interacting variables).

[10]Handling missing data can heavily influence model performance. Dropping observations with missing values can severely limit the number of observations and may be misleading if the excluded data are systematically related to the outcome variable. As a solution, missing values can be imputed. Missing numerical values can simply be replaced with the variable's mean or median value, and missing categorical values can be replaced with the mode. Alternatively, missing values can be replaced with an estimated value—that is, run a regression model to learn what values predict the value for nonmissing observations and fill in missing observations with the predicted values. If a variable has many missing values and is not central to the prediction, it may be best to simply drop the variable.

[11]For example, one common algorithm is the gradient descent algorithm. Imagine that loss (i.e., error) as a function of variables *X* is represented in 3D space by a landscape where peaks represent high loss and valleys represent low loss. The gradient descent algorithm finds the steepest route down from whatever hill it is initially positioned on, and it stops when it cannot descend any farther. Thus, for nonconvex optimization problems (e.g., rugged landscapes with multiple minima) like neural nets, the initial values assigned to an algorithm can lead to substantially different predicted models. For other algorithms (e.g., those that optimize for a linear hypothesis), the loss function is convex (i.e., a landscape with one minimum), thus, this problem is not encountered. In general, however, the problem of multiple local minima can be quite challenging.

multiple local optima is that the model's fit results vary significantly with the choice of initial parameter values. Although there is no simple solution to this problem, it can sometimes be addressed with a better choice of initial parameter values or stronger regularization.

The fitted model is used to predict outcomes in the validation data, and the resulting predictions are evaluated against the true outcomes in the validation data using a performance metric. Throughout this article, we use the log-loss score (i.e., our model's error) as the measure of model performance.

2.2.7 | Step 7: Repeat steps 1–6, varying the algorithm, features, hyperparameters, and regularization choices to maximize predictive performance on validation set

It is difficult to know *a priori* the combination of algorithm, features, and hyperparameters that will yield the best model. We try many different combinations, with the goal of finding the model with the least error on the validation set. Often ML practitioners try as many combinations as is feasible, starting with the simplest algorithms.

Although the objective is to minimize the model's loss (i.e., error) on the validation set, this validation loss should not significantly diverge from the training loss. Divergence of training and validation loss is a sign that the model overfits the in-sample data at the expense of performance on out-of-sample data. Figure 2 plots the training and validation loss of the random forest model predictions as a function of one of its hyperparamters, "tree depth". We trained and evaluated the model (using 10-fold cross-validation) eight separate times, varying the "tree-depth" hyperparameter values from 1 through 8. The orange (upper) line represents the validation loss, and blue (lower) line represents the training loss of

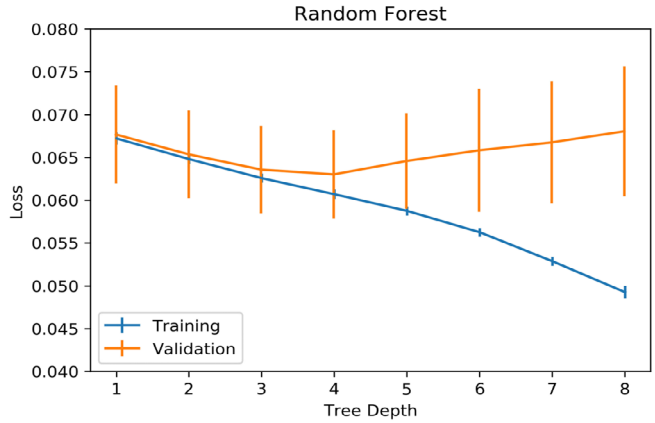

FIGURE 2 Training and validation loss as a function of the random forest "tree depth" hyperparameter. *Note:* The blue (lower) and orange (upper) lines represent training loss and validation loss for each value (1–8) of the tree depth hyperparameter. Error bars represent cross-validation standard error confidence intervals. The figure demonstrates the "elbow" where validation loss diverges from the training loss. Increasing the tree depth removes constraint from the model, allowing it to better describe the data and thereby decrease loss. However, removing too much constraint (i.e., increasing tree depth) can cause overfitting—where the model predicts well in the training data but not in the validation data. It appears that setting the tree depth hyperparameter to 3 or 4 would yield the best generalizable predictions. Note that for illustrative purposes, this random forest algorithm does not contain any regularization other than tree depth. The random forest algorithm used later in the paper differs slightly due to adjustment of other regularization hyperparameters [Color figure can be viewed at wileyonlinelibrary.com]

the model predictions across these eight hyperparameter values. Because the random forest algorithm has unbounded descriptive capacity in-sample, the training loss approaches 0 as regularization is eliminated. It appears that the best choice for tree depth would be around 3 or 4—the choice at which both training and validation losses are low, but the validation loss has not diverged from training loss.

Rather than manually tuning one hyperparameter at a time, many statistical packages include support for "grid search," "random search," and "Bayesian search" techniques. These systematically fit and evaluate the model using many combinations of a user-specified set of hyperparameter values. Despite these tools, it is usually impossible to try every possible combination of algorithm, feature, and hyperparameter values. The actual process of tuning hyperparameters can be messy and iterative in nature. The code in the Supporting Information gives a more detailed guide for implementing this process in practice.

2.2.8 | Step 8: Evaluate final predictive performance, and interpret model

After selecting the best-performing model (in the validation data), we can evaluate final predictive performance by applying the model to the holdout test set. Because this sample was not used to train or validate in the previous steps, it represents the purest out-of-sample test available to evaluate model performance. Performance of the model can be evaluated using various metrics and visualizations (which we demonstrate on our data in Section 3). Ideally, the holdout test performance should be statistically indistinguishable from the training/validation loss. If it is significantly worse than the training and validation scores, then the model has been overfit to the training/validation data. Therefore, the holdout test set is a primary safeguard against overfitting.

When applying ML for exploratory pattern discovery (the focus of this article), we can now attempt to understand the model's structure. ML-built models can be hard to interpret when they contain nonlinearities and interdependencies among explanatory variables. However, visualization tools like partial dependence plots can be extremely helpful (we demonstrate this in Section 3). We cannot always take patterns in the models at face value, or treat them as causal relationships. But the patterns that are robust across multiple ML algorithms can be very informative.

3 | DEMONSTRATION: DISCOVERING PATTERNS IN EMPLOYEE TURNOVER

3.1 | Data and setting

In our demonstration, we attempt to discover employee turnover patterns in a large Indian technology firm, TECHCO. The internal dataset covers the 1,191 entry-level employees that were deployed to any of TECHCO's nine geographically dispersed production centers in 2007. The data are structured as a panel of one observation for each month that an individual is employed at the company for up to 40 months. The data include 36,978 observations from 1,191 employees total; 25,925 observations from 833 employees in training/validation; and 11,053 observations from 358 employees in the holdout test sample. The dependent variable, *Turnover*, indicates whether the employee left during that time period ($y = 1$ if turnover, $y = 0$ if non-turnover). Our goal is to estimate the probability of turnover for a given employee at a given time.

