Chapter 18
Turning Points and Classification

Jeremy Piger

Abstract Economic time-series data is commonly categorized into a discrete number of persistent regimes. I survey a variety of approaches for real-time prediction of these regimes and the turning points between them, where these predictions are formed in a data-rich environment. I place particular emphasis on supervised machine learning classification techniques that are common to the statistical classification literature, but have only recently begun to be widely used in economics. I also survey Markov-switching models, which are often used for unsupervised classification of economic data. The approaches surveyed are computationally feasible when applied to large datasets, and the machine learning algorithms employ regularization and cross-validation to prevent overfitting in the face of many predictors. A subset of the approaches conduct model selection automatically in forming predictions. I present an application to real-time identification of U.S. business cycle turning points based on a wide dataset of 136 macroeconomic and financial time-series.

18.1 Introduction

It is common in studies of economic time series for each calendar time period to be categorized as belonging to one of some fixed number of recurrent regimes. For example, months and quarters of macroeconomic time series are separated into periods of recession and expansion, and time series of equity returns are divided into bull vs. bear market regimes. Other examples include time series measuring the banking sector, which can be categorized as belonging to normal vs. crises regimes, and time series of housing prices, for which some periods might be labeled as arising from a 'bubble' regime. A key feature of these regimes in most economic settings is that they are thought to be persistent, meaning the probability of each regime occurring increases once the regime has become active.

Jeremy Piger
Department of Economics, University of Oregon, Eugene, OR 97403, e-mail: jpiger@uoregon.edu

In many cases of interest, the regimes are never explicitly observed. Instead, the historical timing of regimes is inferred from time series of historical economic data. For example, in the United States, the National Bureau of Economic Research (NBER) Business Cycle Dating Committee provides a chronology of business cycle expansion and recession dates developed from study of local minima and maxima of many individual time series. Because the NBER methodology is not explicitly formalized, a literature has worked to develop and evaluate formal statistical methods for establishing the historical dates of economic recessions and expansions in both U.S. and international data. Examples include Hamilton (1989), Vishwakarma (1994), Chauvet (1998), Harding and Pagan (2006), Fushing, Chen, Berge, and Jordá (2010), Berge and Jordá (2011) and Stock and Watson (2014).

In this chapter I am also interested in determining which regime is active based on information from economic time series. However, the focus is on real-time prediction rather than historical classification. Specifically, the ultimate goal will be to identify the active regime toward the end of the observed sample period (nowcasting) or after the end of the observed sample period (forecasting). Most of the prediction techniques I consider will take as given a historical categorization of regimes, and will use this categorization to learn the relationship between predictor variables and the occurrence of alternative regimes. This learned relationship will then be exploited in order to classify time periods that have not yet been assigned to a regime. I will also be particularly interested in the ability of alternative prediction techniques to identify turning points, which mark the transition from one regime to another. When regimes are persistent, so that there are relatively few turning points in a sample, it is quite possible for a prediction technique to have good average performance for identifying regimes, but consistently make errors in identifying regimes around turning points.

Consistent with the topic of this book, I will place particular emphasis in this chapter on the case where regime predictions are formed in a data-rich environment. In our specific context, this environment will be characterized by the availability of a large number of time-series predictor variables from which we can infer regimes. In typical language, our predictor dataset will be a 'wide' dataset. Such datasets create issues when building predictive models, since it is difficult to exploit the information in the dataset without overfitting, which will ultimately lead to poor out-of-sample predictions.

The problem of regime identification discussed above is an example of a statistical classification problem, for which there is a substantial literature outside of economics. I will follow the tradition of that literature and refer to the regimes as 'classes,' the task of inferring classes from economic indicators as 'classification,' and a particular approach to classification as a 'classifier.' Inside of economics, there is a long history of using parametric models, such as a logit or probit, as classifiers. For example, many studies have used logit and probit models to predict U.S. recessions, where the model is estimated over a period for which the NBER business cycle chronology is known. A small set of examples from this literature include Estrella and Mishkin (1998), Estrella, Rodrigues, and Schich (2003), Kauppi and Saikkonen (2008), Rudebusch and Williams (2009) and Fossati (2016). Because

they use an available historical indicator of the class to estimate the parameters of the model, such approaches are an example of what is called a 'supervised' classifier in the statistical classification literature. This is in contrast to 'unsupervised classifiers,' which endogenously determine clustering of the data, and thus endogenously determine the classes. Unsupervised classifiers have also been used for providing real-time nowcasts and forecasts of U.S. recessions, with the primary example being the Markov-switching framework of Hamilton (1989). Chauvet (1998) proposes a dynamic factor model with Markov-switching (DFMS) to identify expansion and recession phases from a group of coincident indicators, and Chauvet and Hamilton (2006), Chauvet and Piger (2008) and Camacho, Perez-Quiros, and Poncela (2018) evaluate the performance of variants of this DFMS model to identify NBER turning points in real time. An important feature of Markov-switching models is that they explicitly model the persistence of the regime, by assuming the regime indicator follows a Markov process.

