# 11. Machine learning and marketing

*Daria Dzyabura and Hema Yoganarasimhan*

Machine learning (ML) refers to the study of methods or algorithms designed to learn the underlying patterns in the data and make predictions based on these patterns.[1] ML tools were initially developed in the computer science literature and have recently made significant headway into business applications. A key characteristic of ML techniques is their ability to produce accurate out-of-sample predictions.

Academic research in marketing has traditionally focused on causal inference. The focus on causation stems from the need to make counterfactual predictions. For example, will increasing advertising expenditure increase demand? Answering this question requires an unbiased estimate of advertising impact on demand.

However, the need to make accurate predictions is also important to marketing practices. For example, which consumers to target, which product configuration a consumer is most likely to choose, which version of a banner advertisement will generate more clicks, and what the market shares and actions of competitors are likely to be. All of these are prediction problems. These problems do not require causation; rather, they require models with high out-of-sample predictive accuracy. ML tools can address these types of problems.

ML methods differ from econometric methods both in their focus and the properties they provide. First, ML methods are focused on obtaining the best out-of-sample predictions, whereas causal econometric methods aim to derive the best unbiased estimators. Therefore, tools that are optimized for causal inference often do not perform well when making out-of-sample predictions. As we will show below, the best unbiased estimator does not always provide the best out-of-sample prediction, and in some instances, a biased estimator performs better for out-of-sample data.[2]

Second, ML tools are designed to work in situations in which we do not have an a priori theory about the process through which outcomes observed in the data were generated. This aspect of ML contrasts with econometric methods that are designed for testing a specific causal theory. Third, unlike many empirical methods used in marketing, ML techniques can accommodate an extremely large number of variables and uncover which variables should be retained and which should be dropped. Finally, scalability is a key consideration in ML methods, and techniques such as feature selection and efficient optimization help achieve scale and efficiency. Scalability is increasingly important for marketers because many of these algorithms need to run in real time.

To illustrate these points, consider the problem of predicting whether a user will click on an ad. We do not have a comprehensive theory of users' clicking behavior. We can, of course, come up with a parametric specification for the user's utility of an ad, but such a model is unlikely to accurately capture all the factors that influence the user's decision to click on a certain ad. The underlying decision process may be extremely complex and potentially affected by a large number of factors, such as all the text and images in the ad, and the user's entire previous browsing history. ML methods can automatically learn which of these factors affect user behavior and how they interact with each other, potentially in a highly non-linear fashion, to derive the best functional form that explains user behavior virtually in real time. ML methods typically assume a model or structure to learn, but they use a general class of models that can be very rich.

Broadly speaking, ML models can be divided into two groups: supervised learning and unsupervised learning. Supervised learning requires input data that has both predictor (independent) variables and a target (dependent) variable whose value is to be estimated. By various means, the process learns how to predict the value of the target variable based on the predictor variables. Decision trees, regression analysis, and neural networks are examples of supervised learning. If the goal of an analysis is to predict the value of some variable, then supervised learning is used. Unsupervised learning does not identify a target (dependent) variable, but rather treats all of the variables equally. In this case, the goal is not to predict the value of a variable, but rather to look for patterns, groupings, or other ways to characterize the data that may lead to an understanding of the way the data interrelate. Cluster analysis, factor analysis (principle components analysis), EM algorithms, and topic modeling (text analysis) are examples of unsupervised learning.

In this chapter, we first discuss the bias-variance tradeoff and regularization. Then we present a detailed discussion of two key supervised learning techniques: (1) decision trees and (2) support vector machines (SVM). We focus on supervised learning, because marketing researchers are already familiar with many of the unsupervised learning techniques. We then briefly discuss recent applications of decision trees and SVM in the marketing literature. Next, we present some common themes of ML such as feature selection, model selection, and scalability, and, finally, we conclude the chapter.

## BIAS–VARIANCE TRADEOFF

The bias-variance tradeoff demonstrates the key difference between prediction and causal-inference problems. In causal-inference problems, the goal is to obtain unbiased estimates of the model parameters. However, when the goal is the best out-of-sample prediction, parameter values do not need to be unbiased. Therefore, methods built for causal inference are not optimized for prediction, because they restrict themselves to unbiased estimators.

When assessing how good a model will be at making predictions, we distinguish between two different sources of error: bias and variance. Error due to bias is the systematic error we can expect from estimating the model on a new data set. That is, if we were to collect new data and estimate the model several times, how far off would these models' predictions be, on average? The error due to variance is the extent to which predictions for a point differ across different realizations of the data. For example, a model that overfits to the training data will have high variance error because it would produce very different estimates on different data sets. Overfitting occurs when a model is fit too closely to a finite sample data set. Thus, when the model is applied to a different finite sample, it performs poorly.

Let us now examine how these two sources of error affect a model's predictive ability. Let $y$ be the variable we want to predict, and let $x_1, \ldots, x_n$ be the predictors. Suppose a function exists that relates $y$ to $x$, $y = f(x) + \varepsilon$, where $\varepsilon$ is normally distributed with mean 0 and variance $\sigma_\varepsilon$. We would like to estimate a model, $\hat{f}(x)$, to minimize the mean squared error of the prediction. The expected squared prediction error at point $x$ is $MSE(x) = E[(y - \hat{f}(x))^2]$, which can be decomposed as follows:

$$MSE(x) = (E[\hat{f}(x)] - f(x))^2 + E[\hat{f}(x) - E[\hat{f}(x)]]^2 + \sigma_\varepsilon^2 \quad (11.1)$$

The last term, $\sigma_\varepsilon^2$, is inherent noise in the data, so it cannot be minimized and is not affected by our choice of $\hat{f}(x)$. The first term is the squared bias of the estimator; the second term is the variance. We can see that both the bias and variance contribute to predictive error. Therefore, when we are trying to come up with the best predictive model, an inherent tradeoff exists between bias and variance of the estimator. By ensuring no bias, unbiased estimators allow no tradeoff. We refer readers to Hastie et al. (2009) for the formal derivation of the above.

To allow for a tradeoff, we introduce the concept of regularization. Instead of minimizing in-sample error alone, we introduce an additional term and solve the following problem:

$$minimize_x \sum_i^n (y_i - \hat{f}(x_i))^2 + \lambda R(\hat{f}) \qquad (11.2)$$

The term $R(\hat{f})$ is a regularizer. It penalizes functions that create substantial variance. The specific form of $R(\hat{f})$ will depend on the model to be estimated, $\hat{f}$, and is typically chosen a priori. The weight given to the regularizer relative to in-sample fit is captured by $\lambda$, which controls the amount of regularization and allows us to maximize predictive performance by optimally trading off bias and variance. A key idea in ML is that λ can be optimally derived from the data itself instead of being imposed exogenously. Usually it is selected using cross-validation, by splitting the data into several training and validation sets. By repeatedly holding out some subset of the data for validation, we can determine the value of $\lambda$ that leads to the best prediction for the holdout data. Therefore, the model is explicitly optimized to make the best out-of-sample prediction given the data. Note that by introducing regularization, we have sacrificed the unbiasedness of the estimator in favor of getting better out-of-sample predictions. A more formal treatment of regularization follows later.

