Machine Learning

Foundation Entries

SAGE Research Methods Foundations

By: Jennie E. Brand, Bernard Koch & Jiahui Xu
Published:2020
Length: 10,000 Words
DOI: http://dx.doi.org/9781529749212
Methods: Machine Learning
Online ISBN: 9781529749212
Disciplines: Economics, Education, Social Policy and Public Policy, Sociology, Computer Science
Access Date: May 7, 2021
Publishing Company: SAGE Publications Ltd
City: London

Abstract

Machine learning is a statistical and computational approach to extracting important patterns and trends in data. This entry is an overview of machine learning methods for social science research. It covers supervised learning methods including generalized linear models, support vector machines, naive Bayes, *k*-nearest neighbor, artificial neural networks and deep learning, decision trees, and ensemble methods. It also notes several important considerations relevant to supervised learning algorithms including the use of training and test data and cross-validation, loss optimization and evaluation metrics, bias-variance trade-off, and overfitting and regularization strategies. The entry also covers unsupervised learning methods, including *k*-means clustering, hierarchical clustering, network community detection, principal component analysis, and *t*-distributed stochastic neighbor embedding. A section on text analysis incorporates supervised and unsupervised learning of documents and neural networks. The entry provides an overview of new developments at the intersection of machine learning methods and causal inference. Key limitations and considerations for adopting these methods in empirical social science research conclude the entry.

Introduction

Machine learning is a computational and statistical approach to extracting patterns and trends from data (Maini & Sabri, 2017). A machine learning algorithm is also defined as a computer program that learns from experience with data with respect to some class of tasks and improves its performance with greater experience (Mitchell, 1997). Machine learning has widespread applications to many scientific fields and has gained a considerable amount of attention and use in recent years in the era of high-speed computing and big data. As the tasks of statistical sciences have expanded in scope and complexity, machine learning has quickly permeated many scientific fields from statistics and computer science to the social sciences.

There is some debate, particularly in statistics and the social sciences, about the degree of difference between machine learning and classical statistics. Indeed, some scholars describe the field as statistical learning. Machine learning methods have overlapped with methods used in applied statistics and social sciences for many decades, although it is only recently that these fields have adopted machine learning methods in earnest. At the same time, statisticians and social scientists have made considerable progress bridging machine learning methods with areas central to statistics and social science, like causal inference (Athey, 2018).

Machine learning algorithms can be categorized into supervised, unsupervised, and reinforcement learning. Supervised learning algorithms learn from training data that contains both inputs (also called independent variables, covariates, and features) and labeled outcomes (also called targets, dependent variables, outcomes, and responses). Supervised learning tasks involving a categorical outcome variable are referred to as classification tasks; learning tasks involving a continuous outcome are called regression tasks. Some common supervised learning approaches include generalized linear models (GLMS; e.g. logistic regression), kernel methods (e.g., support vector machines [SVMs]), decision tree (DT)-based methods, and artificial neural networks (i.e., deep learning).

In contrast to supervised learning, unsupervised learning algorithms build models with data only on the features, without labels on the outcomes. These algorithms are usually used for the purpose of clustering (e.g., discovering smaller groups within the data) or dimensionality reduction (e.g., reducing the number of features or covariates to a smaller, more informative number of latent features). The algorithms build models with data only on the input features, without labels on the outputs. Some common unsupervised learning techniques include *k*-means clustering, hierarchical clustering, principle components analysis, and t-distributed stochastic neighbor embedding (t-SNE). These techniques may be an intermediate data analysis step in the construction of covariates or outcome variables for supervised learning.

While supervised learning uses data with outcomes (i.e., labels are provided), and unsupervised learning does not (i.e., no labels are provided), an additional approach, reinforcing learning, lies in the middle. Reinforcement learning, sometimes called semisupervised learning, is concerned with how agents should take actions in environments to maximize reward. It is modeled as a Markov decision process. Reinforcement learning has been used in robotics, industrial automation, gaming, digital advertising, and optimal policy design. To correspond to use of these approaches in the social sciences, this entry devotes considerable time to supervised learning, less time to unsupervised learning, and no additional time to reinforcement learning.

A Brief History of Machine Learning: From Computer Science to Computational Social Science

Machine learning is a subset of artificial intelligence (AI). Since Alan Turing first developed the concept of the Turing Test in 1950 and considered the question of whether machines can "think," researchers in computer science have been drawn to the study of AI. Following the Dartmouth conferences of 1956, known as the "birth of AI" (Crevier, 1993), Arthur Samuel coined the term *machine learning* to describe the way in which machines learn from data. Machine learning methods were later noted to bear similarity to statistical science. Machine learning grew in the 1990s as a means by which to apply statistics and probability theory to effectively utilize increasing digitized information.

The term *data science* unifies the fields of machine learning, statistics, and data analysis. Data science is a multidisciplinary field, drawing on mathematics, statistics, computer and information science, that uses scientific methods and algorithms to generate knowledge from data. The term was used by Peter Naur in 1960 to refer to computer science. It was not until recently, however, that data science became a widely used phrase and *data scientist* an increasingly advertised position. Many universities are either instituting or rebranding data science divisions, departments, centers, and labs. Another emerging phrase in the social sciences is *computational social science*. Computational social science is data science applied to social science applications.

The transformative impact of machine learning on social science research and policy is well underway. Machine learning gives social science researchers tools to leverage the large, rich, and often unstructured data that have become increasingly available. While these data benefit from being large relative to traditional social science surveys, they come with their own unique set of challenges such as population representativeness and quality of measures. With increasing technological and statistical progress, machine learning methods are a powerful tool for social scientists to leverage these new and existing data. This does not, however, negate the importance of human interpretative ability and social theory in social science research (Pearl, 2018).

Supervised Learning

Supervised learning algorithms learn to identify labeled outcomes from training data that are then used to predict the outcomes of new unseen test data. In training, the algorithm observes both the covariates and the outcomes. For example, to teach a supervised algorithm how to distinguish between Democrats and Republicans, a researcher might provide a training sample of individuals with a set of covariates and an outcome labeled as Democrat and Republican. After training, the algorithm would predict the party affiliations of individuals in a new data set. As noted earlier, tasks in which the outcome variable is categorical are referred to as classification tasks, and tasks in which the outcome is continuous are called regression tasks. Many algorithms can be adapted for both regression and classification tasks. For example, a researcher might use logistic regression to predict the probability that individuals attend college (regression) or round this probability at a decision boundary to predict whether or not individuals attend college (classification).

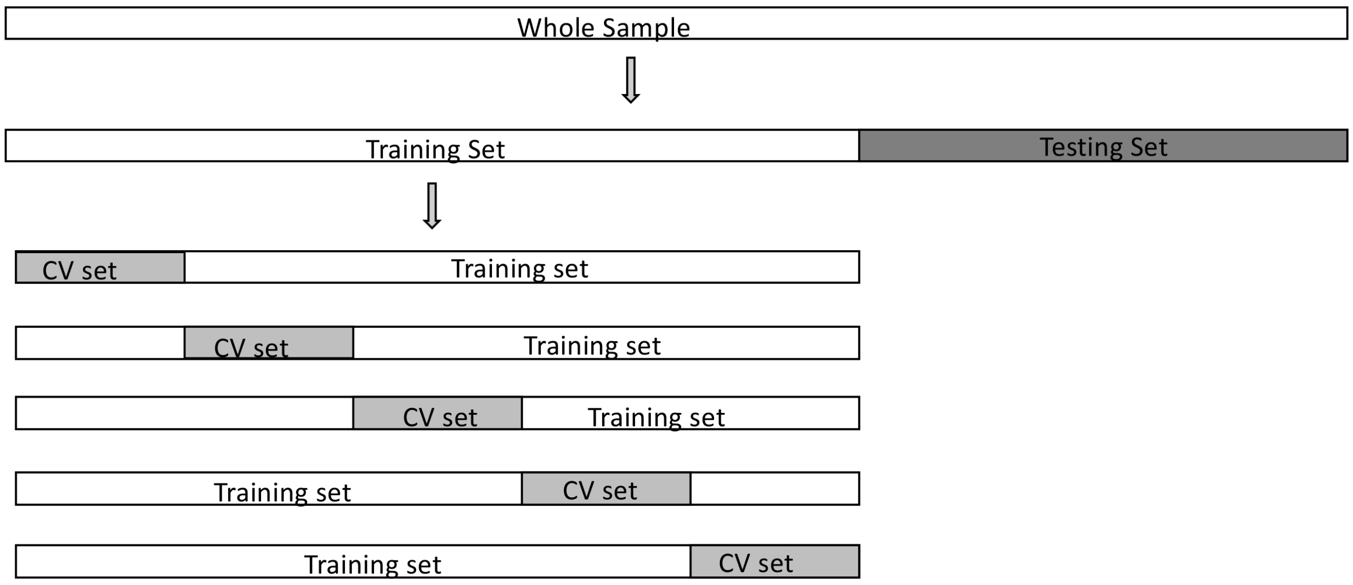
There are a wide range of supervised learning approaches and no single algorithm performs best in all applications. Each has limitations and trade-offs; there is no free lunch (Wolpert & Macready, 1997). Some common supervised learning approaches include (a) generalized linear models (e.g., ordinary least squares [OLS] and logistic regression); (b) kernel methods (e.g., SVMs); (c) naive Bayes classifiers; (d) *k* nearest neighbors (kNN); (e) DT methods (e.g., classification and regression trees [CART], random forests, and gradient boosting machines); and (f) artificial neural networks (i.e., deep learning). The rest of this section proceeds as follows. First, it describes the general workflow of supervised learning, with a focus first on regression tasks. Second, it explains how supervised regression algorithms are trained and introduces the bias-variance trade-off, as well as the concept of regularization. Finally, it discusses classification tasks and the precision-recall trade-off. It then briefly introduces several different classes of supervised learning algorithms.