TABLE 3 Summary statistics

	Training/validation sample			Holdout test sample		
	N	**Mean**	***SD***	***N***	**Mean**	***SD***
Turnover (left this month)	25,925	0.013	0.113	11,053	0.013	0.115
Time (months since start)	25,925	17.569	10.560	11,053	17.530	10.545
Training performance	25,925	4.546	0.320	11,053	4.528	0.328
Logical score	25,925	4.777	3.658	11,053	4.899	3.674
Verbal score	25,925	4.255	4.012	11,053	4.409	3.630
Average home region literacy	25,925	76.755	8.126	11,053	76.219	8.210
Production-center age	25,925	14.742	7.576	11,053	14.940	7.621
Distance	25,925	0.764	0.692	11,053	0.808	0.684
Language similarity	25,925	60.867	35.362	11,053	58.223	34.829
Male	25,925	0.652	0.476	11,053	0.633	0.482

Note: This table includes summary statistics for the unbalanced panel used throughout the paper. The total sample had 36,978 panel observations, one for each employee-month. The table displays separate summaries to compare the training/validation and test holdout samples. In the table, *N* refers to the number of observations, *Mean* refers to the mean value, and *SD* refers to standard deviation of the values.

Choice of explanatory variables was motivated by considerations outlined in the prior theoretical (Jovanovic, 1979) and empirical literature (e.g., Campbell, Ganco, Franco, & Agarwal, 2012; Carnahan, Kryscynski, & Olson, 2017) on employee turnover. These include employees' performance scores in an intensive three-month onboarding training course (*Training Performance*), the time in months spent at the company (*Time*), university verbal and math test scores (*Verbal Score* and *Logical Score*), date of arrival at the company (*Month Arrived*), and demographic information. The data also include the assigned production center's age (*Production Center Age*), its distance from the employee's hometown (*Distance*), and the similarity of the prevailing language in the production center's region in India to that of the employee's hometown (*Language Similarity*). Table 3 provides basic summary statistics.

3.2 | Stylized implementation: Decision tree algorithm

To develop intuition for how ML algorithms work, we apply a decision tree (a relatively simple ML algorithm) to our data. We will also fit two other algorithms, a random forest and a neural network, to compare patterns in the data across multiple algorithms.[12] Conceptual and

[12]We chose these three algorithms for two reasons. First, they are widely used general purpose algorithms that pedagogically demonstrate a variety of ML algorithm attributes: the decision tree is easily interpretable, the random forest is a highly useful general-purpose algorithm that demonstrates ensemble techniques, and the neural network is the basis of many modern technological applications of ML. Second, these algorithms can be specified to optimize the log-loss function. The log-loss function is suitable to our data because we are most interested in comparing the probabilities of turnover rather than the predicted outcome labels. Because the probability of any given employee leaving in a particular month is extremely low, it is very difficult to predict exactly when someone will leave. The purpose is not to draw clear boundaries between classes (i.e., binary predictions such as $\hat{y}=1$ or $\hat{y}=0$), but to learn relative probabilities.

implementation details (including code) for those algorithms are included in the Supporting Information section 3.

The decision tree algorithm builds a model by repeatedly splitting the data into two distinct subsets based on the values of one explanatory variable. Each subset is assigned a single value for the predicted outcome. Each split is determined based on what will minimize the model's total error (i.e., output of the loss function). Within each new subset, the procedure is repeated, splitting the data along one variable at a time to minimize the error within each subset. Each split can be represented visually as a node with two branches, creating the overall impression of a tree. A "root node" represents the first split, and "leaves" are terminal nodes with a predicted value for each subset of the data. To control for overfitting, the model is regularized by "stopping rule" hyperparameters that limit growth of the tree—for example, by limiting the maximum depth of the tree.

Figure 3 is a visual representation of the decision tree model applied to the TECHCO turnover data.[13] One of the desirable attributes of decision tree algorithms is the ease of visualization of the resulting model. For example, the top node (the "root node") of the tree in Figure 3 is labeled *Training Performance* $\leq$ 3.995. Thus, the single split that maximized predictive performance was to separate the data into the 25,557 observations whose training performance score exceeded 3.995 from the 338 observations whose scores were below 3.995.[14] Following the left-hand branch of the tree (labeled "True"), we see that within the 338 observations with low training performance, the split that maximizes predictive performance splits observations with *Time* $\leq$6.5(months) from those >6.5 (months) and so on.

This decision tree reveals an interesting pattern: the tree splits along only two dimensions, *Training Performance* and *Time*. This pattern is indicative that these dimensions may be characterized by important nonlinearities and interactions. Furthermore, the terminal nodes on the bottom left (with *Training Performance* $\leq$ 3.995, *Time* $\leq$ 6.5, and *Time* > 4.5) have a much higher proportion of turnover than any other terminal nodes. This offers clues about interesting heterogeneous effects in subsets of the data. But for now, we avoid too much interpretation based on this figure alone. The specific branches and leaves of a single decision tree can look dramatically different for different samples of the same dataset.

To triangulate these insights with other models, we also implemented a random forest (essentially a combination of many decision trees)and a neural network. For conceptual details on those algorithms, refer to Supporting Information section 2. We will compare all three models' performance and structure in the following sections of the paper.

3.3 | Metrics and visualizations for evaluating model predictive performance

Evaluating a model's predictive performance is essential not only for prediction problems, but also for evaluating how useful a model is for exploratory pattern discovery. Before examining

[13]After tuning the model by trying many combinations of possible hyperparameter values, our best performing decision tree model used the "entropy" criterion of determining splits, required each leaf to have at least 27 observations, and allowed splits only if the decrease in loss (error) was greater than 0.0004.

[14]These numbers represent those in the training data sample, out of the full sample of 36,978. Although our dataset is based on 1,191 employees tracked over 40 months, the number of observations is not a full panel of 1,191 × 40 = 47,640. This is because the data stops tracking employees who leave, so those who leave before month 40 have fewer than 40 observations.