By empirically making the bias–variance tradeoff, regularization allows us to consider a much broader class of models. For example, we can have models with many more predictors than observations, or models with many parameters, such as high-degree polynomials, or highly non-linear models, such as decision trees or random forests.

The ability to consider a rich class of models is important for applications with no hypothesized parametric model that can be estimated on the data. For example, in the computer science literature, a commonly studied problem is image recognition, where the goal is to recognize the object in a picture, and the data are pixels. Of course, this case has many more predictors than data points, and we have no model for how the pixels actually combine to make an image of, say, a dog or a house. As such, classical ML applications focus much less on modeling than does econometrics or classical statistics. Rather, the focus is on "learning" from the data. In such settings, weak assumptions about model structure combined with large data sets that are often characterized by high dimensions and a lot of missing data lead to natural concerns in (1) computation and (2) data overfitting. To deal with these challenges, several techniques have been developed, including regularization, cross-validation, and approximate optimization methods.

## DECISION TREE-BASED MODELS

In the most general formulation of a statistical prediction problem, we are interested in the conditional distribution of some variable *y* given a set of other variables $x = (x_1,. . .,x_p)$. In ML, the *x* variables are often referred to as "predictors" or "features" (in marketing, these are usually called explanatory variables), and the focus of many ML problems is to find a function $f(x)$ that provides a good prediction of *y*. We typically have some observed data $\{x,y\}$ and want to compute a good prediction of *y* for a new draw of *x*. The definition of a good predictor is based on its ability to minimize a user-defined loss function such as the sum of squared residuals. The relevant loss in a prediction problem is associated with new out-of-sample observations of *x*, not the observations used to fit the model.

There are two main types of supervised learning models: (1) decision trees and (2) support vector machines. We discuss decision trees here and support vector machines in the next section.

Linear regression (for continuous variables) and logistic regression (for discrete data) are popular tools used for summarizing relationships in the data. An alternative way to build a predictor is to use a decision tree. We start by describing the simplest class of tree-based models, called classification and regression trees (CART). Breiman et al. (1984) discuss the advantages and disadvantages of CART and then conclude with a description of the boosting technique that alleviates some of the issues with CART.

### Classification and Regression Trees (CART)

CART recursively partitions the input space corresponding to a set of explanatory variables into multiple regions and defines a local model on each region, which could be as simple as assigning an output value for each region. This type of partitioning can be represented by a tree structure, where each leaf of the tree represents an output region. Consider a data set with two input variables $\{x_1,x_2\}$ that are used to predict or model an output variable *y* using a CART. An example tree with three leaves (or output regions) is shown in Figure 11.1. This tree first asks if $x_1$ is less than or equal to a threshold $t_1$. If yes, it assigns the value of 1 to the output *y*. If not (i.e., if $x_1 > t_1$), it then asks if $x_2$ is less than or equal to a threshold $t_2$. If yes, it assigns $y = 2$ to this region. If not, it assigns the value $y = 3$. The chosen *y* value for a region corresponds to the mean value of *y* in that region in the case of a continuous output and the dominant *y* in case of discrete outputs.

A general tree model can be expressed as follows:

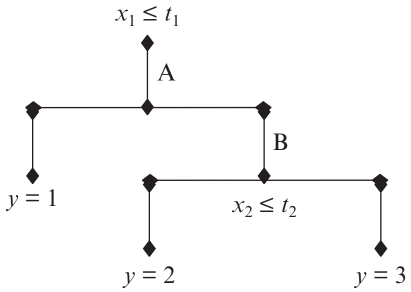


*Figure 11.1 Example of a CART model*

$$y = f(x) = \sum_{k=1}^{K} w_k I(\mathbf{x} \in R_k) = \sum_{k=1}^{K} w_k \phi(\mathbf{x}; v_k), \tag{11.3}$$

where **x** denotes the vector of features or explanatory variables, $R_k$ is the $k^{th}$ region of the $K$ regions used to partition the space, $w_k$ is the predicted value of $y$ in region $k$, and $v_k$ is the choice of variables to split on as well as their threshold values for the path to the $k^{th}$ leaf. When $y$ is continuous, $w_k$ is the mean response in the $k^{th}$ region. For classification problems where the outcome is discrete, $w_k$ refers to the distribution of the $y$'s in the $k^{th}$ leaf.

Growing a tree requires optimally partitioning the data to derive the points of split (threshold values of **x** at each tree node) as well as the value of $y$ in each leaf, which is an NP-complete problem (Hyafil and Rivest, 1976). It is commonly solved using a greedy algorithm that incrementally builds the tree by choosing the best feature and the best split value for that feature at each step of the tree-construction process. That is, the greedy algorithm makes the locally optimal choice at each stage of the optimization process with the hope of finding a global optimum.

Trees are trained (or "grown") by specifying a cost function that is minimized at each step of the tree using a greedy algorithm. For a tree that uses two-way splits, the split function determines the best feature ($j$*)and its corresponding split value ($v$*) as follows:

$$(j^*, \upsilon^*) = arg \quad \min_{j \in (1,\ldots.,d), \upsilon \in X_j} cost(x_i, y_i{:}\, x_{ij} \leq \upsilon) + cost(x_i, y_i{:}\, x_{ij} > \upsilon), \tag{11.4}$$

where $d$ is the number of input variables, $X_j$ is the domain of values assumed by $x_j$, and *cost* is a function that characterizes the loss in prediction accuracy due to a given split. The *cost* function that is used for evaluating splits depends on the setting in which the decision tree would be used. For example, the cost function could be the mean squared error of the predictions in the case of the decision tree being used in a regression setting, or the misclassification rate in a classification setting. The split procedure evaluates the costs of using all of the input variables at every possible value that a given input variable can assume, and chooses a variable $(j^*)$ and the value $(\upsilon^*)$ that yields the lowest cost. The stopping criteria for the tree construction can either be based on the cost function or on desired properties of the tree structure. For example, tree construction can be stopped when the reduction in cost as a consequence of introducing a new tree node becomes small or when the tree grows to a predefined number of leaves or a predefined depth.

The greedy algorithm implies that at each split, the previous splits are taken as given, and the cost function is minimized going forward. For instance, at node B in Figure 11.1, the algorithm does not revisit the split at node A. However, it considers all possible splits on all the variables at each node, even if some of the variables have already been used at previous nodes. Thus, the split points at each node can be arbitrary, the tree can be highly unbalanced, and variables can potentially repeat at later child nodes. All of this flexibility in tree construction can be used to capture a complex set of flexible interactions, which are learned using the data.