Recently, a number of authors have applied machine learning techniques commonly used outside of economics to classification problems involving time-series of economic data. As an example, Qi (2001), Ng (2014), Berge (2015), Davig and Smalter Hall (2016), Garbellano (2016) and Giusto and Piger (2017) have applied machine learning techniques such as artificial neural networks, boosting, naïve bayes, and learning vector quantization to forecasting and nowcasting U.S. recessions, while Ward (2017) used random forests to identify periods of financial crises. These studies have generally found improvements from the use of the machine learning algorithms over commonly used alternatives. For example, Berge (2015) finds that the performance of boosting algorithms improves on equal weight model averages of recession forecasts produced by logistic models, while Ward (2017) finds a similar result for forecasts of financial crises produced by a random forest.

Machine learning algorithms are particularly attractive in data-rich settings. Such algorithms typically have one or more 'regularization' mechanisms that trades off in-sample fit against model complexity, which can help prevent overfitting. These algorithms are generally also fit using techniques that explicitly take into account out-of-sample performance, most typically using cross validation. This aligns the model fitting stage with the ultimate goal of out-of-sample prediction, which again can help prevent overfitting. A number of these methods also have built in mechanisms to conduct model selection jointly with estimation in a fully automated procedure. This provides a means to target relevant predictors from among a large set of possible predictors. Finally, these methods are computationally tractable, making them relatively easy to apply to large datasets.

In this chapter, I survey a variety of popular off-the-shelf supervised machine learning classification algorithms for the purpose of classifying economic regimes in real time using time-series data. Each classification technique will be presented in detail, and its implementation in the `R` programming language will be discussed.[1] Particular emphasis will be placed on the use of these classifiers in data-rich environments. I will also present DFMS models as an alternative to the machine learning

[1] http://www.R-project.org/

classifiers in some settings. Finally, each of the various classifiers will be evaluated for their real-time performance in identifying U.S. business cycle turning points from 2000 to 2018.

As discussed above, an existing literature in economics uses parametric logit and probit models to predict economic regimes. A subset of this literature has utilized these models in data-rich environments. For example, Owyang, Piger, and Wall (2015) and Berge (2015) use model averaging techniques with probit and logit models to utilize the information in wide data sets, while Fossati (2016) uses principal components to extract factors from wide datasets to use as predictor variables in probit models. I will not cover these techniques here, instead opting to provide a resource for machine learning methods, which hold great promise, but have received less attention in the literature to date.

The remainder of this chapter proceeds as follows. Section 18.2 will formalize the forecasting problem we are interested in and describe metrics for evaluating class forecasts. Section 18.3 will then survey the details of a variety of supervised machine learning classifiers and their implementation, while section 18.4 will present details of DFMS models for the purpose of classification. Section 18.5 will present an application to real-time nowcasting of U.S. business cycle turning points. Section 18.6 concludes.

18.2 The Forecasting Problem

In this section I lay out the forecasting problem of interest, as well as establish notation used for the remainder of this chapter. I also describe some features of economic data that should be recognized when conducting a classification exercise. Finally, I detail common approaches to evaluating the quality of class predictions.

18.2.1 Real-time classification

Our task is to develop a prediction of whether an economic entity in period $t+h$ is (or will be) in each of a discrete number (C) of classes. Define a discrete variable $S_{t+h} \in \{1,\ldots,C\}$ that indicates the active class in period $t+h$. It will also be useful to define C binary variables $S^c_{t+h} = I(S_{t+h} = c)$, where $I() \in \{0,1\}$ is an indicator function, and $c = 1,\ldots,C$.

Assume that we have a set of N predictors to draw inference on S_{t+h}. Collect these predictors measured at time t in the vector X_t, with an individual variable inside this vector labeled $X_{j,t}$, $j = 1,\ldots,N$. Note that X_t can include both contemporaneous values of variables as well as lags. I define a classifier as $\widehat{S}^c_{t+h}(X_t)$, where this classifier produces a prediction of S^c_{t+h} conditional on X_t. These predictions will take the form of conditional probabilities of the form $\Pr\left(S_{t+h} = c|X_t\right)$. Note that a user of these predictions may additionally be interested in binary predictions of

S^c_{t+h}. To generate a binary prediction we would combine our classifier $\widehat{S}^c_{t+h}(X_t)$ with a rule, $L()$, such that $L\left(\widehat{S}^c_{t+h}(X_t)\right) \in \{0,1\}$. Finally, assume we have available T observations on X_t and S_{t+h}, denoted as $\{X_t, S_{t+h}\}_{t=1}^T$. I will refer to this in-sample period as the 'training sample.' This training sample is used to determine the parameters of the classifier, and I refer to this process as 'training' the classifier. Once trained, a classifier can then be used to forecast S^c_{t+h} outside of the training sample. Specifically, given an X_{T+q}, we can predict S^c_{T+q+h} using $\widehat{S}^c_{T+q+h}\left(X_{T+q}\right)$ or $L\left(\widehat{S}^c_{T+q+h}\left(X_{T+q}\right)\right)$.