Training, Cross-Validation, and Testing Data

The first step in a machine learning analysis is to divide the data into a labeled training set for optimizing the model parameters and a testing set of unlabeled data for which we wish to predict outcomes. Testing data should not be touched until the final parameters of the model have been chosen; the testing data will be used only once. Partitioning the data in this way is important for learning general patterns and not overfitting training data. Overfitting occurs when a model is fit too closely to particular data points, yielding a complex model that accurately explains idiosyncratic patterns in those data and does not generalize reliably to new data.

Researchers reserve a portion of the training data—the cross-validation set—to evaluate how well the algorithm will perform out-of-sample prediction before applying the model to the testing data. *K*-fold cross-validation is a resampling procedure that is used to evaluate how a model is expected to perform on unseen (i.e., test) data. In *k*-fold cross-validation, researchers divide the data into *k* groups and build a model on $k - 1$ groups, while the remaining group is used for validating. This subsampling process is repeated until all the *k* groups have been used as the test (or validation) set. Figure 1 is an example of *k*-fold cross-validation. The sample is partitioned into a training set and a testing set, and the training set is subsequently partitioned into a cross-validation set to tune hyperparameters. Higher values of *k* are common (e.g., $k = 10$).

Figure 1. Supervised learning workflow with k-fold cross-validation.



K-folds cross-validation strategy, where $k = 5$. Cross-validation is a resampling procedure used to evaluate machine learning models on a sample. The procedure has a single parameter (k) that refers to the number of groups that a given data sample is split into.

Training data can be used to improve model performance by tuning hyperparameters. Hyperparameters are model parameters whose values are set before the learning process begins, and that are adjusted by the researcher. Values of other parameters are derived in the training process. Simple algorithms, like

OLS regression, do not require hyperparameters. Hyperparameter tuning is the process of choosing a set of optimal hyperparameters for a learning algorithm. To sample different hyperparameter values during cross-validation, researchers often use grid search (i.e., explore all possibilities) or random search (i.e., explore random possibilities); more sophisticated strategies such Bayesian optimization are also possible. Grid search works by searching exhaustively through a specified set of hyperparameters. It is guaranteed to identify the optimal parameters, yet is time-consuming. Random search searches randomly, rather than exhaustively, and in so doing is less time-consuming yet also less likely to find the optimal combinations of hyperparameters.

Loss Optimization and Evaluation Metrics

During training, supervised learning algorithms optimize in-sample performance with respect to a loss (sometimes called objective or cost) function. After training, researchers use evaluation metrics to assess out-of-sample predictive performance of the model. Although most supervised learning algorithms can be used for both regression (continuous outcome) and classification (categorical outcome), regression and classification tasks typically have different loss optimization functions and evaluation metrics. Moreover, although the same metrics are often used for in-sample loss and out-of-sample evaluation in regression, researchers typically use different metrics for training and evaluation in classification tasks. This section focuses on functions for loss and evaluation for regression and then for classification.

Regression Loss and Evaluation Metrics

The most common loss function for regression is the mean squared error (*MSE*), or quadratic loss function. The *MSE* measures the mean of the square of the errors between the observed value and the predicted value:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

where y_i is the observed outcome, and $\hat{y}_i$ is the predicted outcome. Sometimes the square root of this function, the root MSE (RMSE), is used for ease of interpretation. The *MSE* is always non-negative, and values closer to zero indicate better model accuracy. In regression training, the algorithm is trained to minimize *MSE* in the training data; researchers are most interested, however, in how well the model performs on the test data. The *MSE* is also the most common evaluation metric in the test data. There is no guarantee that the model with the smallest training set *MSE* will have the smallest test set *MSE*. Mean absolute error (*MAE*) is an alternative regression loss and evaluation function that is more robust to outliers and is easier to interpret than *MSE*. *MAE* weights all errors equally, while the quadratic form of *MSE* penalizes large errors more heavily. *MAE* is the sum of absolute differences between the predicted and actual value rather than the sum of squared differences:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|.$$

Although the *MSE* is by far the most common loss and evaluation metric, there may be specific tasks where a more specialized function like Huber loss, log-cosh, or quantile loss is appropriate. Beyond these functions, many of the model selection metrics that social scientists are already familiar with, such as R^2, are also useful for evaluating out-of-sample performance. R^2, or the coefficient of determination, is a statistical measure of how close the data are to the model. It is measured as the ratio of explained to total variation.

Classification Loss and Evaluation Metrics

Classification accuracy is a simple metric to evaluate classification algorithms by the percentage of correct classifications. It does not, however, completely capture the performance of a classifier. Researchers instead evaluate the performance of a classifier using precision and recall. Precision is defined as:

$$P = \text{True Positives (TP)}/\text{True Positives (TP)} + \text{False Positives (FP)},$$

and recall is defined as:

$$R = \text{True Positives (TP)}/\text{True Positives (TP)} + \text{False Negatives (FN)}.$$

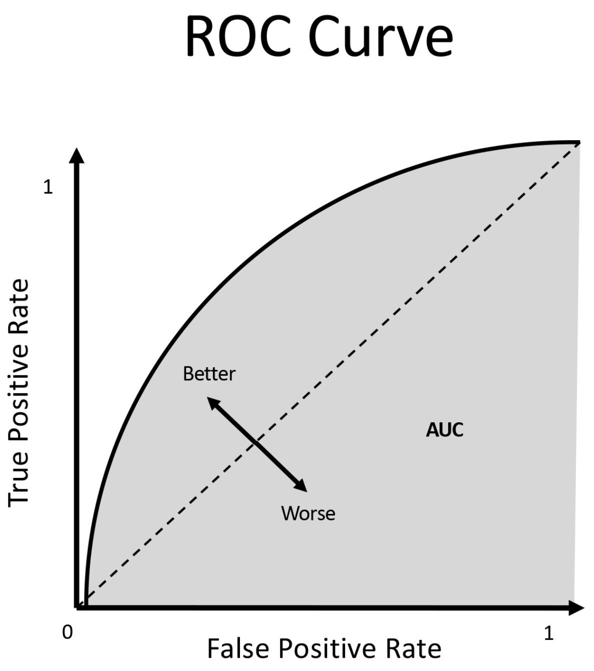
If a classifier has high precision, it will rarely misclassify a false positive as true positive. If a classifier has high recall, it will rarely miss any positives that should be true positives. Generally, there is a trade-off between precision and recall, and there may be times where a researcher favors a more conservative, high-precision, or a more sensitive, high-recall classifier.

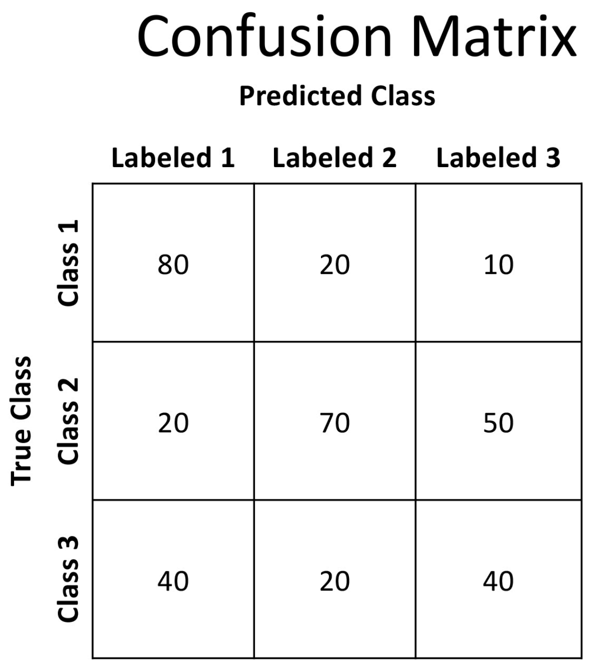
In binary classification, researchers can demonstrate the precision-recall trade-off using a receiver-operating curve (ROC) plot. Ideally, researchers want a classifier that minimizes the trade-off. The area under the ROC curve (AUC) is often used as a reductive statistic for comparing classifiers. ROC curves do not extend easily to multiclassification problems. In this case, researchers may turn to the confusion matrix, which displays the number of each true label categorized into each class. Figure 2 displays ROC and the confusion matrix. Another approach to summarizing the precision-recall trade-off is the *F*1 score:

$$F1 = 2\left[\frac{1}{\frac{1}{P}+\frac{1}{R}}\right].$$

Unlike ROC/AUC, *F*1 can be easily extended to multiclassification problems.