CART is popular in the ML literature for many reasons. The main advantage of a simple decision tree is that it is very interpretable—inferring the effect of each variable and its interaction effects is easy. Trees can accept both continuous and discrete explanatory variables, can work with variables that have many different scales, and allow any number of interactions between features (Murphy, 2012). A key advantage of CART over regression models is the ability to capture rich non-linear patterns in data, such as disjunctions of conjunctions (Hauser et al., 2010). CART models are also robust to errors, both in the output and in the explanatory variables, as well as missing explanatory variable values for some of the observations. Further, CART can do automatic variable selection in the sense that CART uses only those variables that provide better accuracy in the regression or classification task. Finally, because the CART technique is non-parametric, it does not require data to be linearly separable, and outliers do not unduly influence its accuracy. These features make CART the best off-the-shelf classifier available.

Nevertheless, CART has accuracy limitations because of its discontinuous nature and because it is trained using greedy algorithms and thus can converge to a local maximum. Also, decision trees tend to overfit data and provide the illusion of high accuracy on training data, only to underperform on the out-of-sample data, particularly on small training sets. Some of these drawbacks can be addressed (while preserving all of the advantages) through boosting, which gives us MART.

## Boosting or MART

Boosting is a technique that can be applied to any classification or prediction algorithm to improve its accuracy (Schapire, 1990). Applying the additive boosting technique to CART produces MART (multiple additive regression trees), which has been shown empirically to be the best classifier available (Caruana and Niculescu-Mizil, 2006; Hastie et al., 2009). MART can be interpreted as a weighted linear combination of a series of regression trees, each trained sequentially to improve the final output using a greedy algorithm. MART's output $F_N(x)$ can be written as

$$F_N(x) = \sum_{k=1}^{N} \alpha_k f_k(x, \beta_k), \tag{11.5}$$

where $f_k(x, \beta_k)$is the function modeled by the $k^{th}$ regression tree and $\alpha_k$ is the weight associated with the $k^{th}$ tree. Both $f_k(\cdot)$s and $\alpha_k$s are learned during the training or estimation.

We choose $f_k(x, \beta_k)$ to minimize a prespecified cost function, which is usually the least-squares error in the case of regressions and an entropy or logit loss function in the case of classification or discrete choice models. Given the set of data points $(x_i, y_i)|1 \leq i \leq n$ and a loss function $L(y_i, \hat{y}_i)$ corresponding to making a prediction of $\hat{y}_i$ for $y_i$, the boosting technique minimizes the average value of the loss function. It does so by starting with a base model $F_1(x)$ and incrementally refining the model in a greedy fashion:

$$F_1(x) = arg \min_{f_1} \sum_{i=1}^{n} L(y_i, f_1(x_i)), \tag{11.6}$$

$$F_k(x) = F_{k-1}(x) + arg\min_{f_k} \sum_{i=1}^{n} L(y_i, F_{k-1}(x_i) + f_k(x_i)) \tag{11.7}$$

At each step, $f_k(x, \beta_n)$ is computed so as to best predict the residual value $y - F_{k-1}(x)$. In particular, boosting techniques use gradient descent to compute $f_k(\cdot)$ at each step using $g_k$, which is the gradient of $L(y, F(x))$ evaluated at $F(x) = F_{(k-1)}(x)$:

$$g_{ik} = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x) = F_{k-1}(x)} \tag{11.8}$$

Given $g_k$, gradient boosting makes the following update:

$$F_k(x) = F_{k-1}(x) - \gamma_k \cdot g_k, \tag{11.9}$$

where $\gamma_k$ is the step length chosen so as to best fit the residual value:

$$\gamma_k = arg \min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{k-1}(x_i) - \gamma \times g_k(x_i)) \qquad (11.10)$$

Note the gradients are easy to compute for the traditional loss functions. For example, when the loss function is the squared-error loss function $1/2(y_i - F(x_i))^2$, the gradient is simply the residual $y_i - F(x_i)$. In general, boosting techniques can accommodate a broad range of loss functions and can be customized by plugging in the appropriate functional form for the loss function and its gradient.

MART can be viewed as performing gradient descent in the function space using "shallow" regression trees (i.e., trees with a small number of leaves). MART works well because it combines the positive aspects of CART with those of boosting. CART, especially shallow regression trees, tends to have high bias but low variance. Boosting CART models addresses the bias problem while retaining the low variance. Thus, MART produces high-quality classifiers.

### Application of Boosted Decision Trees in Marketing

Two recent studies use boosted trees in marketing applications. In a study involving millions of searches, Yoganarasimhan (2017) used boosted regressions (MART) to show that personalized rank orderings for each consumer (and each instance of search) can improve the likelihood of consumers clicking and dwelling on search results. Further, she finds that logistic regression provides no improvement over the baseline.[3] She uses the predictive model to examine the heterogeneity in returns from personalization as a function of user-history and query-type. Rafieian and Yoganarasimhan (2017) also use boosted trees to build a targeting model for mobile in-app advertisements. In their study, they use data from over 27 million impressions in mobile apps. They show that boosted trees perform better than other commonly used models such as OLS regressions, logistic regressions, LASSO, and Random Forests for predicting click-through rates of consumers for mobile advertisements. They use their results to examine the relative value of behavioral and contextual targeting in mobile ads, and to explore the impact of targeting on competition among advertisers and the incentives of the platform to share data with advertisers. Together, these studies establish the effectiveness of decision-tree-based models in improving marketing decisions.

## SUPPORT VECTOR MACHINES

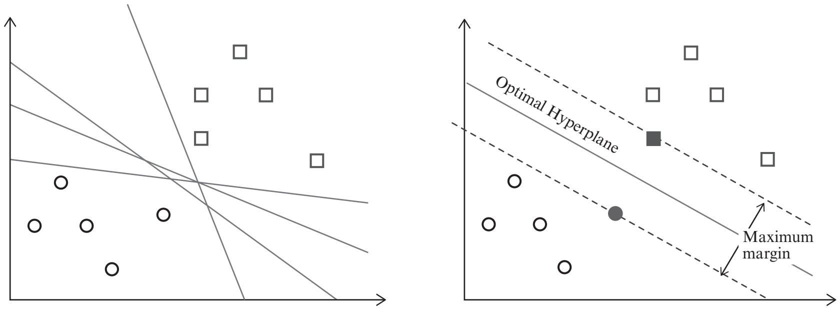
A support vector machine, or SVM, is a semi-parametric method typically used for a specific kind of prediction problem—the classification problem. SVMs are robust to a large number of variables and small samples, can learn both simple (e.g., linear) and complex classification models, and have built-in regularizers that help avoid overfitting. They also produce classifiers with theoretical guarantees of good predictive performance (of unseen data). The theoretical foundations of this method come from statistical learning theory.