I will also be interested in the prediction of turning points, which mark the transition from one class to another. The timely identification of turning points in economic applications is often of great importance, as knowledge that a turning point has already occurred, or will in the future, can lead to changes in behavior on the part of firms, consumers, and policy makers. As an example, more timely information suggesting the macroeconomy has entered a recession phase should lead to quicker action on the part of monetary and fiscal policymakers, and ultimately increased mitigation of the effects of the recession. In order to predict turning points we will require another rule to convert sequences of $\widehat{S}^c_{t+h}(X_t)$ into turning point predictions. Of course, how cautious one is in converting probabilities to turning point predictions is determined by the user's loss function, and in particular the relative aversion to false positives. In the application presented in Section 18.5, I will explore the real-time performance of a specific strategy for nowcasting turning points between expansion and recession phases in the macroeconomy.

18.2.2 Classification and economic data

As is clear from the discussion above, we are interested in this chapter in classification in the context of time-series data. This is in contrast to much of the broader classification literature, which is primarily focused on classification in cross-sectional datasets, where the class is reasonably thought of as independent across observations. In most economic time series, the relevant class is instead characterized by time-series persistence, such that a class is more likely to continue if it is already operational than if it isn't. In this chapter, I survey a variety of off-the-shelf machine learning classifiers, most of which do not explicitly model persistence in the class indicator. In cases where the forecast horizon h is reasonably long, ignoring persistence of the class is not likely to be an issue, as the dependence of the future class on the current class will have dissipated. However, in short horizon cases, such as that considered in the application presented in Section 18.5, this persistence is more likely to be important. To incorporate persistence into the machine learning classifiers' predictions, I follow Giusto and Piger (2017) and allow for lagged values

to enter the X_t vector. Lagged predictor variables will provide information about lagged classes, which should improve classification of future classes.[2]

Economic data is often characterized by 'ragged edges,' meaning that some values of the predictor variables are missing in the out-of-sample period (Camacho et al. (2018)). This generally occurs due to differences in the timing of release dates for different indicators, which can leave us with only incomplete observation of the vector X_{T+j}. There are a variety of approaches that one can take to deal with these missing observations when producing out-of-sample predictions. A simple, yet effective, approach is to use k nearest neighbors (kNN) imputation to impute the missing observations. This approach imputes the missing variables based on fully observed vectors from the training sample that are similar on the dimension of the non-missing observations. kNN imputation is discussed in more detail in Section 18.3.3.

18.2.3 Metrics for evaluating class forecasts

In this section I discuss metrics for evaluating the performance of classifiers. Suppose we have a set of $\widetilde{T}$ class indicators, S^c_{t+h}, and associated classifier predictions, $\widehat{S}^c_{t+h}(X_t)$, where $t \in \Theta$. Since $\widehat{S}^c_{t+h}(X_t)$ is interpreted as a probability, an obvious metric to evaluate these predictions is Brier's Quadratic Probability Score (QPS), which is the analogue of the traditional mean squared error for discrete data:

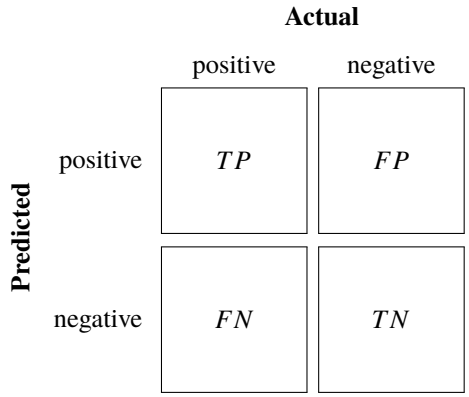
$$QPS = \frac{1}{\widetilde{T}} \sum_{t \in \Theta} \sum_{c=1}^{C} (S^c_{t+h} - \widehat{S}^c_{t+h}(X_t))^2$$

The QPS is bounded between 0 and 2, with smaller values indicating better classification ability.

As discussed above, in addition to predictions that are in the form of probabilities, we are often interested in binary predictions produced as $L\left(\widehat{S}^c_{t+h}(X_t)\right)$. In this case, there are a variety of commonly used metrics to assess the accuracy of classifiers. In describing these, it is useful to restrict our discussion to the two class case, so that $c \in \{1, 2\}$.[3] Also, without loss of generality, label $c = 1$ as the 'positive' class and $c = 2$ as the 'negative' class. We can then define a confusion matrix:

[2] When converting $\widehat{S}^c_{t+h}(X_t)$ into turning point predictions, one might also consider conversion rules that acknowledge class persistence. For example, multiple periods of elevated class probabilities could be required before a turning point into that class is predicted.

[3] Generalizations of these metrics to the multi-class case generally proceed by considering each class against all other classes in order to mimic a two class problem.



where TP is the number of true positives, defined as the number of instances of $c = 1$ that were classified correctly as $c = 1$, and FP indicates the number of false positives, defined as the instances of $c = 2$ that were classified incorrectly as $c = 1$. FN and TN are defined similarly.