Figure 2. Receiver operating characteristic (ROC) curve and confusion matrix.




Confusion Matrix

Predicted Class

		Labeled 1	Labeled 2	Labeled 3
True Class	Class 1	80	20	10
	Class 2	20	70	50
	Class 3	40	20	40

Left: The ROC curve (and the area under this curve) capture the precision-recall trade-off for binary classifiers. Right: Example confusion matrix with 100 training examples in 3 classes. Correct classifications are diagonal elements and misclassifications are off-diagonal elements.

A common classification loss function is the log loss or closely related cross-entropy loss. Rather than optimizing the accuracy of classification, log loss optimizes the model's probabilistic estimates of which class the data belong to. Log loss is often preferable to classification accuracy because it considers the uncertainty of predictions. Suppose there are two possible class labels, *y* is a binary variable (0/1) indicating whether class label *c* is the correct classification for a unit *i*, and *p* is the model's predicted probability that unit *i* is class *c*, the log loss function is:

$$Log\ Loss = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot log_e(\hat{y}_i) + (1-y_i) \cdot log_e(1-\hat{y}_i)].$$

A perfectly predictive model would have a log loss of 0, and the log loss increases as the predicted probability deviates from the actual label. For example, if the true label had a value of 1.0, the log loss function slowly decreases as the predicted probability nears 1.0. Log loss can be extended to multiclass problems by summing the first term in the aforementioned equation for each class. Yet not all classifiers extend generically to multiclassification. Focal loss, KL divergence or relative entropy, exponential loss, and hinge loss (particularly for SVMs) are other possible loss functions for classification.

Bias-Variance Trade-Off

In all supervised machine learning tasks, there is a fundamental trade-off between a model's ability to achieve the optimal in-sample fit and its ability to generalize to new, out-of-sample data. In statistical learning theory, this is known as the bias-variance trade-off. Imagine you have a single datum with covariates x_i and outcome y_i and many different data sets, and you train a supervised learning model *f(x)* on each of these data sets. Error due to bias is the difference between the average (expected) predicted outcome *y* across all models/data sets and the actual value of *y*. That is, a learning algorithm is biased if it systematically incorrectly predicts *y*. The bias is thus the difference between the true values and the algorithm's prediction (in expectation) when trained on multiple data sets. Error due to variance describes the variability of the predictions of *y* across all models/data sets.

Recall the equation for the *MSE*:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = E[(y_i - \widehat{f(x)})^2].$$

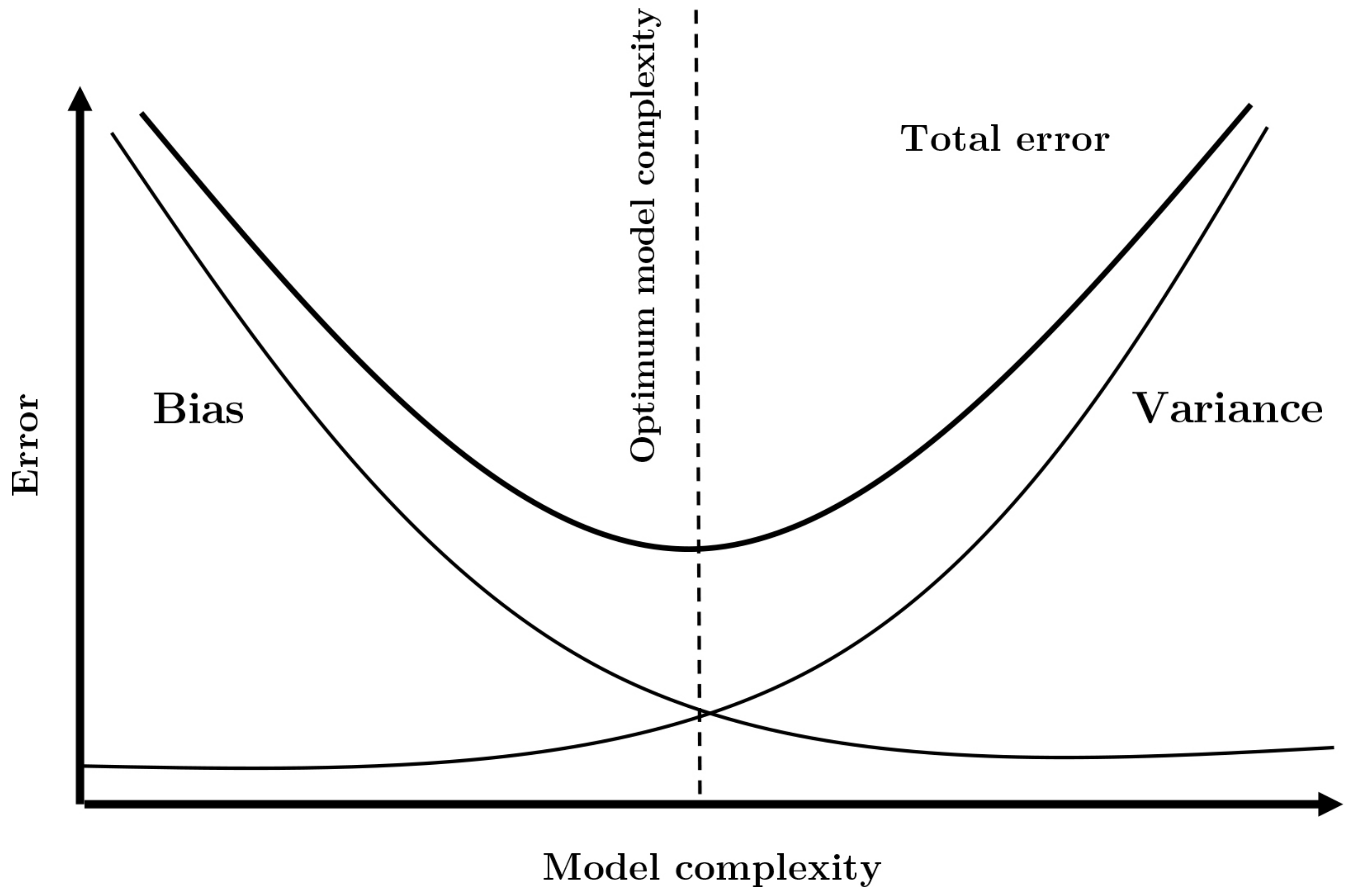
The equation for the *MSE* can be decomposed into errors in prediction as errors arising from bias and those from variance as follows:

$$MSE = \left(E[\widehat{f(x)}] - f(x)\right)^2 + E\left[\left(\widehat{f(x)} - E[\widehat{f(x)}]\right)^2\right] + \sigma_\varepsilon^2.$$

In other words, the *MSE* can be written as bias^2 + variance + irreducible error (Hastie et al., 2009). The irreducible error is the noise in the true relationship that cannot be reduced by any model. The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm, generating a bias-variance trade-off (Geman, Bienenstock, & Doursat, 1992).

Ideally, a model should be neither overly complex nor overly simple. A more flexible, complex model decreases bias but increases variance. A less complex model essentially ignores relevant information, and error due to bias increases while error due to variance decreases. Figure 3 displays this relationship. A learning algorithm must be complex enough to fit the training data and minimize bias, while simultaneously not so complex that variance is high when fit to generalization data sets. Despite a leaning toward minimizing bias in the training data, researchers should prefer an estimator with some bias to reduce variance in generalization data.

Figure 3. Bias-variance trade-off.