### Classification Problems

Classification problems are prediction problems in which the variable of interest is discrete, such as which product(s) the consumer will consider or purchase, or whether or not a consumer will purchase. A general form of a binary (two-class) classification problem is described as follows: given a set $S$ of labeled data points, $S = \{(x_i, y_i)\}$, $|S| = N$, where $x_i \in R_d$ are vectors of predictor variables and $y_i \in \{+1, -1\}$ are class labels, construct a rule that correctly assigns a new point $x$ to one of the classes. A classifier is a rule that is trained on the labeled data and applied to new data to predict the labels. A classifier is typically represented as a function $(x)$: $R^d \rightarrow R$, called the classifier function. In the case of binary classifiers, a point is assigned the label +1 if $f(x) \geq 0$, and the label −1 otherwise.

### Linear Classifiers

We start by describing the SVM methodology for the simple case of *linear classifiers* where the classifying function $f(x)$ has the form $f(x) = \beta_0 + \beta^T x$. A set of points $\{(x_i, y_i)\}$ is *linearly separable* if all the points in the set can be correctly classified using a linear classifier. That is, if $y_i \in \{-1, +1\}$, the set is linearly separable if a linear function $f(x)$ exists such that $y_i \cdot f(x_i) > 0$ for all $i = 1, \ldots, N$. For example, the set of points in Figure 11.2 is linearly separable. To aid visual exposition, the example depicts a simple case with two continuous predictors, $x_1$, $x_2$. However, the same concepts apply to tasks in which the problem is higher dimensional. Note that in this example, several lines (or, more generally, hyperplanes) exist that correctly classify the data; see Figure 11.2a. We can ask whether some are better than others. To help us choose a classifier, we define the concept of a *margin*, which captures this intuition: a line is a weak classifier if it passes too close to the points, because it will be sensitive to noise and will not generalize well. Therefore, our goal should



(a) Many linear classifiers can correctly classify this set of points

(b) The maximum margin classifier is the strongest

*Figure 11.2 A linearly separable set of points*

be to find a line that passes as far as possible from all the points, as shown in Figure 11.2b.

That is, we seek the classifier that gives the *largest minimum distance* to all the training examples; this distance is called the "margin" in SVM theory. For now, we rely on intuition to motivate this choice of the classifier; theoretical support for this choice is provided below. The optimal separating hyperplane maximizes the margin of the training data, as in Figure 11.2b. The training examples that are closest to the hyperplane are called support vectors. Note that the margin in Figure 11.2b, $M$, is twice the distance to the support vectors. The distance between a point $x_i$ and the hyperplane $(\beta, \beta_0)$ is given by

$$distance = \frac{|\beta_0 + \beta^{\mathrm{T}} x_i|}{||\beta||} \tag{11.11}$$

Thus, the margin is given by $M = 2 \cdot \frac{|\beta_0 + \beta^{\mathrm{T}} x_i|}{\|\beta\|}$, which is twice the distance to the closest points. Because a single hyperplane can be defined in infinitely many ways, by scaling with $\|\beta\|$, the parameters of the hyperplane are normalized such that $|\beta_0 + \beta^T x| = 1$. Then the margin is simply given by $= \frac{2}{\|\beta\|}$. A hyperplane $(\beta, \beta_0)$ is called a $\gamma$-margin separating hyperplane if $y_i \cdot f(x_i) > \gamma$ for all $(x_i, y_i) \in S$.

We can now write the problem of finding the maximum margin linear (MML) classifier as an optimization problem that maximizes the margin $M$ subject to some constraints. It is typically written as minimizing $\frac{1}{M^2}$, which is a function of $\beta$, and the constraints require that the hyperplane correctly classifies all the training examples $x_i$:

$$\text{minimize}_x \frac{1}{2} \| \beta \|^2 \tag{11.12}$$

subject to

$$y_i(\beta^T x_i + \beta_0) \geq 1 \ \ \forall i = 1,\ldots,N.$$

The MML has several noteworthy properties. First, it can be efficiently solved because it is a quadratic optimization problem that has a convex objective function. Second, it has a unique solution for any linearly separable set of points. Third, the solution to the MML classifier depends only on the subset of points that act as the support vectors. The other points can lie anywhere outside the margin, and their positions do not affect the solution.

### Allowing Misclassified Examples

Because the optimal separating hyperplane is drawn as far away from the training examples as possible, the MML is only robust to noisy predictors, not to noisy labels. Because it does not allow for misclassified examples, even a single misclassification error in the training data can radically affect the solution. To address this problem, the above approach can be relaxed to allow for misclassified examples. The main idea is this: instead of constraining the problem to classify all the points correctly, explicitly penalize incorrectly classified points. The magnitude of the penalty attached to a misclassification will determine the tradeoff between misclassifying a training example and the potential benefit of improving the classification of other examples. The penalization is done by introducing *slack variables* for each constraint in the optimization problem in equation (11.12), which measure how far on the wrong side of the hyperplane a point lies—the degree to which the margin constraint is violated. The optimization problem then becomes

$$\text{minimize}_x \ \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \tag{11.13}$$

subject to $y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i$, $\xi_i \geq 0, \forall i = 1,\ldots,N.$

Now, if the margin constraint is violated, we will have to set $\xi_i > 0$ for some data points. The penalty for this violation is given by $C \cdot \xi_i$, and it is traded off with the possibility of decreasing $\|\beta\|^2$. Note that for linearly separable data, if $C$ is set to a sufficiently large value, the optimal solution will have all the $\xi_i = 0$, corresponding to the MML classifier. In general, the larger the value of $C$, the fewer margin constraints will be violated. The users typically choose the value of $C$ by cross-validation. Note that in this more general formulation, many more data points affect the choice of the hyperplane: in addition to the points that lie on the margin, the misclassified examples also affect it. We will come back to this formulation shortly and see how this formulation can be seen from the point of view of regularization.