A number of metrics of classifier performance can then be constructed from this confusion matrix. The first is *Accuracy*, which simply measures the proportion of periods that are classified correctly:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Of course, *Accuracy* is strongly affected by the extent to which classes are balanced in the sample period. If one class dominates the period under consideration, then it is easy to have very high accuracy by simply always forecasting that class with high probability. In many economic applications, classes are strongly unbalanced, and as a result the use of *Accuracy* alone to validate a classifier would not be recommended. Using the confusion matrix we can instead define accuracy metrics for each class. Specifically, the *true positive rate*, or TPR, gives us the proportion of instances of $c = 1$ that were classified correctly:

$$TPR = \frac{TP}{TP + FN}$$

while the *true negative rate*, or TNR, gives us the proportion of instances of $c = 2$ that were classified correctly:

$$TNR = \frac{TN}{TN + FP}$$

It is common to express the information in TNR as the *false positive rate*, which is given by $FPR = 1 - TNR$.[4]

It is clear that which of these metrics is of primary focus depends on the relative preferences of the user for true positives vs. false positives. Also, it should be

[4] In the classification literature, *TPR* is referred to as the *sensitivity* and *TNR* as the *specificity*.

remembered that the confusion matrix, and the metrics defined from its contents, are dependent not just on the classifier $\widehat{S}^c_{t+h}(X_t)$, but also on the rule L used to convert this classifier to binary outcomes. In many cases, these rules are simply of the form:

$$L\left(\widehat{S}^c_{t+h}(X_t)\right) = \begin{cases} 1 & \text{if } \widehat{S}^c_{t+h}(X_t) > d \\ 2 & \text{if } \widehat{S}^c_{t+h}(X_t) \leq d \end{cases}$$

such that $c = 1$ is predicted if $\widehat{S}^c_{t+h}(X_t)$ rises above the threshold d, and $0 \leq d \leq 1$ since our classifier is a probability. A useful summary of the performance of a classifier is provided by the 'receiver operator characteristic' (ROC) curve, which is a plot of combinations of TPR (y-axis) and FPR (x-axis), where the value of d is varied to generate the plot. When $d = 1$ both TPR and FPR are zero, since both TP and FP are zero if class $c = 1$ is never predicted. Also, $d = 0$ will generate TPR and FPR that are both one, since FN and TN will be zero if class $c = 1$ is always predicted. Thus, the ROC curve will always rise from the origin to the (1,1) ordinate. A classifier for which X_t provides no information regarding S^c_{t+h}, and is thus constant, will have $TPR = FPR$, and the ROC curve will lie along the 45 degree line. Classifiers for which X_t does provide useful information will have a ROC curve that lies above the 45 degree line. Figure 18.1 provides an example of such a ROC curve. Finally, suppose we have a perfect classifier, such that there exists a value of $d = d^*$ where $TPR = 1$ and $FPR = 0$. For all values of $d \geq d^*$, the ROC curve will be a vertical line on the y-axis from (0,0) to (0,1), where for values of $d \leq d^*$, the ROC curve will lie on a horizontal line from (0,1) to (1,1).

As discussed in Berge and Jordá (2011), the area under the ROC curve (AUROC) can be a useful measure of the classification ability of a classifier. The AUROC for the 45 degree line, which is the ROC curve for the classifier when X_t has no predictive ability, is 0.5. The AUROC for a perfect classifier is 1. In practice, the AUROC will lie in between these extremes, with larger values indicating better classification ability.

In the application presented in Section 18.5, I consider both the QPS and the AUROC to assess out-of-sample classification ability. I will also evaluate the ability of these classifiers to forecast turning points. In this case, given the relative rarity of turning points in most economic applications, it seems fruitful to evaluate performance through case studies of the individual turning points. Examples of such a case study will be provided in Section 18.5.

18.3 Machine Learning Approaches to Supervised Classification

In this section I will survey a variety of off-the-shelf supervised machine learning classifiers. The classifiers I survey have varying levels of suitability for data-rich environments. For all the classifiers considered, application to datasets with many predictors is computationally feasible. Some of the classifiers also have built in mechanisms to identify relevant predictors, while others use all predictor variables