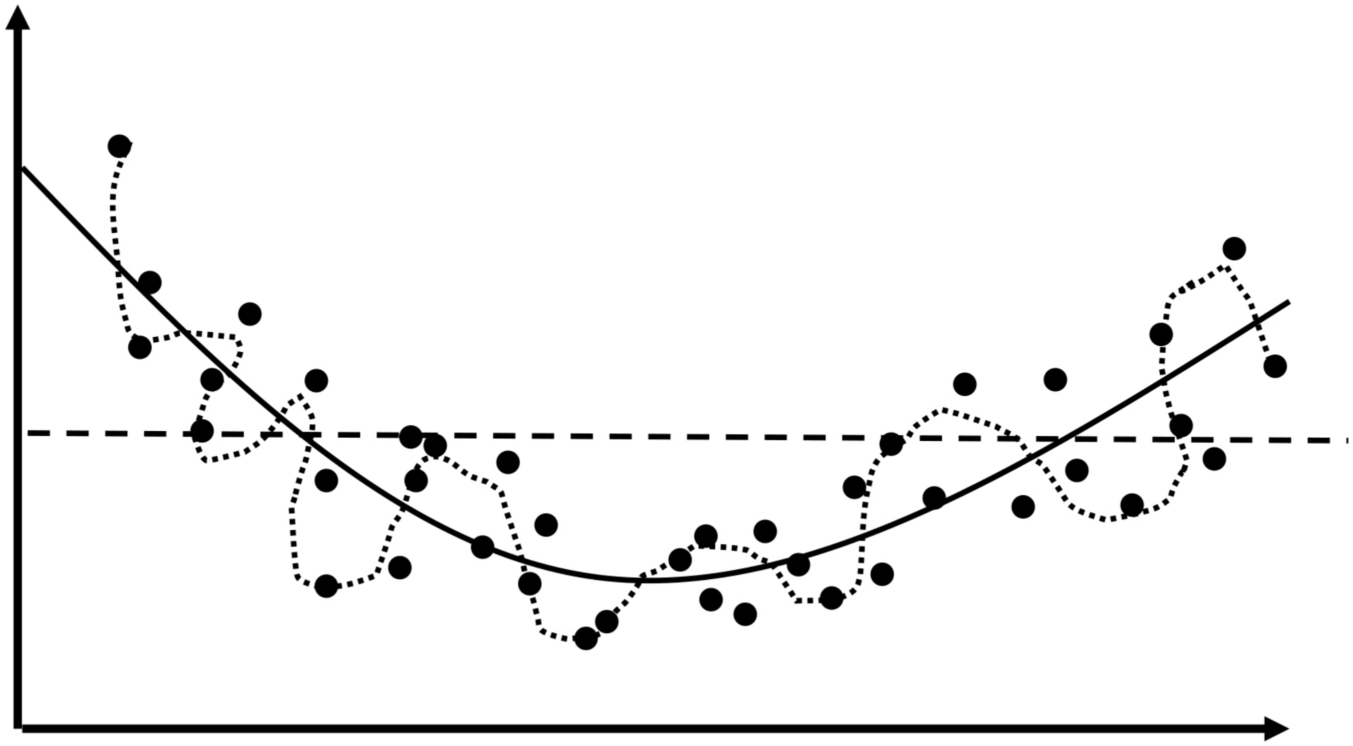
A learning algorithm must be complex enough to fit the training data and minimize bias, while simultaneously not so complex that variance is high when fit to generalization data sets.

The size of training data and the number of features influence model building capacity. If the true relationship between features and the outcome is simple, then a small amount of training data and a simple model will be sufficient to learn that function with low bias and variance. If the true function is complex, more data and a more flexible algorithm are needed to approximate the true function, and the bias-variance trade-off is more salient. Likewise, if the feature space has high dimensionality, learning may be difficult. This can be true even if the true function only depends on a small subset of those features.

Overfitting and Regularization

Overfitting occurs when a supervised learning model learns too exactly the particularities of the training data and may therefore fail to predict future observations reliably in the test data or more generally to other samples. An overfit model contains more flexibility (e.g., more parameters) than can be justified given the data and extracts noise from the data as if that noise or variation represented true model structure. An overfit model corresponds to high variance and low bias. The alternative is that the model is underfit, that is, a simple model corresponding to low variance and high bias. Variance is error resulting from sensitivity to small fluctuations in the training data. When there is high variance, the algorithm models the noise in the training data. Bias is error resulting from faulty assumptions in the model. High bias results in overlooking important relationships between the independent and dependent variables. Figure 4 depicts an example of overfitting and underfitting a regression model.

Figure 4. Example of overfitting and underfitting a training set.



Black dots represent training points. Dotted line represents a complex, overfit, low-bias but high-variance model that will not generalize well to new data. Dashed line represents an underfit linear model, high bias in the training set. Solid line represents a model that falls somewhere in between the overfit and underfit models.

Regularization encompasses a range of strategies to prevent overfitting, improve generalization, and reduce model complexity. There are many different approaches to regularization, including reducing the number of covariates using dimensionality reduction, feature selection algorithms, or domain knowledge. Shrinkage methods reduce the magnitude of parameters by applying a constraint and penalty on the complexity of the model. The two most common shrinkage penalties are L1 or least absolute shrinkage and selection operator (LASSO) regression (Tibshirani, 1996) and L2, Tikhonov, or ridge regularization. Although LASSO and ridge penalties can be applied to any continuous loss function, this section mainly focuses on their application to GLMs (e.g., OLS, logistic regression). For researchers using GLMs, the key difference between LASSO and ridge regression is that LASSO selects a subset of covariates for the final model, and effectively shrinks parameters that do not contribute to the predictivity of the model to zero, whereas ridge regression shrinks large parameters but does not allow the coefficients to be zero. LASSO therefore results in a simpler, more interpretable model.

Consider a sample of N units, one outcome y_i, k covariates, and $x_i = (x_1, x_2, \ldots, x_p)^\mathrm{T}$ is the covariate vector for the ith case. The objective of LASSO is to solve:

$$\min_{\beta_0,\beta}\left\{\frac{1}{N}\sum_{i=1}^{N}\left(y_i-\left(\beta_0+\sum_{j=1}^{k}\beta_j\,x_{ij}\right)\right)^2\right\},$$

subject to the constraint

$$\sum_{j=1}^{k}\left|\beta_j\right|\leq t,$$

where *t* is a prespecified free parameter that determines the degree of regularization. The function with the LASSO or L1 regularization can then be written as:

$$L1=\sum_{i=1}^{N}\left(y_i-\left(\beta_0+\sum_{j=1}^{k}\beta_j\,x_{ij}\right)\right)^2+\lambda\sum_{j=1}^{k}\left|\beta_j\right|,$$

where the last term is the penalty, or regularization, term. The *λ* term is a hyperparameter, which can be tuned in order to improve performance. The ridge regression or L2 regularization is associated with a different penalty term:

$$L2=\sum_{i=1}^{N}\left(y_i-\left(\beta_0+\sum_{j=1}^{k}\beta_j\,x_{ij}\right)\right)^2+\lambda\sum_{j=1}^{k}\beta_j^2,$$

where the last term is the penalty term added. L1 and L2 regularization both prevent overfitting by shrinking the parameters as the penalty term *λ* increases. If *λ* is zero, both forms revert to OLS regression. We are faced with a bias-variance trade-off in choosing the value of *λ*. As *λ* increases, the estimators increase bias and minimize variance.

Supervised Learning Algorithms

Generalized Linear Models

Generalized linear models, particularly linear and logistic regression, are among the simplest and most widely used supervised learning algorithms. Despite its name, logistic regression is mainly used for binary classification tasks by applying a threshold to the probability generated by the model (probabilities greater than the threshold are assigned to one class, and below the threshold to the other). In the logistic regression algorithm, the objective is to classify two classes, that is, $y \in \{0, 1\}$ (e.g., classifying tumors as malignant or benign). Log odds for the value labeled 1 is a linear combination of one or more discrete or covariates. The log odds can also be understood as the log of the probability the event will occur (or the output is 1) divided by the probability that it does not occur (or the output is 0):

$$logit(p) = log\frac{p}{1-p}$$

This is a linear model, as the log odds are a linear combination of the inputs. The covariates enter the logistic regression by passing through a sigmoid (*S*-shaped) function. The logistic function is given by the inverse logit.

Consider the tumor example in more detail. Suppose researchers have the features of tumors (e.g. tumor size, position). The purpose is to use these features to predict malignant tumors, which can be interpreted as probabilities from the logistic function. If the probability for one tumor is greater than 0.5 (where $x = x^*$ when the probability is 0.5), researchers can predict that the tumor is malignant. Otherwise, it is assumed to be benign. Researchers might call 0.5 the decision rule. The vertical line $x = x^*$ is a decision boundary which separates the tumors ($y = 1$) from the nontumors ($y = 0$). Clearly, the data may not, however, be separable at this decision boundary. This example could be extended to incorporate high-dimensional features as well as multiclass classification, such as filtering emails to various categories.