The above problem is also a quadratic optimization problem that has a convex objective function and therefore can be efficiently solved. One common method for solving it is by introducing Lagrange multipliers and forming a dual problem. The Lagrange function resulting from the optimization problem in equation (11.13) is obtained by introducing Lagrange multipliers to the objective function for the constraints:

$$L_p = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i(y_i(\beta^T x_i + \beta_0) - (1-\xi_i)) - \sum_{i=1}^{N}\mu_i\xi_i\alpha_i, \mu_i, \xi_i \geq 0, \tag{11.14}$$

where $\mu_i$ and $\alpha_i$ are Lagrange multipliers. We obtain first-order conditions by taking derivatives with respect to $\beta$, $\beta_0$, and $x_i$:

$$\beta = \sum_{i=1}^{N}\alpha_i y_i x_i, \tag{11.15}$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i, \alpha_i = C - \mu_i, \forall i = 1,\ldots,N.$$

Plugging these into the Lagrangian function in (11.14), we obtain the Lagrangian dual problem:

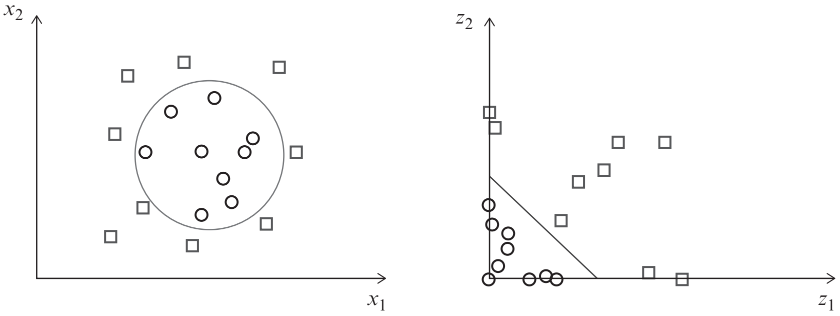
$$\text{maximize}\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_i' y_i y_i' x_i^T x_i' \tag{11.16}$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \sum_{i=1}^{N}\alpha_i y_i = 0.$$

Note that in the above optimization problem, the input features $x_i$ only enter via inner products. This property of SVM is critical to the computational efficiency for nonlinear classifiers. Next, we show how the SVM machinery can be used to efficiently solve nonlinear classification problems.

### Non-linear Classification—Kernel Method

Suppose now that our data are not separable by a linear boundary, but can be separated by a non-linear classifier, such as in Figure 11.3a. The kernel



(a) Points cannot be correctly separated with a linear classifier, but a nonlinear classifier $f(x) = -2 + x_1^2 + x_2^2$ separates them perfectly.

(b) The same points in the transformed space are now linearly separable.

*Figure 11.3 Nonlinear classification*

method, also known as the "kernel trick," is a way to transform the data into a different space, and construct a linear classifier in this space. If the transformation is non-linear, and the transformed space is high dimensional, a classifier that is linear in the transformed space may be nonlinear in the original input space.

Consider the example of the circle shown in Figure 11.3a, which represents the equation $x_1^2 + x_2^2 = 2$. That is, the non-linear classifier $f(x) = -2 + x_1^2 + x_2^2$ separates the data set perfectly. Let us now apply the following nonlinear transformation to **x**:

$$z = \Phi(\mathrm{x}):\ z_1 = x_1^2,\ \ z_2 = x_2^2 \tag{11.17}$$

After the transformation, the classifier becomes a linear one defined as follows: $(z) = -2 \cdot 1 + 1 \cdot z_1 + 1 \cdot z_2 = \tilde{\beta}z$.

Now, if we plot the data in terms of $z$, we have linear separation, as shown in Figure 11.3b. The transformed space that contains the **z** vectors is called the feature space, because its dimensions are higher-level features derived from the raw input **x**. The transform, typically referred to as the *feature transform*, is useful because the non-linear classifier (circle) in the $X$-space can be represented by a linear classifier in the $Z$-space. Let $d$ be the dimensionality of the $X$ space, and $\tilde{d}$ the dimensionality of the $Z$ space; similarly, we let $\tilde{\beta}$ represent the weight vector. Then a linear classifier $\tilde{f}$ in **z** corresponds to a classifier in **x**, $f(x) = \tilde{f}(\Phi(\mathbf{x}))$.

If the transformed data are linearly separable, we can apply methods developed for linear classifiers to obtain the solution in the transformed space, $\tilde{\beta}$, and then transform it back to the $X$ space. Note the in-sample error in the original space $X$ is the same as in the feature space $Z$.

The feature transform can be general, but as it becomes more complex, the dimensionality of the feature space increases, which in turn affects the guarantees on the classifier's performance on new data. The kernel trick addresses this issue by using so-called kernel functions, the mapping does not have to be explicitly computed, and computations with the mapped features remain efficient. This efficiency is obtained by noting that the Lagrangian dual formulation in equation (11.16) only involves the inner products of input features. The objective function in the transformed feature space becomes

$$\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_i' y_i y_i' \langle \Phi(x_i), \Phi(x_i') \rangle. \qquad (11.18)$$

Thus, the solution involves $\Phi(\mathbf{x})$ only through inner products. Therefore, we never need to specify the transform $\Phi(x)$, but only the function that computes inner products in the transformed space:

$$K(x,x') = \langle \Phi(x), \Phi(x) \rangle. \qquad (11.19)$$

The function $K(x, x')$ is known as the kernel function. The most commonly used choices for $K$ are polynomial kernels:

$$\mathrm{K}(x, x') = (1 + \langle x, x' \rangle)^d, \qquad (11.20)$$

and Gaussian kernels:

$$K(x,\mathrm{x}') = exp(-\|x - x'\|^2/(2\sigma^2)) \qquad (11.21)$$

By replacing the inner product in the SVM formulation in equation (11.14) by the kernel, we obtain a MML classifier in the transformed feature space defined by the kernel, which is non-linear in the original space.

### Margin, VC Dimension, and Generalization

Generalization refers to a ML model's predictive power outside the training data, that is, its ability to make the best prediction $\hat{y}$ for a new data point $x$, which is not a part of the training set. In this context, we present the Vapnik-Chervonenkis generalization theorem, which provides a bound on the ability of a model fit to a training set to generalize to new data points.

The Vapnik-Chervonenkis (VC) dimension measures the richness, or flexibility, of a classifier. The VC dimension measures how complex a classifier can be through the maximum number $k$ of data points that can be separated into all possible $2^k$ ways using the model, a process which is referred to as "shattering" the set of data points. A classifier $f(x)$ with parameter vector $\theta$ shatters a set of data points if, for all possible labels of those points, a $\theta$ exists such that $f$ correctly classifies all the data points.

The more complex the set of classifiers captured by $f$, the higher the VC dimension. For example, the VC dimension of a line in two dimensions is three, because any three points (that are not collinear) can be shattered using this model, but no set of four points can be shattered. In higher dimensions, the VC dimension of hyperplanes in $R^d$ is known to be $d + 1$. The VC dimension can be viewed as the number of a model's hypotheses. We have the following result that provides the upper bounds for the VC dimension $h$ for the set of $\gamma$-margin separating hyperplanes.