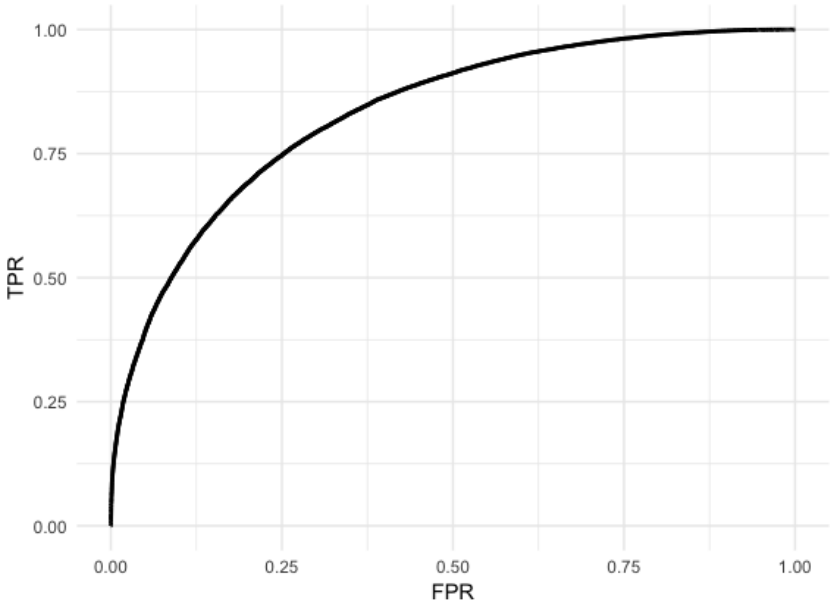


Fig. 18.1 Example Receiver Operator Characteristic (ROC) Curve

equally. Where relevant, I will discuss the wide-data attributes of each classifier below.

All of the classifiers that I discuss in this section involve various specification choices that must be set in order to implement the classifier. In some cases, these choices involve setting the value of a parameter, while in others they may involve the choice between two or more variants of the classifier. Following the machine learning literature, I refer to these various choices as tuning parameters. While these tuning parameters can be set *a priori*, in this chapter I instead implement the commonly used procedure of cross validation to set the tuning parameters automatically in a data-based way. In the following subsection I briefly describe cross validation, before moving to discussions of the individual classifiers.

18.3.1 Cross validation

The central idea of cross validation is to randomly partition the full training sample into a new training sample and a (non-overlapping) evaluation sample. For specific values of the tuning parameters, the classifier is trained on the partition of data labeled the training sample, and is then used to classify the partition labeled the evaluation sample. The performance of the classifier on the evaluation sample is recorded for each point in a grid for the tuning parameters, and the values of the

tuning parameters with the best performance classifying the evaluation sample is selected. These optimal values for the tuning parameters are then used to train the classifier over the full training sample. Performance on the evaluation sample is generally evaluated using a scalar metric for classification performance. For example, in the application presented in Section 18.5, I use the AUROC for this purpose.

In k-fold cross validation, k of these partitions (or 'folds') are randomly generated, and the performance of the classifier for specific tuning parameter values is averaged across the k evaluation samples. A common value for k, which is the default in many software implementations of k-fold cross validation, is $k = 10$. One can increase robustness by repeating k-fold cross validation a number of times, and averaging performance across these repeats. This is known as repeated k-fold cross validation. Finally, in settings with strongly unbalanced classes, which is common in economic applications, it is typical to sample the k partitions such that they reflect the percentage of classes in the full training sample. In this case, the procedure is labeled stratified k-fold cross validation.

Cross validation is an attractive approach for setting tuning parameters because it aligns the final objective, good out-of-sample forecasts, with the objective used to determine tuning parameters. In other words, tuning parameters are given a value based on the ability of the classifier to produce good out-of-sample forecasts. This is in contrast to traditional estimation, which sets parameters based on the in-sample fit of the model. Cross validation is overwhelmingly used in machine learning algorithms for classification, and it can be easily implemented for a wide variety of classifiers using the **`caret`** package in `R`.

18.3.2 Naïve Bayes

We begin our survey of machine learning approaches to supervised classification with the **Naïve Bayes** (NB) classifier. NB is a supervised classification approach that produces a posterior probability for each class based on application of Bayes Rule. NB simplifies the classification problem considerably by assuming that inside of each class, the individual variables in the vector X_t are independent of each other. This conditional independence is a strong assumption, and would be expected to be routinely violated in economic datasets. Indeed, it would be expected to be violated in most datasets, which explains the 'naïve' moniker. However, despite this strong assumption, the NB algorithm works surprisingly well in practice. This is primarily because what is generally needed for classification is not exact posterior probabilities of the class, but only reasonably accurate approximate rank orderings of probabilities. Two recent applications in economics include Garbellano (2016), who used a NB classifier to nowcast U.S. recessions and expansions in real time, and Davig and Smalter Hall (2016), who used the NB classifier, including some extensions, to predict U.S. business cycle turning points.

To describe the NB classifier I begin with Bayes rule:

$$\Pr\left(S_{t+h} = c|X_t\right) \propto f\left(X_t|S_{t+h} = c\right)\Pr\left(S_{t+h} = c\right) \ . \tag{18.1}$$

In words, Bayes Rule tells us that the posterior probability that S_{t+h} is in phase c is proportional to the probability density for X_t conditional on S_{t+h} being in phase c multiplied by the unconditional (prior) probability that S_{t+h} is in phase c.