Support Vector Machines

Like logistic regression, SVMs learn a binary decision boundary between two classes (Cortes & Vapnik, 1995). However, they use a geometric approach rather than probabilistic approach. If the feature space has *k* dimensions, SVMs will learn a $k - 1$-dimensional hyperplane (a two-dimensional hyperplane is a line, a three-dimensional hyperplane is a plane, etc.) to divide the two classes, such that the hyperplane is maximally distant from the nearest training points in each class (i.e., maximizes the “margin” between the hyperplane and each class). Finding the hyperplane that meets the criteria is only computationally feasible by using the *kernel trick*, which allows the algorithm to compute dot products (i.e., distances) between points in high-dimensional spaces simply by transforming them using a kernel function (James et al., 2017). Even with $k - 1$ dimensions of flexibility, this *hard margin* approach can lead to overfitting of outliers so *soft margin* SVMs allow a few training units to be misclassified, thus decreasing variance.

Naive Bayes Classifiers

Naive Bayes classifiers, also known as probabilistic classifiers, constitute a simple method for classifying data. Naive Bayes is a conditional probability model that applies Bayes theorem:

$$P(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

where $P\left(C_k|x\right)$ is the probability of class C given a vector of features **x**. It is naive in that all the features are assumed to be independent. The naive Bayes classifier works by first calculating the probabilities for each class given the input features. For example, suppose we want to filter e-mails. An e-mail could be classified as spam or nonspam based on the text of the email (i.e., the features). A user manually labels an email as spam, and this is observed by the machine learning algorithm. When a new email comes in, the algorithm will compute the probability the email is spam given the text, and classify the e-mail as spam if the probability of it being spam exceeds the probability of it being nonspam.

k-Nearest Neighbor

The *k*NN is a simple, nonparametric algorithm for both classification and regression. When used for classification, the output is class membership; when used for regression, the output is the value of the response. There is essentially no training phase in *k*NN. In testing, a new datum is assigned to the class of the plurality (in classification) or mean outcome (in regression) of the *k* nearest training neighbors of the datum with respect to some distance metric (usually Euclidean, but others such as Hamming are also common). In classification, if $k = 3$ then the new datum is assigned to the majority class of the three nearest neighbors. Higher values of *k* reduce overfitting, but when *k* is too high the boundaries between classes blur. To address this issue, researchers can assign weights to the neighbors, such that nearer neighbors are assigned higher weights. A common weighting technique is to assign each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.

Decision Trees

DTs are a widely used nonparametric learner for both classification and regression that recursively split the data into increasingly smaller subsets where datapoints are more similar (i.e., have a smaller variance in regression or are more homogenous in classification). The resulting hierarchical data structure can be presented by a tree. These “white-box” models are becoming popular in social science because they are simple to understand and interpret. One of the most popular DT algorithms is CART (Breiman et al., 1984).

During training, DTs are built from the “root” by recursively (1) selecting a feature, (2) selecting a threshold or cutoff point on that feature that makes the remaining data on both sides of the threshold more similar, and (3) partitioning the remaining data into two smaller subsets on that feature and threshold. In theory, these steps are repeated until the terminal subsets (i.e., the “leaves”) contain only a single data point. In practice, researchers use regularization techniques so that each leaf contains a group of similar training points to reduce variance. In regression, each leaf represents the average value of *y* for data in that leaf. In classification, each leaf represents the plurality class for the data in that leaf. During testing, researchers can apply the sequence of decisions used to construct the tree to place each testing datum into the most appropriate leaf.

DTs determine which feature and threshold to split the data on by minimizing an in-sample loss function. Because it is impossible to explore all possible sequences of decisions, DTs minimize this loss function only on the current subset of data at each split, rather than on the entire data set. This “greedy” approach is efficient but may result in a locally optimal rather than globally optimal solution. In classification, the loss function tries to maximize homogeneity within leaves at every split. In so doing, researchers maximize entropy or impurity across leaves. For regression problems, the loss function minimizes variance with the leaf:

$$c = \sum (y_i - \bar{y})^2$$

where y is the mean of *y.* For classification problems, there are several possible loss functions: misclassification rate, entropy (deviance), and the Gini impurity. For example, the Gini impurity cost function is an indicator of how pure the two groups will be with respect to class heterogeneity after the split. Formally, the function can be written as:

$$g = \sum_{c=1}^{J} p_c(1 - p_c) = 1 - \sum_{c=1}^{J} p_i^2,$$

where p_c is the portion of units being classified as class c and $1 - p_c$ is the portion being misclassified.

Because the objective of DTs is to make leaves homogenous, these algorithms are susceptible to overfitting and require regularization. Three common ways to regularize trees are (1) limit the maximum leaf size, (2) limit the maximum tree depth during training, or (3) prune the tree depth after training.

Decision Trees: Ensemble Methods

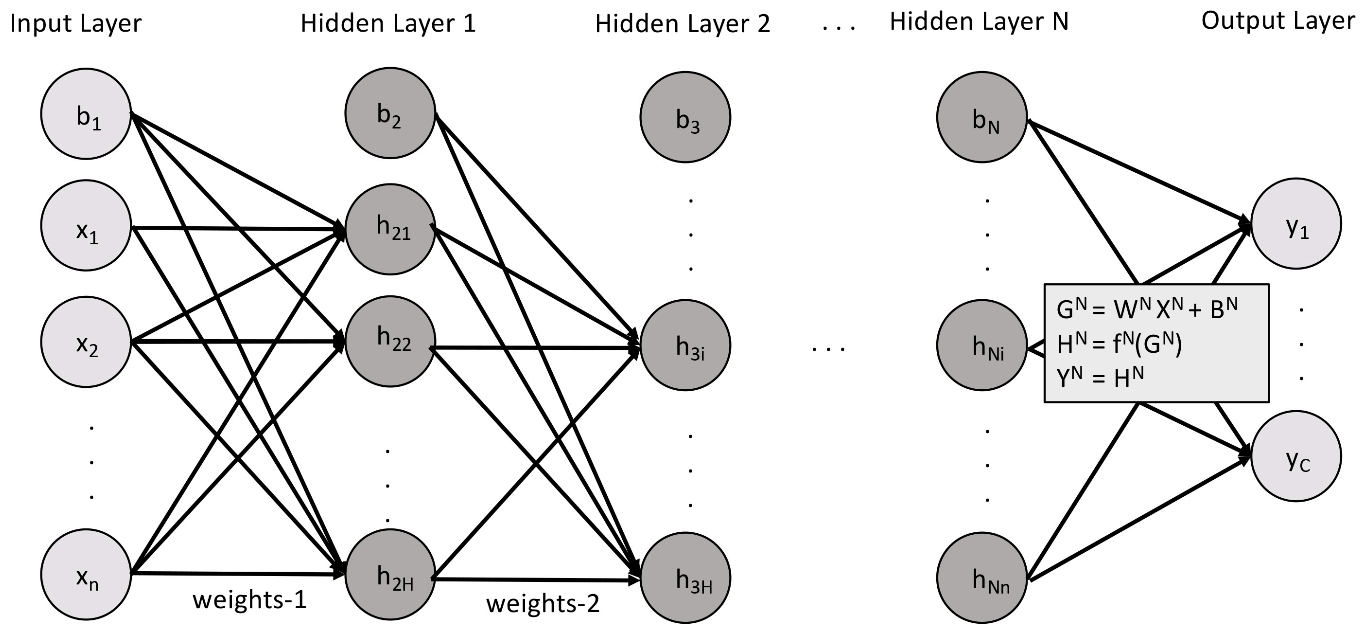
A disadvantage of single DTs is that their “greedy” optimization produces high-variance solutions. That is, minor modifications to the input data can produce large effects on the tree structure. To combat overfitting, ensemble methods construct many decision trees which then vote on the prediction for each datum. Bootstrap aggregating or “bagging,” is an ensemble method that constructs multiple decision trees by repeatedly resampling training data with replacement and generating a consensus prediction. Even with bagging, greedy trees will tend to use the same features for similar decision sequences and thus be correlated. Random forests combine bagging with a feature bagging scheme that forces greedy trees to explore different decision sequences with other predictive features: at each split, a given tree in the forest can only choose from a random subset of features (Amit & German, 1997; Breiman, 2001; Ho, 1995; Zhu, Zeng, & Kosorock, 2015). Because random forests are built from more varied data (bagging) and are forced to explore a broader solution space (feature bagging), the predictions produced by consensus of the forest tend to have lower variance than a single DT without increasing bias. That is, while predictions for a single tree are sensitive to noise in the training set, the average of many trees is not (if the trees are uncorrelated). However, ensemble methods are black-box algorithms—this decrease in variance thus comes at a cost of interpretability.

Another popular ensemble method, *boosting*, takes a different approach. Boosting algorithms train very shallow trees which are relatively high bias but low variance (weak learners) and successively add more weak learners that focus on areas of the data that the current ensemble is doing a poor job of predicting. The gradient boosting machine (GBM) algorithm trains many models in an additive, sequential way to improve predictions (Breiman, 1997; Friedman, 1999). Bayesian additive regression trees (BART) is a Bayesian "sum-of-trees" model motivated by ensemble methods and boosting algorithms (Chipman et al., 2010). By using priors, this Bayesian approach eliminates many user decisions about regularization, tree depth, and other hyperparameter tuning options.