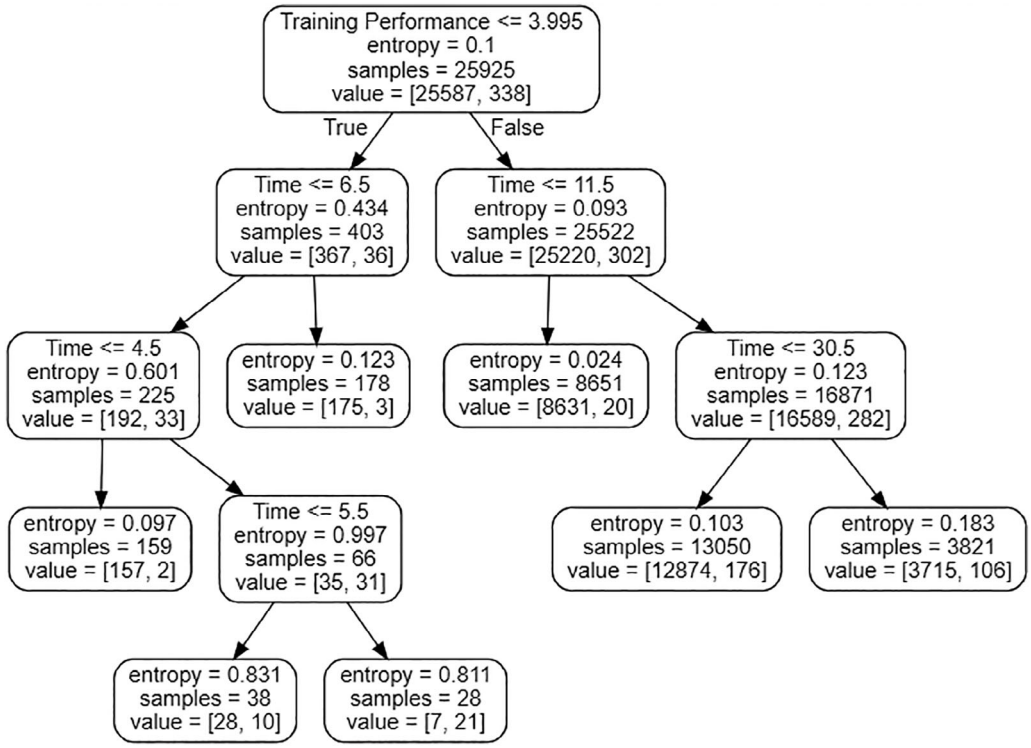

FIGURE 3 Decision tree model. *Note:* This image depicts a decision tree model that was trained to predict the dependent variable *Turnover* (1 if turnover occurs in a given time interval, 0 otherwise). The label *samples* within each node denotes the number of observations within the subset of data represented by that node. For example, the top "root node" contains all 25,925 training observations. Adding all the values of *samples* in the terminal "leaf nodes" also sums to 25,925. The label *value* within each node denotes the number of non-turnover events and the number of turnover events within that node. For example, the terminal node on the bottom left reads *value* = [28,10]. This indicates that of the 38 observations, 28 were non-turnover ($y = 0$) and 10 were turnover ($y = 1$). The probability of turnover for these observations can be represented as a probability of turnover ($10/38 = 0.26$) given the attributes on the path to the node. This image was created by graphviz in Python (see the code in Supporting Information section 3)

the structure of models for exploratory data analysis, models should be optimized to perform well. This helps ensure that researchers are using models based on some objective criteria rather than simply selecting the ones that confirm their priors. Final performance on a holdout test set can be reported using various metrics. For regression problems, a common metric is mean squared error. Common metrics for classification include the log-loss score, accuracy, true positive rate (i.e., recall), false-positive rate, positive predictive value (i.e., precision), F_1 score (i.e., harmonic mean of precision and recall), average precision (area under the precision-recall curve), and the area under the receiver operating characteristic curve (AUC) score.

It is common to think of predictive performance in terms of accuracy $\left(\frac{\text{Correct Predictions}}{\text{Total}}\right)$. Yet accuracy can be misleading, particularly when there is imbalance in the classes. For example, consider a sample in which only 1% of observations are blue and 99% are red. A model that always predicts "red" will be highly accurate at 99%. But that model will not be very useful, especially if it is very important to detect which observation is "blue".

In many situations, a better way to think about classification predictive performance is to visualize a confusion matrix (Table 4). The confusion matrix is helpful for conceptualizing how to balance the rate of false negatives and false positives predicted by a classifier. In different contexts, we may choose to prioritize one or the other. For example, when the downside of missing true positives is high (e.g., detecting rare diseases) we may choose to give more weight to recall ($\frac{\text{True Positive}}{\text{True Positive}+\text{False Negative}}$). Conversely, when recommending TV shows, precision ($\frac{\text{True Positive}}{\text{True Positive}+\text{False Positive}}$) may be more heavily weighted. Metrics like theF_1score are designed to strike a balance between precision and recall. Besides summary metrics, we also demonstrate the ROC curve visualization, which can be helpful for richer evaluations of the balance between true and false positives and negatives.

3.3.1 | Plot of ROC curve

Like the confusion matrix, the receiver operating characteristics (ROC) curve helps us visualize how well the model distinguishes between different classes (Figure 4). The curve compares the model's true positive rate ($\frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}}$) against the false positive rate ($\frac{\text{False Positives}}{\text{False Positives}+\text{True Negatives}}$), varying the cutoff threshold for distinguishing between classes. For example, in the bottom left-hand corner, no observations are classified as $\hat{y}=1$ because the predicted probability of $y = 1$ would have to be >1. The classification threshold is lowered until in the top right corner the predicted probability of $y = 1$ is ≥0, so everyobservationis classified as $\hat{y}=1$.

Intuitively, a model that classified each observation randomly would produce points along the dotted 45° line (i.e., equally likely to classify an observation as a true or false positive). Points above the diagonal line represent better-than-random classification results, and points below the line represent a classification results that are worse than random. A perfect predictive model would include a point in the very top left corner, representing a model that could give no false negatives and no false positives.

The area under the curve (AUC) metric can be used to summarize the ROC curve. The AUC score represents the actual area under the ROC curve. This number can also be interpreted as the probability that the model will rank a randomly chosen $y = 1$ observation higher than a randomly chosen $y = 0$ observation. In Figure 4, the random forest model has an AUC score of 0.746. Achieving a significantly better score with these data may not be possible because there is no hard boundary between turnover ($y = 1$) and non-turnover ($y = 0$)

TABLE 4 Confusion matrix

	Predicted	
Actual	Negative ($\hat{y}=0$)	Positive ($\hat{y}=1$)
Negative ($y = 0$)	True Negative	False Positive
Positive ($y = 1$)	False Negative	True Positive

Note: The confusion matrix is an important tool for evaluating performance when ML predictions are for actual classification labels (e.g., assigning each observation a label $\hat{y}=1$ or $\hat{y}=0$ based on a decision threshold for predicted probabilities). For the demonstration in this article, we are more interested in predicted probabilities (e.g., 0.4 probability of $y = 1$) than in assigning labels (e.g., $\hat{y}=1$ or $\hat{y}=0$).