The primary difficulty in operationalizing (18.1) is specifying a model for X_t to produce $f\left(X_t|S_{t+h} = c\right)$. The NB approach simplifies this task considerably by assuming that each variable in X_t is independent of each other variable in X_t, conditional on $S_{t+h} = c$. This implies that the conditional data density can be factored as follows:

$$f\left(X_t|S_{t+h} = c\right) = \prod_{j=1}^{N} f_j\left(X_{j,t}|S_{t+h} = c\right) \ ,$$

where $X_{j,t}$ is one of the variables in X_t. Equation (18.1) then becomes:

$$\Pr\left(S_{t+h} = c|X_t\right) \propto \left[\prod_{j=1}^{N} f_j\left(X_{j,t}|S_{t+h} = c\right)\right]\Pr\left(S_{t+h} = c\right) \ . \tag{18.2}$$

How do we set $f_j\left(X_{j,t}|S_{t+h} = c\right)$? One approach is to assume a parametric distribution, where a typical choice in the case of continuous X_t is the normal distribution:

$$X_{j,t}|S_{t+h} = c \sim N\left(\mu_{j,c}, \sigma^2_{j,c}\right) \ ,$$

where $\mu_{j,c}$ and $\sigma^2_{j,c}$ are estimated from the training sample. Alternatively, we could evaluate $f_j\left(X_{j,t}|S_{t+h} = c\right)$ non-parametrically using a kernel density estimator fit to the training sample. In our application of NB presented in Section 18.5, I treat the choice of whether to use a normal distribution or a kernel density estimate as a tuning parameter.

Finally, equation (18.2) produces an object that is proportional to the conditional probability $\Pr\left(S_{t+h} = c|X_t\right)$. We can recover this conditional probability exactly as:

$$\Pr\left(S_{t+h} = c|X_t\right) = \frac{\left[\prod_{j=1}^{N} f_j\left(X_{j,t}|S_{t+h} = c\right)\right]\Pr\left(S_{t+h} = c\right)}{\sum_{c=1}^{C}\left(\left[\prod_{j=1}^{N} f_j\left(X_{j,t}|S_{t+h} = c\right)\right]\Pr\left(S_{t+h} = c\right)\right)} \ .$$

Our NB classifier is then $\widehat{S}^c_{t+h}\left(X_t\right) = \Pr\left(S_{t+h} = c|X_t\right)$.

The NB classifier has a number of advantages. First, it is intuitive and interpretable, directly producing posterior probabilities of each class. Second, it is easily scalable to large numbers of predictor variables, requiring a number of parameters linear to the number of predictors. Third, since only univariate properties of a predictor in each class are estimated, the classifier can be implemented with relatively

small amounts of training data. Finally, ignoring cross-variable relationships guards against overfitting the training sample data.

A primary drawback of the NB approach is that it ignores cross-variable relationships potentially valuable for classification. Of course, this drawback is also the source of the advantages mentioned above. Another drawback relates to the application of NB in data-rich settings. As equation (18.2) makes clear, all predictors are given equal weight in determining the posterior probability of a class. As a result, the performance of the classifier can deteriorate if there are a large number of irrelevant predictors, the probability of which will increase in data-rich settings.

Naïve Bayes classification can be implemented in R via the **caret** package, using the **naive_bayes** method. Implementation involves two tuning parameters. The first is **usekernel**, which indicates whether a Gaussian density or a kernel density estimator is used to approximate $f_j\left(X_{j,t}|S_{t+h}=c\right)$. The second is **adjust**, which is a parameter indicating the size of the bandwidth in the kernel density estimator.

18.3.3 k-nearest neighbors

The k-nearest neighbor (kNN) algorithm is among the simplest of supervised classification techniques. Suppose that for predictor data X_t, we define a neighborhood, labeled $R_k\left(X_t\right)$, that consists of the k closest points to X_t in the training sample (not including X_t itself). Our class prediction for $S_{t+h}=c$ is then simply the proportion of points in the region belonging to class c:

$$\widehat{S}_{t+h}^{c}\left(X_t\right)=\frac{1}{k}\sum_{X_i\in R_k(X_t)} I\left(S_{i+h}=c\right)$$

In other words, to predict S_{t+h}^{c}, we find the k values of X that are closest to X_t, and compute the proportion of these that correspond to class c.

To complete this classifier we must specify a measure of 'closeness.' The most commonly used metric is Euclidean distance:

$$d(X_t,X_i)=\sqrt{\sum_{j=1}^{N}(X_{j,t}-X_{j,i})^2}$$

Other distance metrics are of course possible. The region $R_k\left(X_t\right)$ can also be defined continuously, so that training sample observations are not simply in vs. out of $R_k\left(X_t\right)$, but have declining influence as they move farther away from X_t.

kNN classifiers are simple to understand, and can often provide a powerful classification tool, particularly in cases where X_t is of low dimension. However, kNN is adversely affected in cases where the X_t vector contains many irrelevant predictors, as the metric defining closeness is affected by all predictors, irregardless of their classification ability. Of course, the likelihood of containing many irrelevant predictors increases with larger datasets, and as a result kNN classification is not a commonly

used classifier when there are a large number of predictors. Other approaches, such as the tree-based methods described below, are preferred in data-rich settings, in that they can automatically identify relevant predictors.