Artificial Neural Networks and Deep Learning

Artificial neural networks are sophisticated supervised learning algorithms inspired by the human brain. Much like neurons in the brain firing due to impulses from other connected neurons, each node in a neural network takes a weighted combination of the outputs of incoming neurons as its input. An activation function then transforms this weighted combination and outputs a single scalar to any neurons to which it is connected. A perceptron is the simplest neural network, where input nodes for each feature feed to a single output node. When the activation function for this neuron is a sigmoid (i.e., logistic) function, this model corresponds exactly to logistic regression. However, the power of neural networks comes from the additional layers of neurons between the input and output layer to create multilayer perceptron (MLP). With a single fully connected "hidden" layer of neurons, a neural network can theoretically approximate any continuous function. Neural networks are trained by "backpropagating" the final loss to the weights in previous layers using calculus's chain rule. Figure 5 is an example of an MLP.

Figure 5. Feedforward neural network (multilayer perceptron).



In the hidden layers of a multilayer neural network, each neuron in layer n takes as input the outputs of layer n − 1 and multiplies these inputs by the weight parameters Wn before summing them. A bias parameter is also included in this linear combination (Gn =WnXn + Bn). The linear combination GNI feeding into each neuron is then passed through a nonlinear function (e.g., sigmoid or rectified linear unit) to determine it's output.

Despite this flexibility for modeling nonlinear functions, neural networks have lagged behind other supervised learning approaches in performance for decades because of the computational expense, difficulty, and large amount of data needed to train these models. More recently, neural networks with novel architectures began to outperform other supervised learning approaches in what is now collectively called *deep learning* (LeCun, Bengio, & Hinton, 2015). Two of the most enduring and influential ideas in deep learning are the concepts of convolution and recurrence. Convolutional neural networks are regularized feed forward networks that force neurons to hierarchically learn the complexity of the feature space using a series of convolution and pooling layers. Unlike feed-forward networks, which are directed and acyclic, recurrent networks have neurons that do not just feed forward but also connect back to earlier layers in the network. Because they give the network a type of memory, recurrent architectures are popular in natural language processing.

MLPs are not widely used by social science researchers as classifiers or regressors because (a) unlike GLMs and DTs, they are black-box algorithms with uninterpretable parameters, and (b) they require large amounts of data to train correctly. However, deep learning should become increasingly common in social science as researchers build familiarity with pretrained deep learning models, gain access to larger and more complex datasets, and deep learning integrates with causal inference.

Unsupervised Learning

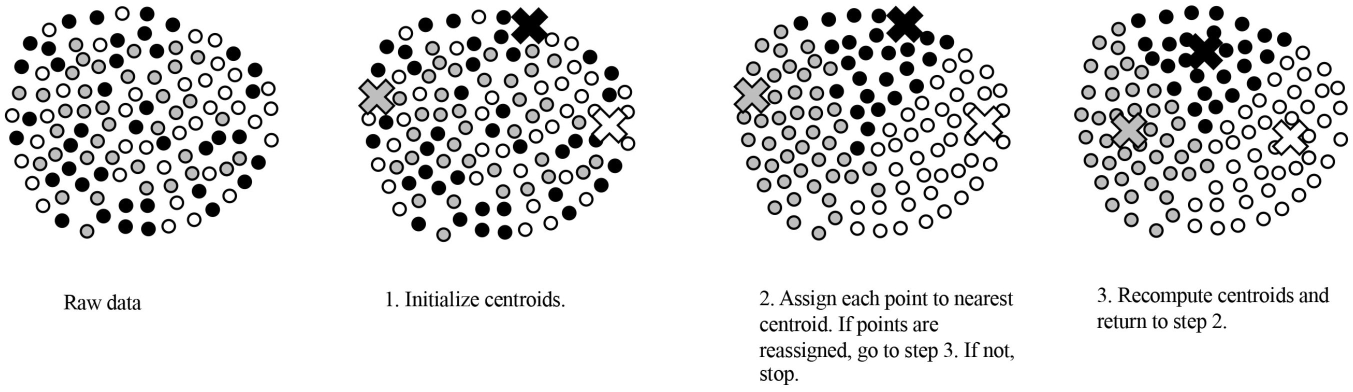
Unsupervised learning algorithms identify patterns and associations in the data without help (i.e., labeled examples) to train on. In most cases, these algorithms are used for clustering data into groups or reducing the dimensionality of data. Identifying different types of individuals on an online dating site or reducing a dataset of conversational data with 150 possible human gestures down to 10 latent ones are examples of unsupervised learning tasks. Some common examples of unsupervised learning algorithms include (a) *k*-means clustering; (b) hierarchical clustering; (c) principal component analysis; (d) t-SNE; and (e) unsupervised learning of documents for text analysis. This section discusses each of these algorithms in more detail.

Unsupervised Learning Algorithms

k-Means Clustering

The *k*-means clustering algorithm is among the simplest clustering algorithms. The intuition behind *k*-means is that the data can be clustered by assigning each of *n* observations into *k* (≤ *n*) clusters to the nearest centroid (i.e., mean) in the *x*-dimensional feature space. The *k*-means clustering is then optimized by minimizing the variance within clusters and maximizing the variance between clusters. Researchers initialize a predefined number of cluster centroids *k* in the feature space, then repeat the following two steps until the within-cluster variance is minimized: (1) reassign each unit to the nearest centroid or mean using some distance metric (e.g. Euclidean distance) and (2) calculate the new means of the units in each cluster as new centroids. As a larger *k* creates smaller groups with more granularity, while a smaller *k* creates larger groups with less granularity, it may be useful to run the algorithm multiple times varying this parameter. Silhouette scores are one popular method for evaluating the coherence and separation of clusters. Figure 6 is an example of *k*-means clustering.

Figure 6. K-means clustering algorithm.


Raw data
1. Initialize centroids.
2. Assign each point to nearest centroid. If points are reassigned, go to step 3. If not, stop.
3. Recompute centroids and return to step 2.

Example *k* = 3. The balls with different shadings represent different classes. The cross signs are centroids in the clusters.

Hierarchical Clustering

Hierarchical clustering generates, as the name indicates, a hierarchy of clusters. Agglomerative hierarchical clustering algorithms (e.g., the popular neighbor-joining unweighted pair-group method with unweighted mean UPGMA algorithm) take a bottom-up approach where each unit begins as an individual and is repeatedly paired with other individuals or clusters as researchers move up the hierarchy to a single group. In top-down divisive hierarchical clustering, all observations start in one cluster and the splits are made recursively as researchers move down the hierarchy. Like *k*-means, researchers must choose a distance metric to decide which clusters should be paired or divided. The Euclidean distance is a common metric for both *k*-means and hierarchical clustering, but other options include the Manhattan, Mahalanobis, or Hamming distances. Because hierarchical clustering is sensitive to outliers and overfitting, it is common to bootstrap the data to evaluate the stability of clusters.

Network Community Detection

Complex networks are graphical models where nodes are connected to other nodes via edges. These edges represent some relationship between the nodes. If the nodes are people in a social network, the edges might represent relationships, but other types of relationships can also be represented in a network: scientific papers that are connected via citations or documents that are connected by the co-occurrence of words. *Communities* in networks are clusters of nodes that are more densely connected to each other than they are to other nodes, and the discovery of these communities can be thought of as a form of unsupervised clustering. Even for data that cannot conventionally be thought of as a network, community detection can be a useful unsupervised learning approach when there are not many features but rich relational data (Newman, 2004). Modularity-detection, information theory, and label propagation are viable approaches to community detection. Some of these algorithms can produce hierarchical communities as well.

Dimensionality Reduction: Principal Components Analysis

Including noisy or highly correlated features in supervised learning can potentially lead to overfitting. The simplest solution to this problem is to use domain knowledge to remove irrelevant features. Dimensionality reduction is another way to reduce the number of features and focus on the principal features. It can be achieved by feature selection and feature extraction. Researchers might perform feature selection by using LASSO to identify relevant features and eliminate irrelevant features. Feature extraction creates new latent features, where the new features are combinations of the old features. Researchers can use either a linear or nonlinear feature extraction procedure. Principal components analysis (PCA) is an example of linear feature extraction; t-SNE is an example of nonlinear feature extraction.