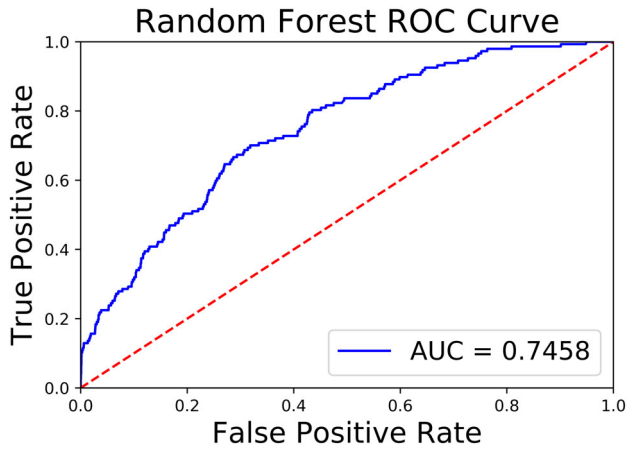

FIGURE 4 ROC curve. *Note*: The blue line plots the true positive rate ($\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$) and false positive rate ($\frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$) as the classification threshold is varied. The red dotted 45° line represents a model that randomly classifies each point (that is equally likely to classify someone as a true or false positive). Points above the diagonal line represent better-than-random classification results, and points below the line represent a classification results that are worse than random. A perfect predictive model would have a point in the very top-left corner, representing a model that gave no false negatives and no false positives. The area under the curve (AUC score) of 0.746 means that there is a 0.746 probability that the random forest model will rank a randomly chosen positive observation higher than a randomly chosen negative observation. For our data, this is probably close to the highest possible score with any model because there is no hard boundary between $y = 1$ and $y = 0$ observations (i.e., the underlying probability of turnover is less than 1 for all observations, rather than 1 in some regions and 0 in other regions) [Color figure can be viewed at wileyonlinelibrary.com]

observations. That is, the underlying probability of turnover is less than 1 for all observations, rather than 1 in some regions and 0 in other regions.

3.3.2 | Plot of training and cross-validation error (loss)

When a model's error (loss) on the training set is significantly lower than the validation set, the model may be overfitted. Figure 5 plots the training and validation loss of each model. For comparison, the plot also includes the loss from a baseline logistic regression(which includes a coefficient for each explanatory variable plus month fixed effects). The position of points on the plot indicates the training and cross-validation loss of each model; error bars are calculated from a standard deviation variation among *k*-folds. The figure helps us clearly visualize which models may be overfitted; anything above the dashed line has higher validation loss than training loss, a sign of overfitting. It appears that the random forest could be in danger of having been overfitted to the training data (it is above and to the left of the dashed line). However, the performance on the holdout test loss was 0.0633—very close to the validation loss. This mitigates the concern that the model's overfitting is hurting its out-of-sample performance.

The figure shows that random forest and decision tree algorithms both offer small performance increases over the baseline logistic regression. The explanation for such small gains in performance is that meaningful interactions and nonlinearities among variables are only relevant for a small subset of the data. Furthermore, points labeled as turnover events ($y = 1$) significantly overlap with points labeled as non-turnover events ($y = 0$), making it difficult to predict precisely which points are one or the other. In this case, these ML models may be more useful for pattern discovery than for predictive performance gains.

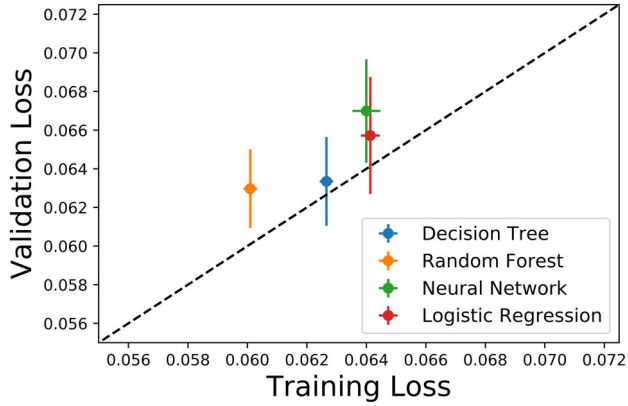

FIGURE 5 Plots of training and validation loss for each model relative to logistic regression. *Note:* This plot displays the training and validation loss for each model, including the baseline logistic regression model. Points toward the lower-left corner are better predictions (lower loss). Error bars represent a standard deviation variation in predictions yielded by the k-folds cross-validation. The dashed line represents a line along with the loss from the training set is equal to the loss on the validation set. Points above and to the left of the line represent models for which the validation loss is higher than training loss, indicating that the model may be overfitted on the training data [Color figure can be viewed at wileyonlinelibrary.com]

3.4 | Interpreting models using variable importance and partial dependence plots

After finding a high performing model, we can use the model to find interesting patterns. Many ML models are complex and can be difficult to interpret. Some models (e.g., LASSO) return familiar coefficients, and some (e.g., random forest) return relatively easily interpretable measures of *variable importance*. Other algorithms (e.g., neural networks) contain no intrinsic measure of how much each variable impacted the predicted outcome. However, regardless of the algorithm or the complexity of the model, we can interpret a model using partial dependence plots. These plots can be used to visualize the marginal effect of one or more variables on the predicted outcome of the model (Friedman, 2001; Zhao & Hastie, 2019).

3.4.1 | Variable importance

Although not as rich as partial dependence plots, examining variable importance (when possible) is a reasonable first step for interpreting ML models. Variable importance can be calculated in different ways for different algorithms, but it generally shows how useful each variable was for predicting the outcome.[15] The scale is not meaningful—only relative comparisons matter. Figure 6 plots relative variable importance for our random forest model. These variable importance values should not be interpreted like econometric coefficients but can be used to give

[15]For tree-based algorithms, variable importance is often calculated as the average decrease in node impurity each time a variable was used to split a node. Node impurity is a measure of the likelihood of misclassifying an element in the subset if it were randomly labeled according to the distribution of labels in the subset. Intuitively, a node for which all the data are labeled as "1" has low impurity. A node for which half the data are labeled "1" and half labeled "0" would be high impurity.