As was discussed in Section 18.2.2, in many applications some values of the predictor variables will be missing in the out-of-sample period. It is also possible to have missing values in the training sample. A simple and effective approach to handle missing values is to apply a kNN type procedure to impute the missing values. Specifically, suppose we have a vector X_t that is only partially observed. Segment this vector into X_t^* and $\widetilde{X}_t$, where X_t^* holds the N^* variables that are observed and $\widetilde{X}_t$ holds the $N - N^*$ variables that are not observed. Define a neighborhood, labeled $R_k\left(X_t^*\right)$, that consists of the k closest points to X_t^* over the portion of the training sample for which there are no missing values. Closeness can again be defined in terms of the Euclidean metric:

$$d(X_t^*, X_i^*) = \sqrt{\sum_{j=1}^{N^*}(X_{j,t}^* - X_{j,i}^*)^2}$$

We then impute the missing variable values contained in $\widetilde{X}_t$ using the mean of those same variables in $R_k\left(X_t^*\right)$:

$$\widetilde{X}_t^{imputed} = \frac{1}{k} \sum_{X_i \in R_k\left(X_t^*\right)} \widetilde{X}_i$$

kNN classification can be implemented in R via the **caret** package, using the **knn** method. The **knn** method involves a single tuning parameter, **k**, which indicates the value of k. Also, when implementing any machine learning classification method in R using the **caret** package, kNN imputation can be used to replace missing values via the **preProc** argument to the **train** function.

18.3.4 Learning vector quantization

Learning Vector Quantization (LVQ) is a classifier that forms predictions on the basis of the closeness of X_t to some key points in the predictor space. In this sense it is like kNN, in that it generates predictions based on a nearest-neighbor strategy. However, unlike kNN, these key points are not collections of training sample data points, but instead are endogenously learned from the training sample data. LVQ is widely used in real-time classification problems in a number of fields and applications, and was used in the economics literature by Giusto and Piger (2017) to identify U.S. business cycle turning points in real time. LVQ methods were developed by Tuevo Kohonen and are described in detail in Kohonen (2001).

To describe LVQ it is useful to begin with Vector Quantization (VQ). A VQ classifier relies on the definition of certain key points, called *codebook vectors*, defined in the predictor space. Each codebook vector is assigned to a class, and

there can be more than one codebook vector per class. We would generally have far fewer codebook vectors than data vectors, implying that a codebook vector provides representation for a group of training sample data vectors. In other words, the codebook vectors *quantize* the salient features of the predictor data. Once these codebook vectors are singled out, data is classified via a majority vote of the nearest group of k codebook vectors in the Euclidean metric.

How is the location of each codebook vector established? An LVQ algorithm is an adaptive learning algorithm in which the locations of the codebook vectors are determined through adjustments of decreasing magnitude. Denote our codebook vectors as $v_i \in R^N, i = 1, \ldots, V$, let $g = 1, 2, \ldots, G$ denote iterations of the algorithm, and let α^g be a decreasing geometric sequence where $0 < \alpha < 1$. Given the initial location of the codebook vectors, the LVQ algorithm makes adjustments to their location as described in Algorithm 15:

Algorithm 15 Learning Vector Quantization (Kohonen (2001))

Initialize $v_i^0, i = 1, \ldots, V$
for $g = 1$ to G **do**.
  for $t = 1$ to T **do**.
    Identify the single codebook vector v_*^{g-1} closest to X_t in the Euclidean metric.
    Adjust the location of v_*^g according to:

$$v_*^g = v_*^{g-1} + \alpha^g (X_t - v_*^{g-1}) \qquad \text{if } X_t \text{ and } v_*^{g-1} \text{ belong to the same class}$$
$$v_*^g = v_*^{g-1} - \alpha^g (X_t - v_*^{g-1}) \qquad \text{otherwise}$$

  end for
end for

This LVQ algorithm is very simple. A data vector is considered, and its nearest codebook vector is identified. If the class attached to this codebook vector agrees with the actual classification of the data vector, its location is moved closer to the data vector. If the selected codebook vector does not classify the data vector correctly, then it is moved farther from the data vector. These adjustments are made in a simple linear fashion. These calculations are repeated for each data vector in the data set. When the data has all been used, a new iteration is started where the weight α^g, which controls the size of the adjustment to the codebook vectors, is decreased. This continues for G iterations, with the final codebook vectors given by $v_i^G, i = 1, \ldots, V$.

Once the final codebook vectors are established, the LVQ classifier produces a prediction, $\widehat{S}_{t+h}(X_t)$, via a majority voting strategy. First, we identify the k closest codebook vectors to X_t in the Euclidean metric. Second, the predicted class for X_t is set equal to the majority class of these k codebook vectors. Denote this majority class as c^*. Then:

$$\widehat{S}_{t+h}^c(X_t) = 1, \qquad \text{if } \; c = c^*$$
$$\widehat{S}_{t+h}^c(X_t) = 0, \qquad \text{otherwise}$$

Here I have laid out the basic LVQ algorithm, which has been shown to work well in many practical applications. Various modifications to this algorithm have been proposed, which may improve classification ability in some contexts. These include LVQ with nonlinear updating rules, as in the Generalized LVQ algorithm of Sato and Yamada (1995), as well as LVQ employed with alternatives to the Euclidean measure of distance, such as the Generalized Relevance LVQ of Hammer and Villmann (2002). The latter allows for adaptive weighting of data series in the dimensions most helpful for classification, and may be particularly useful when applying LVQ to large datasets.