PCA extracts new features from data by remapping the data to a smaller space. PCA's data reduction technique involves creating one or more index variables, the components, from a larger set of observed variables. PCA is used to determine the optimal choice of observed variables for each component and the optimal weights. Typically, this linear transformation is conducted through an eigenvector decomposition of the covariance matrix of the features. In this space, the axes are uncorrelated and each axis explains a percentage of the variance in the data set. The first principal component explains the greatest proportion of variance, the second explains the second greatest proportion of variance, and so on. Typically, researchers will only use the first few principal components which account for a majority of the variation and discard the rest as noise. PCA is often used in exploratory data analysis by plotting the data set onto the first two or three principal components to find clusters in the data. Alternatively, the user might use these principal components as latent features in a downstream supervised learning task.

Dimensionality Reduction: t-SNE

Another approach to dimensionality reduction, t-SNE, enables reduction by nonlinear feature extraction. It is a visualization algorithm of clusters for dimensionality reduction, that is, reducing high-dimensional data in a low-dimensional space such that similar objects are modeled by nearby points. Researchers use t-SNE to map high-dimensional data to a lower dimensional space and identify patterns in the data by locating observed clusters. The t-SNE algorithm proceeds in two steps. First, it constructs a probability distribution such that similar objects in the high-dimensional space have a high probability of being selected while dissimilar units have a low probability of being selected. Second, it constructs an analogous probability distribution in the low-dimensional space. The similarity metric is commonly the Euclidean distance. This approach, in contrast to PCA, is very computationally intensive. The input features are also no longer identifiable. It is most often used for data exploration and visualization.

Machine Learning Methods and Text Analysis

As the amount of social science data available online grows, text analysis has become an increasingly important part of social science research. The most common uses of machine learning for text analysis are supervised classification of documents by label, topic, or sentiment, unsupervised clustering of documents to discover topics, name disambiguation, and entity or meaning recognition.

Supervised Labeling of Documents

A common task in text analysis is the automated labeling of documents for topic, sentiment, or class. For example, rather than manually coding 1,000 book reviews for political orientation, researchers might want to label just 250 of them and train an algorithm to label the remaining reviews. By using word frequencies as features, many of the supervised learning algorithms described earlier can be adapted for this task. These algorithms, called *discriminative* models, learn hard decision boundaries between classes given the observed features. In contrast, *generative* models make structural assumptions about the relationships between classes and features by modeling each class as a probability distribution of features. These models are called generative because once trained, the label distribution can be used to generate features consistent with the class probabilistically (i.e., a document of words that co-occur within that class with high probability). Generative models have been particularly successful for classification in small text corpuses because the strong probabilistic assumptions make them robust to outliers in the training data.

As described earlier, the naive Bayes algorithm is one of the simplest supervised generative text classification models. Using Bayes Rule, naive Bayes models the probability [$P(Y|X)$] of a document with word distribution X belonging to a class Y as a product of the prior frequency of documents of that class [$P(Y)$], and the likelihood of the words observed in the document being generated from that class [$P(X|Y)$]. As a concrete example, suppose that the document is a Yelp review to be classified as positive or negative. The prior probability of the positive class $P(positive)$ would be the frequency of positive reviews in our training set. If the document consisted of only two words “very delicious,” the likelihood would be calculated as the probability of those words occurring together in positive sentiment documents. The naive assumption in naive Bayes is that researchers can approximate this probability by simply multiplying the probabilities $P(very|positive)$ and $P(delicious|positive)$ together.

There are several ways to calculate the probability of words being generated by documents (i.e., the likelihood), although all of them fundamentally assume that a document is an unordered “bag” of words. The simplest way is to compute the probability of a word being generated from class Y is to compare the number of times that X occurs or does not occur in training documents of class Y (Bernoulli naive Bayes). An alternative approach instead considers the counts of word in documents of class Y (categorical/multinomial naive Bayes). Once the prior probabilities and Bernoulli or categorical word distributions for each class have been calculated from the training data, the model can be used to apply labels to a testing dataset.

Unsupervised Topic Modeling

In unsupervised topic modeling, each document is not labeled with a single class but as a proportional mixture of latent unspecified "topics." For example, a Yelp review might be labeled as 50% Topic 1, 40% Topic 2, and 10% Topic 3. After applying the topic model, the researcher can then go through the corpus and interpret what the topics mean semantically. In this example, Topic 1 might largely correspond with descriptions of the food, Topic 2 might capture sentiment words, and Topic 3 may be about the service. Topic models are useful tools because they allow the researcher to both discover topics or themes in a corpus without a priori assumptions and to group documents that are topically or thematically similar together. Moreover, often the only hyperparameter that needs to be chosen is the number of topics.

Topic models differ from supervised learning of documents in several key ways. First, although a document is still represented as a bag of words, not every word in the document must come from the same topic. Second, the topics themselves are categorically distributed. Consider the popular probabilistic latent semantic analysis algorithm (pLSA). For each position in a document, the algorithm (1) picks a topic from the categorical distribution of topics and (2) generates a word from the categorical distribution of words given that topic. In the Bayesian version of pLSA, latent Dirichlet allocation (LDA), researchers have additional priors on the two categorical distributions, improving the performance on small corpuses.

Word Embedding Using Neural Networks

Topic models discover similarities between documents through the global co-occurrence of words between documents. However, the past 5 years have seen exciting advances in a different approach to finding structure in text data: learning the relationship between words based on their local co-occurrence with other words (i.e., the local context of words no more than 3–10 words away) using a neural network. Word embedding methods train a neural network to correctly predict the 3–5 words surrounding each word over thousands or millions of examples in large corpora. Although this is technically a supervised learning problem, the actual predictions are not actually of interest. What makes these models useful is the function encoded in the hidden layer of this trained network which transforms words into the lower dimensional space. Somewhat surprisingly, vocabulary words represented in this lower dimensional embedding space (i.e., transformed by the hidden layer of the neural network) are closer to other vocabulary words with similar semantic meaning and/or syntactic usage. The ability of word embeddings to solve analogy problems through simple vector arithmetic demonstrates this property. For example, subtracting the word vector for "man" from "king" and adding the word vector for "women" produces a new word vector that is very close to the word vector for "queen." These analogy tests also highlight that the word embedding space contains latent dimensions of meaning where relevant words are aligned along an axis (e.g., man and woman along a gender axis).

Word embeddings in social science are used for several aims. First, they are used to compare the relative meaning of words to other words across corpuses or across time. For example, researchers could ask how the usage of the word “man” differs between Victorian novels and modern news media. Second, performing dimensionality reduction on the vector space can help identify interesting latent meaning dimensions (e.g., gender) encoded in the corpus (Arseniev-Koehler & Foster, 2019). Third, modifications like Doc2Vec of the original word embedding approaches project documents into the same vector space as the vocabulary words. This is not quite equivalent to discovering topics but documents with similar language should be closer in vector space, and users can then perform subsequent unsupervised clustering of documents in this space to find groups of similar documents. This approach works better than topic modeling for identifying similar abbreviated documents (e.g. tweets). The two most popular word embedding algorithms are the original method Word2Vec (Mikolov et al., 2013) and GloVe, an approach that combines the local context approach with global co-occurrence information when producing vectors. However, word embedding methods are very active areas of research, and newer models like BERT use more sophisticated neural networks to capture more nuanced relationships between words than the widely used Word2Vec and GloVe models.

Machine Learning Methods and Causal Inference

Two of the most important developments in social science methodology over the last several decades include advances in causal inference methods and advances in machine learning methods. Yet they have largely developed along different axes of interest. Susan Athey (2018) states, “Despite the fascinating examples of ‘off-the-shelf’ of slightly modified prediction methods, in general ML prediction models are solving fundamentally different problems from much empirical work in social science, which instead focuses on causal inference” (p. 10). A model with high explanatory power that is optimized for prediction might be a relatively poor model, and even inferior to a model with low explanatory power, for causal inference. Social scientists interested in causal questions prefer unbiased estimates of causal effects to accurate prediction of outcomes (Mullainathan & Spiess, 2017). Off-the-shelf machine learning algorithms are not designed to attend to key assumptions in a causal model to identify causal effects such as the key identifying assumption that treatment assignment is unconfounded. Judea Pearl (2018), in the introductory chapter “Mind Over Data,” cautions,

> While awareness of the need to a causal model has grown leaps and bounds among the sciences, many researchers in artificial intelligence would like to skip the hard step of constructing or acquiring a causal model and rely solely on data for cognitive tasks. The hope … is that the data themselves will guide us to the right answers whenever causal questions come up. I am an outspoken skeptic of this trend because I know how profoundly dumb data are about cause and effects. (p. 16)

Researchers, however, are making progress in merging machine learning and causal inference methods. Indeed, the intersection of these two paradigms represents an extremely promising direction for empirical social scientific knowledge. Scholars of causal inference are increasingly adopting approaches from machine learning, while attending to identification strategies from statistics and econometrics, to enhance research in fields such as economics, political science, and sociology. Methods for conditioning on covariates offer a promising intersection between machine learning methods and the goals of estimating causal effects. In order to estimate causal effects without exogenous shocks, we assume selection on observables, or $[Y_i(x) \mid W_i = w_i \mid X_i = x_i]$ (Imbens & Rubin, 2015). Some research has suggested using LASSO for selecting covariates (Belloni, Chernozhukov, & Hansen, 2014), and other work has adopted machine learning algorithms for conditioning on covariates.