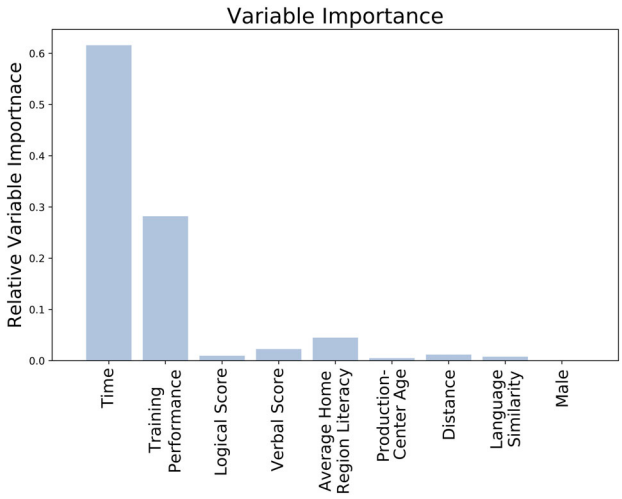

FIGURE 6 Variable importance. *Note:* This figure represents the "variable importance" (aka "feature importance") of each variable in the random forest model. This is calculated as the average decrease in node impurity across all the variable's nodes, weighted by the probability of reaching that node (i.e., number of samples that reach that node divided by total samples). Variables with higher values are more important. Scale is relative, and the sum of all values adds up to 1 [Color figure can be viewed at wileyonlinelibrary.com]

clues for which variables to explore. Figure 6 clearly shows that *Training Performance* and *Time* warrant further exploration using partial dependence plots

3.4.2 | One-way partial dependence plots and individual conditional expectation plots

One-way partial dependence plots (PDPs) show how a model's predicted outcome varies in response to changes in a single explanatory variable (conditional upon other variables). A major advantage of these plots is that they can show the shape of the relationship between the variable and the outcome. Intuitively, the partial dependence function at a particular point on the *x*-axis represents the average prediction if all data points had that *x* value.

Yet taking an average can hide heterogeneous effects among individuals. Instead, we can plot the predicted outcome's dependence on an explanatory variable separately for each individual in the data. This method is known as the individual conditional expectation (ICE) plot (Goldstein, Kapelner, Bleich, & Pitkin, 2015). Each line of an ICE plot shows what the model predicts for an individual if we changed the values of one particular variable. The PDP is simply the average of all the lines of an ICE plot. ICE plots are particularly useful tool for detecting when a variable's effect on the outcome is highly interdependent with other variables. When individual ICE lines are parallel this signifies that there are no complex interdependent relationships with that variable in the model.

Figure 7 displays the overall PDP and individual ICE lines as a function of a few select variables in our random forest model. We also plot a baseline logistic regression model side-by-side for comparison. This model included each of our explanatory variables plus *Time* fixed effects. Figure 7 demonstrates how, by construction, the predicted outcome has a linear relationship with each variable (except *Time* fixed effects) in the logistic regression.

In contrast, the random forest model reveals interesting nonlinear relationships. In this model, *Training Performance* was unrelated to probability of turnover, except for a sharp discontinuous jump for scores lower than 4.0. In the *Time* plot, there are a few ICE lines that have drastically higher probability of turnover at around 6 months. This hints

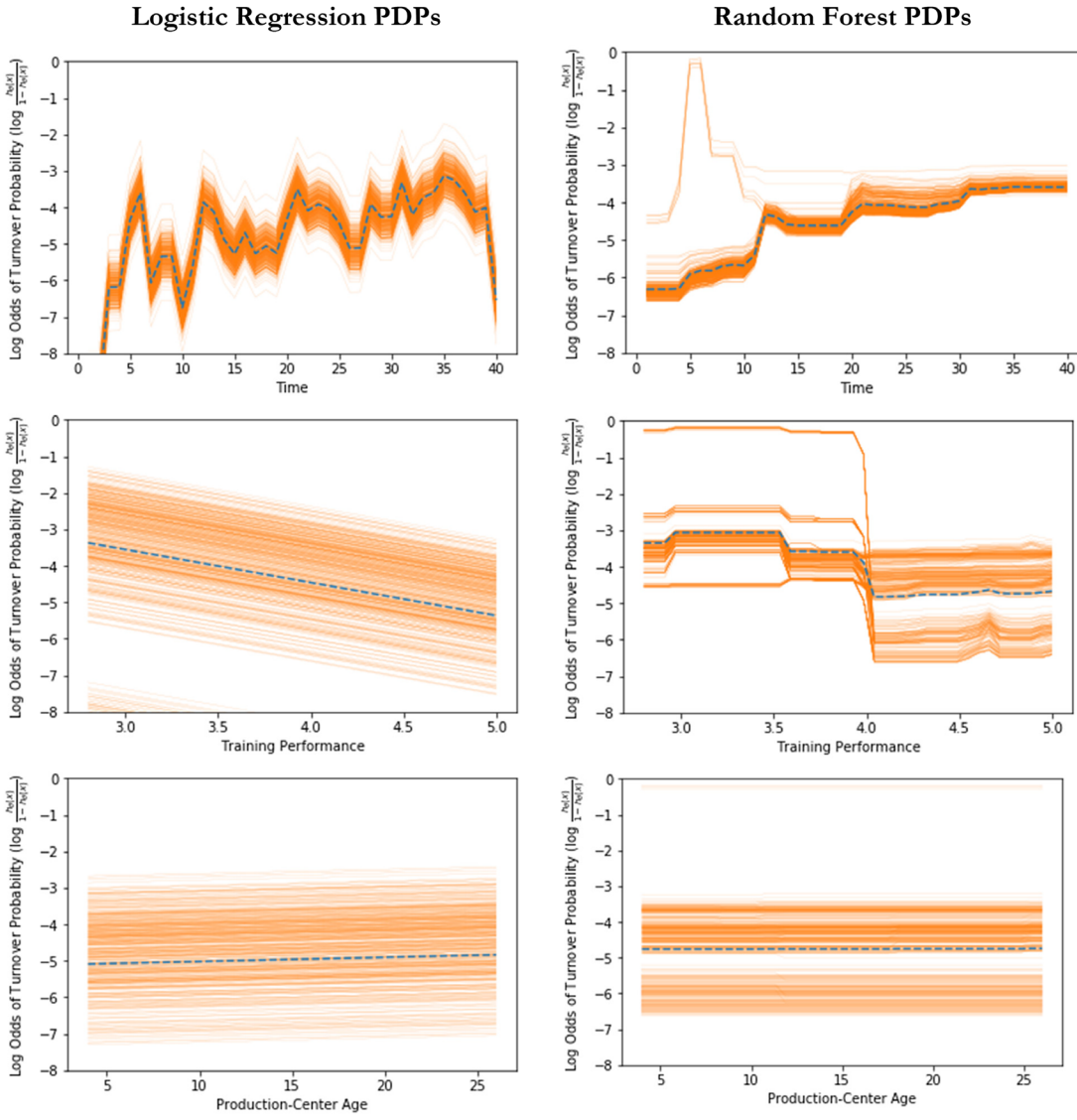

FIGURE 7 One-way partial dependence plots. *Note*: This figure overlays a one-way partial dependence (PDP) plot and individual conditional expectation (ICE) lines. The dependent variable is employee turnover ($y = 1$ if employee left in that period). Logistic regression predictions appear in the left-hand plots; random forest predictions appear in the right-hand plots. Each vertical axis is on the same scale, with units as the log of the odds ratio of the predicted probability of turnover. The ICE lines were generated by randomly selecting 500 samples from the full dataset and, for each sample, predicting the outcome using 40 values of the variable across the entire variable range while holding all other variable values fixed. The predictions from each sample are represented by an orange (solid) line across the entire range. The result is a distribution of 500 orange ICE lines, one for each sampled observation. Each plot also shows the average of the ICE lines (the overall PDP) as the dotted blue line [Color figure can be viewed at wileyonlinelibrary.com]

at heterogeneous effects for *Time* that are interdependent with the values of other explanatory variables. We can use two-way partial dependence plots to explore these interdependencies.