Let $\mathrm{x_i}$ be a set of points in $R^d$ that belong to a sphere of radius $Q$. Then the set of $\gamma$-margin separating hyperplanes has VC dimension $h$:

$$h \leq \min\left(\left(\frac{R}{\gamma}\right)^2, d\right) + 1. \tag{11.22}$$

Note the upper bound is inversely proportional to the margin γ, suggesting the larger the margin, the lower the VC dimension of the corresponding set of classifiers.

In evaluating a classification algorithm, we are interested in the number of errors the classifier will make when classifying *unseen*, out-of-sample, data when all we know for sure is the number of errors made on the *training*, or in-sample, data. This number cannot be computed exactly, but it can be upper-bounded using the VC dimension. The VC generalization bound gives an upper bound on the probability of a test sample being misclassified by a γ-margin hyperplane. With probability $1 - \delta$, the probability of a test sample being misclassified is

$$P_{err} \leq \frac{m}{N} + \frac{E}{2}\left(1 + \sqrt{1 + \frac{4m}{NE}}\right), \tag{11.23}$$

where

$$E = 4\,\frac{h\left(\ln\frac{2N}{h} + 1\right) - \ln\frac{\delta}{4}}{N}. \tag{11.24}$$

$N$ is the number of points in the training sample, $m$ is the number of training examples misclassified by the hyperplane, and $h$ is the VC dimension. Vapnik and Chervonenkis (1971) developed the unifying relationship between the VC dimension, sample size, and classification errors.

The first term in equation (11.23) is the proportion of misclassified data points in the training sample; the second term is a function of the model complexity, which increases with the VC dimension, $h$. Therefore, the bound on the probability of misclassifying a new data point is proportional to the VC dimension of the set of classifiers. Thus, all else being equal, a more complex classifier (one with a higher VC dimension) is likely to be a worse predictor than a simple classifier. We have also seen in equation (11.22) that the VC dimension decreases as the margin ($\gamma$) increases; this finding provides a theoretical foundation for looking for classifiers with the maximum margin, such as the MML. More generally, it motivates regularization, which is a method used to prevent model overfitting.

## Regularization

The VC generalization bound tells us that, as far as out-of-sample prediction is concerned, we should be better off fitting the data using a "simpler" model. Therefore, rather than simply finding a model that minimizes error, we introduce a term to the optimization that penalizes for model complexity, called the regularization penalty. This approach avoids overfitting by constraining the algorithm to fit the data using a simpler model.

Consider the SVM optimization problem in equation (11.13). $\xi_i$ is set to $1 - y_i(\beta^T x_i + \beta_0)$, if a data point in the training set is misclassified, and 0 if it is classified correctly. The optimization problem can be rewritten as

$$\text{minimize}_x \ \frac{1}{2}\left\|\beta\right\|^2 + C\sum_{i=1}^{N}(1 - y_i(\beta^T x_\mathrm{i} + \beta_0))^+ \qquad (11.25)$$

Here, we can view the second term, $C\sum_{i=1}^{N}(1-y_i(\beta^T x_i+\beta_0))^+$, as the loss for misclassifying a data point, and can view the first term, the inverse squared geometric margin $\frac{1}{2}\|\beta\|^2$, as the regularization penalty that helps stabilize the objective. Regularization thus helps us select the solution with the largest geometric margin, corresponding to lower VC dimension, or model complexity.

This type of regularizer, which penalizes the squared or $L_2$ norm of the parameter values, is sometimes referred to as *weight decay*, because it forces the weights to decay toward 0. Note that when applied to linear regression, it results in what is called *ridge regression* in econometrics. Similarly, the $L_1$ regularizer $|\beta|$ corresponds to lasso regression when applied to *linear regression*. With the $L_1$ regularizer, many of the less relevant features will be set exactly to 0, resulting in feature selection. Other than linear regression, regularization is also used for logistic regression, neural nets, and some matrix-decomposition methods. In the more general form of this regularization, called *Tikhonov* regularization, different penalties can be placed on different weights being large, resulting in the form $\beta^T \Gamma^T \Gamma \beta$ (Tikhonov and Arsenin, 1977).

Typically, the optimization is written in the form of minimizing in-sample errors, plus the regularization penalty, that is,

$$\text{minimize}_{\text{x}} \qquad E_i n\,(\mathbf{w}) + \lambda_C\, \mathbf{w}^T \mathbf{w} \tag{11.26}$$

for $L_2$ regularization. The functional form of the regularizer is usually chosen ahead of time, whereas the parameter $\lambda$, which determines the amount of regularization, needs to be trained. Such training is necessary because the type of regularization is usually known based on the type of data, and type of model to be fit, but the data themselves should dictate the amount of regularization. We want to pick the $\lambda$ that will result in the best *out-of-sample* prediction (the best in-sample fit is achieved using $\lambda = 0$). To determine which value of $\lambda$ leads to the best out-of-sample prediction, we train it using a *validation* method, which we describe next.

In general, regularization is necessary if the class of models is too rich for the data. Then we can combat overfitting by regularization, which penalizes the sizes of the parameters. For example, Hauser et al. (2010) estimated a very rich model of non-compensatory consideration-set formation, called disjunction of conjunctions, which allow for non-compensatory rules of the form (fuel efficient AND Toyota AND sedan) OR (powerful AND BMW AND sports car). Note the complexity of this model is exponential in product attributes and is prone to overfitting. In fact, any training data consisting of considered and non-considered products can be perfectly fit with a separate conjunction for all the features of each considered product. The authors use regularization to combat overfitting and look for "simple" decision rules to fit the data, resulting in good out-of-sample performance.

Note that, because we are explicitly penalizing for complexity, we can consider a much broader class of models (e.g., many more predictors than data points, high-degree interactions, etc.) because the regularizer will guarantee we find the best predictive model that does not overfit.

### Applications of SVM in Marketing

Because of SVM's robustness and ability to handle large, high dimensional data, it has become one of the most popular classification algorithms over the past 20 years, with applications in image recognition, text mining, and disease diagnosis. Cui and Curry (2005) introduced it to marketing and provide an excellent overview of SVM theory and implementations. They also compare the predictive performance of SVM to that of the multinomial logit model on simulated choice data, and demonstrate SVM performs better, particularly when data are noisy and products have many attributes (i.e., high dimensionality). They also see that when predicting choices from larger choice sets, SVM more significantly outperforms the multinomial logit model. Although both methods' predictive ability falls as the size of the choice set increases, because the first-choice prediction task becomes more difficult, the decline is much steeper for multinomial logit than for SVM. Evgeniou et al. (2005) present and test a family of preference models, including highly non-linear ones, which are estimated using SVM methodology. The estimation procedure uses regularization to prevent the complex models from overfitting that is similar to that of SVM. For linear utility models, they find the SVM significantly outperforms logistic regression on out-of-sample hit rates. The improvement of using SVM versus logistic regression is particularly large when the choice design is random; the methods perform approximately equally well for a balanced orthogonal choice design. Similar to Cui and Curry (2005), they find SVM performs significantly better when noise increases, suggesting SVM is more robust to noise. Next, they test the performance of the methods on utility models that involve interactions among attributes. For these models, they show SVM performs similar to hierarchical Bayes (HB) estimation of a correctly specified nonlinear model. However, SVM better captures the nonlinear parts of the model. Additionally, SVM can handle more complex models with more interactions than HB can, because it is computationally efficient.