Propensity score weighting and matching on propensity scores has a long history in the causal inference literature. Propensity scores are summary measures of selection into treatment based on observed covariates $[P(x) = P(W_i = w_i \mid X_i = x_i)]$. Estimating propensity scores is a prediction problem and, thus, at least on the surface, lends itself well to machine learning approaches. Many recent papers consider various strategies for estimating propensity scores based on machine learning algorithms (Lee, Lessler, & Stuart, 2010). Classification trees and random forests offers promising approaches. Iterative procedures that mimic machine learning algorithms, but are not black-box, are also possible (see Imbens & Rubin, 2015, for a description).

The fundamental problem of causal inference is that researchers do not observe potential outcomes in both the treated and control states, and thus there is no ground truth in causal models like there is in predictive models. Adopting machine learning methods for estimating causal effects thus requires changing the objective function. Machine learning methods can be adopted for experimental and observational data settings, instrumental variable models, difference–indifference designs, and regression discontinuity. Estimation of average treatment effects has benefited from a residual balancing approach (Athey & Imbens, 2016) as well as a double machine learning approach (Chernozhukov et al., 2017). Estimation of treatment effect heterogeneity represents an especially promising use of machine learning methods for causal inference. Athey and Guido Imbens (2017) offer a detailed review of a variety of questions that can be addressed related to treatment effect heterogeneity. One approach developed by Athey and Imbens (2016) is to use DTs (*causal trees*) to partition the feature space according to effect heterogeneity, and estimate effects within each leaf (see also Brand et al., 2019, for an empirical application using observational data). This approach yields easily interpretable patterns of variation in treatment effects. Stefan Wager and Athey (2018) also offer a *causal forest* based on random forests, which is an average of many causal trees where trees differ due to subsampling. Each of these approaches offers new methods by which to uncover effect heterogeneity.

Conclusion

Machine learning methods are not without important limitations for social science research. First, many machine learning algorithms are black-box, rendering substantive interpretation in social science applications challenging. Relatedly, machine learning methods are not focused on solving estimation problems, while empirical social science researchers are often concerned with how effects change when one or more selected covariates are added to or eliminated from a model.

If the data from which a machine learns are limited, so too will be what researchers learn from that data. This is always true in model estimation, but machine learning models render researchers increasingly dependent on the belief that social scientists should be able to effectively predict social behavior. Peter M. Blau and Otis Dudley Duncan (1967) once noted,

> Sociologists are often disappointed in the size of the residual, assuming that this is a measure of their success in 'explaining' the phenomenon under study. They seldom reflect on what it would mean to live in a society where nearly perfect explanation of the dependent variable … In such a society it would indeed be true that some are destined to poverty almost from birth … By no effort of their own could they materially alter the course of destiny, nor could any stroke of fortune, good or ill, lead to an outcome not already in the cards. (p. 174)

Researchers should, in other words, not be dismayed that data and models, even machine learning models, cannot perfectly predict social behavior. That said, machine learning methods hold tremendous promise and indeed have and will produce a profound and transformative impact on empirical social science research.

Further Readings

Attewell, P., **Monaghan, D.**, & **Kwong, D.** (2015). *Data mining for the social sciences: An introduction*. Berkeley: University of California Press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin, Germany: Springer.

Domingos, P. (2015). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. New York, NY: Basic Books.

Foster, I., **Ghani, R.**, **Jarmin, R. S.**, **Kreuter, F.**, & **Lane, J.** (2017). *Big data and social science: A practical guide to methods and tools*. Boca Raton, FL: Taylor & Francis Group.

Goodfellow, I., **Bengio, Y.**, & **Courville, A.** (2016). *Deep learning*. MIT Press. Retrieved from http://www.deeplearningbook.org

Hastie, T., **Tibshirani, R.**, & **Friedman, J.** (2009). *The elements of statistical learning*. Berlin, Germany: Springer. Retrieved from http://www-stat.stanford.edu/˜tibs/ElemStatLearn/

Kelleher, J. D., **Namee, B. M.**, & **D'Arcy, A.** (2015). *Fundamentals of machine learning for predictive data analytics: Algorithms, worked examples, and case studies*. Cambridge, MA: MIT Press.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. Cambridge, MA: MIT Press.

References

Amit, Y., & **German, D.** (1997). Shape quantization and recognition with randomized trees. *Neural Computation*, 9, 1545–1588.

Arseniev-Koehler, A., & **Foster, J**, . (2019). Teaching an algorithm what it means to be fat: Machine learning as a model for cultural learning(Unpublished manuscript).

Athey, S. (2018). The impact of machine learning on economics(Unpublished manuscript).

Athey, S., & **Imbens, G.** (2016). Recursive portioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113, 7353–7360.

Athey, S., & **Imbens, G.** (2017). The state of applied econometrics: Causality and policy evaluation. *Journal of Economic Perspectives*, 31, 3–32.

Belloni, A., **Chernozhukov, V.**, & **Hansen, C.** (2014). High-dimensional methods and inference on structural and treatment effects. *Journal of Economic Perspectives*, 28, 29–50.

Blau, P. M., & **Duncan, O. D.** (1967). *The American occupational structure*. New York, NY: Wiley.

Brand, J. E., **Xu, J.**, **Koch, B.**, & **Geraldo, P.** (2019). Uncovering sociological effect heterogeneity using machine-learning(Unpublished manuscript).

Breiman, L. (1997). *Arcing the edge (Technical Report 486)*. Berkeley, CA: Statistics Department, University of California.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.

Breiman, L., **Freidman, J. H.**, **Olshen, R. A.**, & **Stone, C. J.** (1984). *Classification and regression tress*. Monterey, CA: Wadsworth & Brooks/Cole.

Chernozhukov, V., **Chetverikov, D.**, **Demirer, M.**, **Duflo, E.**, **Hansen, C.**, **Newey, W.**, & **Robins, J.** (2017). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21, C1–C68.

Chipman, H. A., **George, E. I.**, & **McCulloch, R. E.** (2010). BART: Bayesian additive regression trees. *Annals of Applied Statistics*, 4, 266–298.

Cortes, C., & **Vapnik, V.** (1995). Support-vector networks. *Machine Learning*, 20, 273–297.

Crevier, D. (1993). *AI: The tumultuous history of the search for artificial intelligence*. New York, NY: Basic Books.

Freidman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), 1189–1232.

Geman, S., **Bienenstock, E.**, & **Doursat, R.** (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, 4, 1–58.

Hastie, T., **Tibshirani, R.**, & **Friedman, J.** (2009). The elements of statistical learning. Springer. Retrieved from http://www-stat.stanford.edu/˜tibs/ElemStatLearn/

Ho, T. K. (1995). Random decision forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, QC. August 14–16.

Imbens, G., & **Rubin, D.** (2015). *Causal inference for statistics, social, and biomedical sciences*. Cambridge, England: Cambridge University Press.

James, G., **Witten, D.**, **Hastie, T.**, & **Tibshirani, R.** (2017). *An introduction to statistical learning: With applications in R*. Berlin, Germany: Springer.

LeCun, Y., **Bengio, Y.**, & **Hinton, G.** (2015). Deep learning. *Nature*, 521, 436–444.

Lee, B. K., **Lessler, J.**, & **Stuart, E.** (2010). Improving propensity score weighing using machine learning. *Statistics in Medicine*, 29, 337–346.

Maini, V., & **Sabri, S.** (2017). *Machine learning for humans*. Retrieved from https://www.dropbox.com/s/e38nil1dnl7481q/machine_learning.pdf?dl=0

Mikolov, T., **Chen, K.**, **Corrado, G.**, & **Dean, J.** (2013). Efficient estimation of word representations in vector space. *In Proceedings of Workshop at ICLR*. Retrieved from http://arxiv.org/pdf/1301.3781.pdf

Mitchell, T. M. (1997). *Machine learning*. Boston, MA: McGraw-Hill.

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

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, 066133.

Pearl, J. (2018). *The book of why*: *The new science of cause and effect*. New York, NY: Basic Books.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58, 267–288.

Wager, S., & **Athey, S.** (2018). Estimation and inference of heterogenous treatment effects using random forests. *Journal of the American Statistical Association*, 113, 1228–1242.

Wolpert, D. H., & **Macready, W. G.** (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82.

Zhu, R., **Zeng, D.**, & **Kosorock, M. R.** (2015). Reinforcement learning trees. *Journal of the American Statistical Association*, 110, 1770–1784.