3.4.3 | Two-way partial dependence plots

To explore nonlinear interactions between two explanatory variables' effect on the predicted outcome, we can use two-way partial dependence plots. The intuition is the same as one-way PDPs. We average the model's predicted outcomes for each combination of values between *two* variables for each individual (holding other variable values constant). The advantage of this approach is it is easy to see interactions between explanatory variables. The disadvantage is we cannot see whether these effects are heterogeneous between individuals (as in the ICE plots).

Figure 8 displays the predicted probability of employee turnover for each ML model as a function of *Training Performance* and *Time*. Again, for comparison, we include a baseline logistic regression that includes each explanatory variable and *Time* fixed effects. We discover several interesting patterns. In the decision tree model plot, a conspicuous line separates *Training Performance* ≤3.995 from *Time* ≤6.5 (exactly as in Figure 3). The random forest model offers similar insights—that the probability of turnover tends to increase over time, and is much

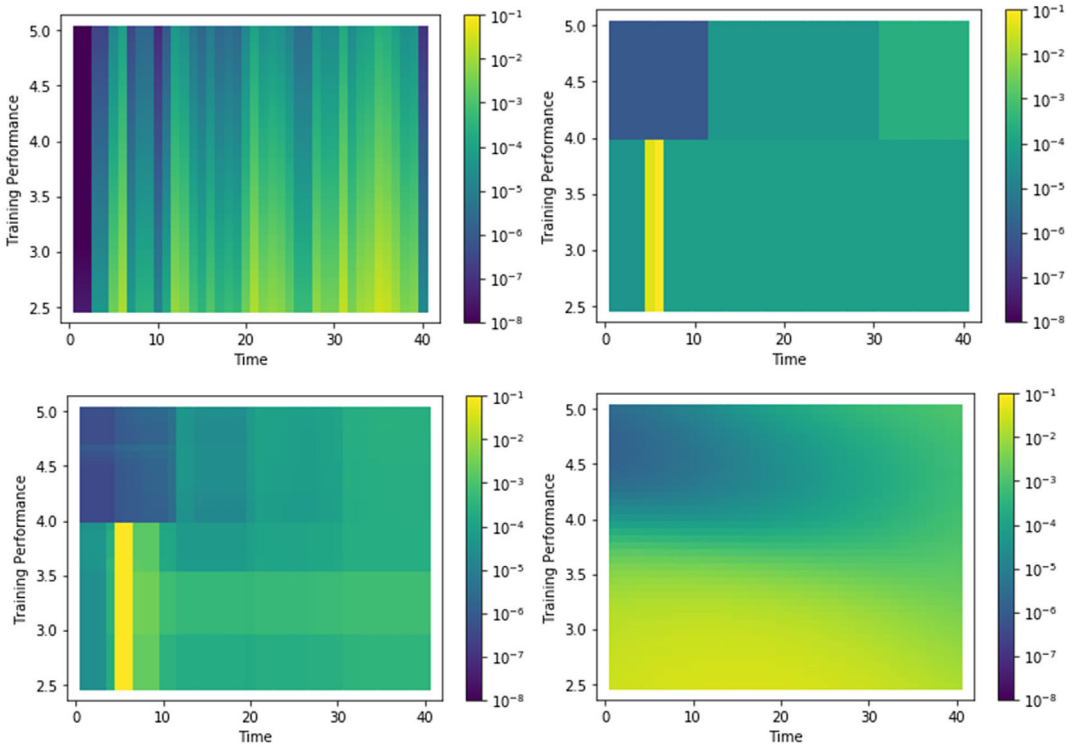

FIGURE 8 Two-way partial dependence plots. *Note:* These two-way partial dependence plots represent the probability of turnover predicted by each model along the dimensions *Training Performance* and *Time*. Higher probabilities of turnover are represented in yellow; the lowest probabilities are represented in dark blue. All plots are on the same scale. For each observation in the dataset, we used our models to predict the probability of turnover for each point represented by each combination of 40 evenly spaced values of both *Training Performance* and *Time* (resulting in a grid of 1,600 points total). The final estimated probability for each point on the grid is the average estimated probability across all observations for that point on the grid. Two-way PDP's of other variables did not reveal meaningful interactions or nonlinearities [Color figure can be viewed at wileyonlinelibrary.com]

higher for those with a training score below about 4, especially around 6 months. The global negative effect of training performance on turnover estimated by the logistic model (Figure 7) must have been driven by the narrow yellow strip of employees with training scores below 4 at about month 6.

3.5 | Comparing ML models to a baseline regression model

How does a baseline regression model fare discovering patterns from these data? A striking insight from Figure 8 is how poorly the logistic regression modeled the probability of turnover as a function of *Training Performance* and *Time*. The other models all predicted dramatically higher turnover probabilities for employees with low training scores during their first months on the job. The logistic regression model, however, can only yield a linear fit as constrained by the linear functional form we specified for the model. It can only tell us that on average those with lower *Training Performance* tend to have a higher probability of turnover (with constants added for each time interval from the *Time* fixed effects).

In fact, this negative relationship only exists for a very small fraction of the sample (the 40 out of 833 who had training scores below 4 in the training sample). The relationship between *Training Performance* and probability of turnover appears to be *positive* for the majority of the population. The logistic regression fails to capture important nuances because *Training Performance* was not explicitly modeled as a function of *Time* in the specified model. This finding demonstrates the potential cost of naively estimating global linear relationships when there are many unknowns in the data.

Of course, it is possible to use the logistic regression to test nonlinear or interactive relationships by adding transformed variables to the model. For example, we may add a quadratic term to test a U-shaped relationship, or multiply two variables to test an interaction effect. But, which terms should be included? Researchers informed by theory may model such complexities based on known relationships. However, it is often unknown *a priori* how best to model each variable, and the number of combinatorial possibilities increases rapidly as more variables are considered. ML presents a solution to the problem of knowing how to model the data.

4 | AVOIDING COMMON MISINTERPRETATION PITFALLS

Like any tool, ML can be used well or used poorly. In addition to the guidance we have given throughout the article, it is worth explicitly stating a few common misinterpretation pitfalls. Many of these pitfalls are not inherent properties of ML, but rather result from misinterpretation of ML by its users and audiences.