Evgeniou et al. (2007) extend SVM to develop a framework for modeling choice data for multi-attribute products, which allows the capturing of respondent heterogeneity and the pooling of choice data across respondents. The attribute partworths are shrunk to the mean with regularization parameters that are trained using cross-validation.

More recently, Huang and Luo (2015) used fuzzy SVM, an extension of SVM methodology, for preference elicitation of complex products with a large number of features. They proposed an adaptive question-selection process using fuzzy SVM active learning to adaptively select each subsequent question. They showed that, due to the convex nature of SVM optimization, such an approach is computationally efficient for preference elicitation of complex products on the fly.

Another extension is the latent-class SVM model, which allows the use of latent variables within SVM. Liu and Dzyabura (2016) develop an algorithm for estimating multi-taste consumer preferences by building on the convex–concave procedure used to estimate latent-class SVM while capturing respondent heterogeneity. They show their model's prediction is better than single-taste benchmarks.

## COMMON ISSUES IN ML METHODS

### Training, Validation, and Testing

Dividing the data into separate sets for the purpose of training, validation, and testing is common. Researchers use the training data to estimate models, the validation data to choose a model, and the testing data to evaluate how well the model performs. We discuss below the reasons for splitting the data into the constituent parts and issues related to this framework.

We first examine the need for using a testing data set. As discussed earlier, the goal of ML techniques is to provide the best out-of-sample predictions as opposed to simply improving the model fit on the sample data set. Given this need, the predictive ability of ML techniques is evaluated by first constructing a model on a training data set and then evaluating its accuracy on a testing data set, whose corresponding data items weren't included in the training data set. This approach provides a meaningful estimate of the expected accuracy of the model on out-of-sample data.

Let us now examine the need for having a validation data set. Consider an ML technique that trains multiple models on a training set $S$, and picks the model that provides the best in-sample accuracy (the lowest error on the set $S$). This approach will prefer larger and more detailed models to less detailed ones, even though the less detailed ones might have better predictive performance on out-of-sample data. For example, if we are approximating a variable $y$ using a polynomial function applied on inputs $x$, then, if we determine the order of the polynomial based on the accuracy of prediction on the training set $S$, we would always pick a very high-degree, high-variance polynomial model that overfits the data in $S$ and may, as a consequence, perform poorly on the testing data. To address this issue, *cross-validation* splits the input data set $S$ into two components: $S_t$ (training) and $S_v$ (validation). It then uses the training set $S_t$ to generate candidate models, and then picks a model that performs best on $S_v$ as opposed to basing the decision solely on $S_t$ fit. Cross-validation thus ensures the chosen model does not overfit $S_t$ and performs well on out-of-sample data.

The cross-validation enhancement can be applied to any ML algorithm. For example, in the case of boosted MART, cross-validation typically works as follows. After each tree is computed based on $S_t$ and added to the MART ensemble, the MART is evaluated on the validation data set $S_v$. Although the additional tree would have improved the accuracy on $S_t$, it might not have necessarily improved the accuracy on $S_v$. So the algorithm could introduce a stopping rule for MART construction that terminates the MART construction when *k* consecutive iterations (or trees) have not yielded accuracy improvements on $S_v$. The algorithm would then select the MART (or the output of an intermediate step) that yielded the best accuracy on $S_v$ as the best model.

Note that validation is not free. The algorithm has to split the input data set into two smaller components and use only one of them for the purpose of training. This procedure leaves fewer samples to train a model, resulting in a suboptimal model. However, the accuracy gains realized from avoiding overfitting typically trump the reduction in the size of the training data, particularly for large data sets. As a consequence, most ML practitioners use cross-validation as part of their modeling toolkit.

## Additional Techniques to Avoid Overfitting

In addition to the cross-validation method discussed above, additional techniques exist for reducing the effect of overfitting. We now discuss some of those techniques.

### Regularization

Because simple models tend to work better for out-of-sample forecasts, ML researchers have come up with ways to penalize models for excessive complexity. This process is known in ML as "regularization" or "complexity control," and we will give examples when we discuss specific methods. Although economists also tend to prefer simpler models (for the same reason), they have not been as explicit about quantifying complexity costs.

#### Tuning regularization parameters using cross-validation

If we have an explicit numeric measure of model complexity, we can view it as a parameter that can be *tuned* to produce the best out-of-sample predictions. The standard way to tune a parameter is to use *k*-fold cross-validation:

(1) Divide the data into *k* equal subsets (folds) and label them $s = 1,. . .,k$. Start with $s = 1$.
(2) Pick an initial value for the tuning parameter.
(3) Fit your model using the $k - 1$ subsets other than *s*.

(4) Predict the outcome variable for subset *s* and measure the associated loss.
(5) Stop if $s = k$; otherwise, increment s by 1 and go to step 2.

After cross-validation, we end up with *k* values of the tuning parameter and the associated loss, which you can then examine to choose an appropriate value for the tuning parameter. Even if no tuning parameter exists, using cross-validation to report goodness-of-fit measures is generally a good idea, because it measures out-of-sample performance, which is generally more meaningful than in-sample performance (such as $R^2$). Using the test–train cycle and cross-validation in ML is common, particularly when large data sets are available. If the data are large enough that a model can be estimated on a subset of the data, using separate training and testing sets provides a more realistic measure of prediction performance.

**Feature selection**

Feature selection is a standard step in ML settings that involve supervised learning (Guyon and Elisseeff, 2003). Feature selection typically provides a faster and more computationally efficient model by eliminating less relevant features with minimal loss in accuracy. It is thus particularly relevant for training large data sets that are typical in various target application settings. Feature selection also provides more comprehensible models that offer a better understanding of the underlying data-generating process. When the data sets are modest in size and the number of features is large, feature selection can actually improve the predictive accuracy of the model by eliminating irrelevant features whose inclusion often results in overfitting. Many ML algorithms, including neural networks, decision trees, CART, and naive Bayes learners, have been shown to have significantly worse accuracy when trained on small data sets with superfluous features (Duda and Hart, 1973; Aha et al., 1991; Breiman et al., 1984; Quinlan, 1993).