Confusing correlation with causation is perhaps the biggest potential problem of using ML. The patterns uncovered by the algorithm should be considered merely correlational until convincingly proven to be causal, using other means. This article advocates for ML to explore, not test, relationships in data. That said, there are certain circumstances under which we can use PDP and other tools to causally interpret ML models (Zhao & Hastie, 2019).[16]

[16]Specifically, Zhao and Hastie (2019) propose that a partial dependence plot captures the causal effect if it can be shown that the "backdoor criterion" is met. That is, adjusting for all factors that influence both X and y allows a causal interpretation of PDPs.

Algorithmic bias Data can be inherently biased,[17] for example when the data is selected based on convenience, or suffers from the feedback loop of prior human biases (Cowgill & Tucker, 2019). This can also occur when the data suffer from "input incompleteness"—gaps in the data input related to incentives of human agents, that lead to biased prediction (Choudhury, Starr, & Agarwal, 2018). Effective ML users should be familiar with the data-generating process and research context to avoid biased interpretations. For an excellent review of how to address issues of algorithmic bias, see Cowgill and Tucker (2019).

Over-interpreting variable weights of ML models is also a common mistake. The effects of variables in ML models may heavily depend on other variables in the model. In the wrong hands, negative weight on a variable in an ML model can have "enormous, undeserved rhetorical heft" (Cowgill & Tucker, 2019; p. 43). For example, Amazon was ridiculed for using an algorithm that included a negative weight for graduates of two all-women's colleges. Yet it was unclear how they were weighted relative to other women's or men's colleges.[18]

Cherry picking or data dredging Data dredging is the practice of cherry-picking interesting relationships from a large set of variables (Selvin & Stuart, 1966). Data dredgers try to draw attention to the cherry-picked relationships without acknowledging the process by which they were found. This can easily lead to identifying spurious relationships.

Multi-collinearity arises when variables in a model are highly correlated. For example, when many highly correlated variables are included in a LASSO regression, the model structure will be highly unstable (Mullainathan & Spiess, 2017). Highly correlated variables contribute redundant information to a model, so algorithms may randomly assign weight in the model to one variable or another. In our data, for example, if the employees' *Verbal Score* had been highly correlated with *Training Performance*, some models might have indicated a large effect of *Verbal Score* on turnover and none in *Training Performance*. The problem of multicollinearity becomes more serious with more variables and more highly correlated variables. To mitigate this problem, it can help to include a table of correlations between variables (which we do in Supporting Information S3). It also helps to demonstrate that patterns are robust across models built by multiple different algorithms.

p-Hacking In general, p-hacking refers to the practice of adjusting models until they statistically confirm some desired effect. Relative to traditional methods, ML is less likely to be susceptible to p-hacking because it has built-in safeguards—the validation and holdout test sets. After all, p-hacking is essentially the same as overfitting the training sample, and any model that is overfit on the training data will perform poorly on the validation and/or holdout test set. Comparing the training performance to the validation data and/or holdout test set informs us how well the model generalizes to out-of-sample data. Models that have not been overfit will have similar error for training, validation, and holdout test samples. For this reason, we recommend that reviewers always require these comparisons. It is still possible that a researcher would select among a handful of different models which perform equally well on the holdout test set. Therefore, we also suggest triangulating on an underlying model by presenting models from multiple ML algorithms.

HARKing Lastly, we caution against testing ML-identified patterns using the same dataset as though they were pre-specified hypotheses. This would be a form of hypothesizing after the results are known (HARKing), which is a violation of the assumptions of deductive hypothesis testing (Kerr, 1998). Instead, newly generated hypotheses should be tested on new data with exogenous variation to prove causal claims.

[17]Note that "algorithmic bias" refers to biased data—not the same as balancing the bias-variance tradeoff.

[18]"Amazon scraps secret AI recruiting tool that showed bias against women," Reuters 2018, as quoted in Cowgill and Tucker (2019).

5 | DISCUSSION AND CONCLUSION

This article began with two goals: (a) to demonstrate the application of supervised machine learning methods for discovering robust patterns in quantitative data, and (b) to provide guidance on evaluating research that uses such methods. Discovering new and robust empirical patterns can help management scholars by acting as an "observation" for engaging in abductive or inductive research. These patterns can later be deductively validated using traditional causal inference techniques. Pattern discovery can also be used in post hoc analysis of traditional regression results to detect patterns that may have gone unnoticed.

We are not the first to suggest that ML methods can be used for pattern discovery in research. In the physical sciences, researchers have used ML methods to uncover underlying relationships in physical phenomena (Hirsh, Brunton, & Kutz, 2018; Rudy, Alla, Brunton, & Kutz, 2018; Ruff et al., 2017). In the social sciences, researchers have evangelized what they call "forensic social science" (Goldberg, 2015; McFarland, Lewis, & Goldberg, 2016). They argue that rather than testing variations of prespecified hypotheses, ML can help us make new discoveries using digital trace data. Management scholars are also beginning to promote the idea that ML can help inductively develop a new theory (Puranam, Shrestha, He, & von Krogh, 2020).

Yet to date, there has been lack of practical guidance for transparently and effectively applying these methods in our fields. Given the guidance in this article, we are excited about the potential future of ML methods. One exciting future application is the use of ML as new tools to aid researchers in theorizing from quantitative data. This brings quantitative empirical researchers closer to the tradition of grounded theory in which researchers theorize models based on patterns in data (Bamberger & Ang, 2016; Eisenhardt, 1989; Glaser & Strauss, 1967). In our view, the ML methods do not build theory itself—rather they represent observational tools that the researcher can use to build theory. For example, a recent paper successfully pairs ML with qualitative case studies to develop new theory about optimal revenue models in iOS apps (Tidhar & Eisenhardt, 2020).

As an increasing number of management scholars use these methods for pattern discovery in their research, it is important to evaluate the choices made by researchers in using these methods. To reiterate, the implementation of ML methods for pattern discovery involves human agency and the researcher has to make important choices relative to several actions: (a) selecting data, (b) selecting algorithms, (c) setting hyper-parameters, (d) splitting training and validation data, (e) preprocessing, and (f) selecting metrics to measure predictive performance of the models. We provide guidance on all these choices. Additionally, given that ML methods do not generate the familiar coefficients with confidence intervals, we illustrate tools such as partial dependence plots, plots of training and validation loss and the ROC curve that could be used both by the researcher and the reader to visualize, interpret and evaluate the robustness of patterns discovered.

Throughout this article, we have highlighted that because ML algorithms do not produce statistically consistent coefficient estimates or reliable standard errors, they should not be used for traditional deductive hypothesis testing (Mullainathan & Spiess, 2017). Rather, the use of ML models to discover patterns should be seen as an exploratory exercise. Thus, we present ML as a complement to, not a substitute for, traditional econometric methods. Misunderstanding this point can lead to severe misinterpretation. We also emphasized caution against common pitfalls such as being subject to biases in the data, data dredging, p-hacking, and HARKing.

In conclusion, we demonstrate how ML methods can be a helpful exploratory tool to identify robust patterns in quantitative data. These patterns can be helpful for researchers engaged in data-driven abductive or inductive scientific discovery. We provide tools for implementing

and visualizing ML analyses, and guidance for interpretation and evaluation. We also provide detailed code to help future researchers. We look forward to the potential of ML as a methodological framework in future empirical management research.

ACKNOWLEDGEMENTS

The authors thank Kathy Eisenhardt, Connie Helfat, Dan Levinthal, Rory McDonald, Joe Mahoney, Evan Starr, Ron Tidhar, participants at the 2018 Academy of Management and the 2018 Strategy Science conference held at the Wharton School and two anonymous reviewers for helpful comments on a prior draft. All errors remain ours.

ORCID

Prithwiraj Choudhury https://orcid.org/0000-0002-5936-2079
Ryan T. Allen https://orcid.org/0000-0002-8227-8844
Michael G. Endres https://orcid.org/0000-0002-1411-360X

REFERENCES

Athey, S., & Imbens, G. W. (2015). Machine learning methods for estimating heterogeneous causal effects. *Stat*, *1050*(5), 1–26.

Autor, D. H. (2015). Why are there still so many jobs? The history and future of workplace automation. *Journal of Economic Perspectives*, *29*(3), 3–30.

Bamberger, P., & Ang, S. (2016). The quantitative discovery: What is it and how to get it published. *Academy of Management Discoveries 2*(1), 1–6.

Campbell, B. A., Ganco, M., Franco, A. M., & Agarwal, R. (2012). Who leaves, where to, and why worry? Employee mobility, entrepreneurship and effects on source firm performance. *Strategic Management Journal*, *33*(1), 65–87.

Carnahan, S., Kryscynski, D., & Olson, D. (2017). When does corporate social responsibility reduce employee turnover? Evidence from attorneys before and after 9/11. *Academy of Management Journal*, *60*(5), 1932–1962.

Choudhury, P., Starr E., & Agarwal, R. (2020). Machine learning and human capital complementarities: Experimental evidence on bias mitigation. *Strategic Management Journal*, *41*(8), 1381–1411. http://dx.doi.org/10.1002/smj.3152.

Choudhury, P., Wang, D., Carlson, N. A., & Khanna, T. (2019). Machine learning approaches to facial and text analysis: Discovering CEO oral communication styles. *Strategic Management Journal*, *40*(11), 1705–1732.

Cowgill, B., & Tucker, C. E. (2019). Economics, fairness and algorithmic bias. *SSRN Electronic Journal*, http://dx.doi.org/10.2139/ssrn.3361280.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review*, *14*(4), 532–550.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Furman, J. L., & Teodoridis, F. (2020). Automation, research technology, and researchers' trajectories: Evidence from computer science and electrical engineering. *Organization Science*, *31*(2), 330–354.

Glaser, B. S., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. New Brunswick, NJ: AldineTransaction.

Goldberg, A. (2015). In defense of forensic social science. *Big Data & Society*, *2*(2), 2053951715601145.

Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, *24*(1), 44–65.

Gross, D. (2018). Creativity under fire: The effects of competition on creative production (NBER Working Paper No. 25057). Retrieved from National Bureau of Economic Research website: https://www.nber.org/papers/w25057.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction*. Berlin: Springer Science & Business Media.

Hirsh, S. M., Brunton, B. W., & Kutz, J. N. (2018). *Data-driven spatiotemporal modal decomposition for time frequency analysis. Working Paper*. Seattle, WA: University of Washington Retrieved from https://arxiv.org/pdf/1806.08739.pdf

Jovanovic, B. (1979). Job matching and the theory of turnover. *Journal of Political Economy*, *87*(5, Part 1), 972–990.

Kaplan, S., & Vakili, K. (2015). The double-edged sword of recombination in breakthrough innovation. *Strategic Management Journal*, *36*(10), 1435–1457.

Kerr, N. L. (1998). HARKing: Hypothesizing after the results are known. *Personality and Social Psychology Review*, *2*(3), 196–217.

Mantree, S., & Ketokivi, M. (2013). Reasoning in organization science. *Academy of Management Review*, *38*(1), 70–89.

McFarland, D. A., Lewis, K., & Goldberg, A. (2016). Sociology in the era of big data: The ascent of forensic social science. *The American Sociologist*, *47*(1), 12–35.

Menon, A., Choi, J., & Tabakovic, H. (2018). What you say your strategy is and why it matters: Natural language processing of unstructured text. *Academy of Management Proceedings*, *2018*, 18319.

Microsoft. (2019). How to choose algorithms. Retrieved from https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

Mullainathan, S., & Spiess, J. (2017). Machine learning: An applied econometric approach. *Journal of Economic Perspectives*, *31*(2), 87–106.

Puranam, P., Shrestha, Y. R., He, V. F., & von Krogh, G. (2020). Algorithm supported induction for building theory: How can we use prediction models to theorize? *Organization Science*.

Rudy, S., Alla, A., Brunton, S. L., & Kutz, J. N. (2018). *Data-driven identification of parametric partial differential equations*. Working Paper. Retrieved from https://arxiv.org/abs/1806.00732

Ruff, C. T., Lacoste, A., Nordio, F., Fanola, C. L., Silverman, M. G., Argentinis, E., ... Sabatine, S. (2017). Classification of cardiovascular proteins involved in coronary atherosclerosis and heart failure using Watson's cognitive computing technology. *Circulation*, *136*(S1), A16678.

Scikit Learn. (2018). Choosing the right estimator. Retrieved from https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Selvin, H. C., & Stuart, A. (1966). Data-dredging procedures in survey analysis. *The American Statistician*, *20*(3), 20–23.

Suddaby, R. (2006). From the editors: What grounded theory is not. *Academy of Management Journal*, *49*(4), 633–642.

Tidhar, R., & Eisenhardt, K. M. (2020). Get rich or die trying... finding revenue model fit using machine learning and multiple cases. *Strategic Management Journal*, *41*, 1245–1273.

Zhao, Q., & Hastie, T. (2019). Causal interpretations of black-box models. *Journal of Business & Economic Statistics*, 1–10. http://dx.doi.org/10.1080/07350015.2019.1624293.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Choudhury P, Allen RT, Endres MG. Machine learning for pattern discovery in management research. *Strat Mgmt J*. 2021;42:30–57. https://doi.org/10.1002/smj.3215