The goal of feature selection is to find the smallest set of features that can provide a fixed predictive accuracy. In principle, this problem is straightforward because it simply involves an exhaustive search of the feature space. However, with even a moderately large number of features, an exhaustive search is practically impossible. With *F* features, an exhaustive search requires $2^F$ runs of the algorithm on the training data set, which is exponentially increasing in *F*. In fact, this problem is known to be NP-hard (Amaldi and Kann, 1998).

The wrapper method addresses this problem by using a greedy algorithm (Kohavi and John 1997). Wrappers can be categorized into two types—*forward selection* and *backward elimination*. In forward selection, features are progressively added until a desired prediction accuracy is reached or until the incremental improvement is very small. By contrast, a backward-elimination wrapper starts with all the features and sequentially eliminates the least valuable features. Both wrappers are greedy in the sense that they do not revisit former decisions to include (in forward selection) or exclude features (in backward elimination). More importantly, they are "black box" techniques in the sense that they can work with any ML algorithm by invoking them without needing to understand their internal structure.

To enable a wrapper algorithm, the researcher needs to specify a selection as well as a stopping rule. A commonly used and robust selection rule is the *best-first* selection rule (Ginsberg, 1993), wherein the most promising node is selected at every decision point. For example, in a forward-selection algorithm with 10 features, at the first node, this algorithm considers 10 versions of the model (each with one of the features added) and then picks the feature whose addition offers the highest prediction accuracy. The process continues until a stopping-rule condition is satisfied. A stopping rule consists of a cut-off point for the incremental gain obtained at each step of the algorithm, and when the incremental gain is less than this cut-off point, the feature-selection process ends and emits the currently selected set of features.

Wrappers offer many advantages. First, they are agnostic to the underlying learning algorithm and the accuracy metric used for evaluating the predictor. Second, greedy wrappers have been shown to be robust to overfitting and computationally advantageous (Reunanen, 2003); the resulting model requires fewer features to be computed during testing, and the testing-classification process itself is faster because the model is compact.

## CONCLUSION

ML methods are gaining traction in both marketing practice and academic research. They provide a set of valuable tools to help us increase the out-of-sample performance of marketing models and thereby improve the quality of marketing decisions. In this chapter, we presented a brief overview of the two most commonly used ML methods, decision trees and SVM, as well as a discussion of their applications in marketing. With the advent of large data sets, focus on real-time performance, and the availability of cheap and fast computing (e.g., Amazon EC2), we hope marketers can use ML techniques to answer a new set of exciting and challenging substantive questions going forward.

## NOTES

1. The authors thank Bryan Bollinger, Shahryar Doosti, Theodoros Evgeniou, John Hauser, Panos Ipeirotis, Lan Luo, Eugene Pavlov, Omid Rafieian, and Amin ZadKazemi for their comments.
2. For a detailed discussion of the roles of causal, predictive, and descriptive research in social sciences, please see Shmueli (2010).
3. In a comparison of logistic regression and decision trees, Perlich et al. (2003) examined several data sets. Taking different sized subsamples of the data, they estimated both models using *learning curves*, that is, how the model's predictive accuracy improves as the sample size increases. They found that logistic regressions work better for smaller data sets, and trees work better for larger data sets. Interestingly, they found this pattern holds even for training sets from the same domain.

## REFERENCES

Aha, W., D. Kibler, and M. K. Albert. Instance-based Learning Algorithms. *Machine Learning*, 6(1): 37–66, 1991.

Amaldi, E. and V. Kann. On the Approximability of Minimizing Nonzero Variables or Unsatisfied Relations in Linear Systems. *Theoretical Computer Science*, 209(1): 237–260, 1998.

Breiman, L., J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

Caruana, R. and A. Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ACM, 2006, 161–168.

Cui, D. and D. Curry. Prediction in Marketing using the Support Vector Machines. *Marketing Science*, 24(4): 595–615, 2005.

Duda, R. O. and P. E. Hart. *Pattern Recognition and Scene Analysis*, New York: Wiley, 1973.

Evgeniou, T., C. Boussios, and G. Zacharia. Generalized Robust Conjoint Estimation. *Marketing Science*, 24(3): 415–429, 2005.

Evgeniou, T., M. Pontil, and O. Toubia. A Convex Optimization Approach to Modeling Consumer Heterogeneity in Conjoint Estimation. *Marketing Science*, 26(6): 805–818, 2007.

Ginsberg, M. *Essentials of Artificial Intelligence*. San Mateo, CA: Morgan Kaufmann, 1993.

Guyon, I. and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Hastie, T., R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The Elements of Statistical Learning*, Vol. 2. New York: Springer, 2009.

Hauser, J. R., O. Toubia, T. Evgeniou, R. Befurt, and D. Dzyabura. Disjunctions of Conjunctions, Cognitive Simplicity, and Consideration Sets. *Journal of Marketing Research*, 47(3): 485–496, 2010.

Huang, D. and L. Luo. Consumer Preference Elicitation of Complex Products using Fuzzy Support Vector Machine Active Learning. *Marketing Science*, 35(3): 445–464, 2015.

Hyafil, L. and R. L. Rivest. Constructing Optimal Binary Decision Trees is NP-complete. *Information Processing Letters*, 5(1): 15–17, 1976.

Kohavi, R. and G. H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1): 273–324, 1997.

Lemmens, A. and C. Croux. Bagging and Boosting Classification Trees to Predict Churn. *Journal of Marketing Research*. 43(2): 276–286, 2006.

Liu, L. and D. Dzyabura. Capturing Multi-taste Preferences: A Machine Learning Approach. *Working Paper*, 2016.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.

Perlich, C., F. Provost, and J. S. Simonoff. Tree Induction vs. Logistic Regression: A Learning-curve Analysis. *Journal of Machine Learning Research*, 4:211–255, 2003.

Quinlan, J. R., *C4. 5: Programs for Machine Learning*, Volume 1. San Mateo, CA: 1993.

Rafieian, O. and H. Yoganarasimhan. Targeting and Privacy in Mobile Advertising. *Working Paper*, 2017.

Reunanen, J. Overfitting in Making Comparisons between Variable Selection Methods. *Journal of Machine Learning Research*, 3:1371–1382, 2003.

Schapire, R. E. The Strength of Weak Learnability. *Machine Learning*, 5(2): 197–227, 1990.

Shmueli, G. To Explain or to Predict? *Statistical Science*, 25(3): 289–310, 2010.

Tikhonov, A. N. and V. Y. Arsenin. *Solutions of Ill-posed Problems*. Washington, DC: Winston, 1977.

Vapnik, V. N. and A. Y. Chervonenkis. On the Uniform Convergence of Relative Frequencies of Events to their Probabilities. *Theory of Probability & Its Applications*, 16(2): 264–280, 1971.

Yoganarasimhan, H. Search Personalization using Machine Learning. *Working Paper*, 2017.