LVQ classification can be implemented in R via the **caret** package using the **lvq** method. The **lvq** method has two tuning parameters. The first is the number of codebook vectors to use in creating the final classification, k, and is labeled **k**. The second is the total number of codebook vectors V, and is labeled **size**. To implement the classifier, one must also set the values of G and α. In the **lvq** method, G is set endogenously to ensure convergence of the codebook vectors. The default value of α in the **lvq** method is 0.3. Kohonen (2001) argues that classification results from LVQ should be largely invariant to the choice of alternative values of α provided that $\alpha^g \to 0$ as $g \to \infty$, which ensures that the size of codebook vector updates eventually converge to zero. Giusto and Piger (2017) verified this insensitivity in their application of LVQ to identifying business cycle turning points.

The LVQ algorithm requires an initialization of the codebook vectors. This initialization can have effects on the resulting class prediction, as the final placement of the codebook vectors in an LVQ algorithm is not invariant to initialization. A simple approach, which I follow in the application, is to allow all classes to have the same number of codebook vectors, and initialize the codebook vectors attached to each class with random draws of X_t vectors from training sample observations corresponding to each class.

18.3.5 Classification trees

A number of commonly used classification techniques are based on **classification trees**. I will discuss several of these approaches in subsequent sections, each of which uses aggregations of multiple classification trees to generate predictions. Before delving into these techniques, in this section I describe the single classification trees upon which they are built.

A classification tree is an intuitive, non-parametric, procedure that approaches classification by partitioning the predictor variable space into non-overlapping regions. These regions are created according to a conjunction of binary conditions. As an example, in a case with two predictor variables, one of the regions might be of the form $\{X_t | X_{1,t} \geq \tau_1, X_{2,t} < \tau_2\}$. The partitions are established in such a way so as to effectively isolate alternative classes in the training sample. For example, the region mentioned above might have been chosen because it corresponds to cases where

S_{t+h} is usually equal to c. To generate predictions, the classification tree would then place a high probability on $S_{t+h} = c$ if X_t fell in this region.

How specifically are the partitions established using a training sample? Here I will describe a popular training algorithm for a classification tree, namely the classification and regression tree (CART). CART proceeds by recursively partitioning the training sample through a series of binary splits of the predictor data. Each new partition, or split, segments a region that was previously defined by the earlier splits. The new split is determined by one of the predictor variables, labeled the 'split variable,' based on a binary condition of the form $X_{j,t} < \tau$ and $X_{j,t} \geq \tau$, where both j and τ can differ across splits. The totality of these recursive splits partition the sample space into M non-overlapping regions, labeled A_m^*, $m = 1, \ldots, M$, where there are T_m^* training sample observations in each region. For all X_t that are in region A_m^*, the prediction for S_{t+h} is a constant equal to the within-region sample proportion of class c:

$$P_{A_m^*}^c = \frac{1}{T_m^*} \sum_{X_t \in A_m^*} I(S_{t+h} = c) \tag{18.3}$$

The CART classifier is then:

$$\widehat{S}_{t+h}^c(X_t) = \sum_{m=1}^{M} P_{A_m^*}^c \; I\left(X_t \in A_m^*\right) \tag{18.4}$$

How is the recursive partitioning implemented to arrive at the regions A_m^*? Suppose that at a given step in the recursion, we have a region defined by the totality of the previous splits. In the language of decision trees, this region is called a 'node.' Further, assume this node has not itself yet been segmented into subregions. I refer to such a node as an 'unsplit node' and label this unsplit node generically as A. For a given j and τ^j we then segment the data in this region according to $\{X_t | X_{j,t} < \tau^j, X_t \in A\}$ and $\{X_t | X_{j,t} \geq \tau^j, X_t \in A\}$, which splits the data in this node into two non-overlapping regions, labelled A^L and A^R respectively. In these regions, there are T^L and T^R training sample observations. In order to determine the splitting variable, j, and the split threshold, τ^j, we scan through all pairs $\{j, \tau^j\}$, where $j = 1, \ldots N$ and $\tau^j \in \mathcal{T}^{A,j}$, to find the values that maximize a measure of the homogeneity of class outcomes inside of A^L and A^R.[5] Although a variety of measures of homogeneity are possible, a common choice is the Gini impurity:

[5] In a CART classification tree, $\mathcal{T}^{A,j}$ is a discrete set of all non-equivalent values for τ^j, which is simply the set of midpoints of the ordered values for $X_{j,t}$ in the training sample observations relevant for node A.

$$G_L = \sum_{c=1}^{C} P_{A^L}^c (1 - P_{A^L}^c)$$

$$G_R = \sum_{c=1}^{C} P_{A^R}^c (1 - P_{A^R}^c)$$

The Gini impurity is bounded between zero and one, where a value of zero indicates a 'pure' region where only one class is present. Higher values of the Gini impurity indicate greater class diversity. The average Gini impurity for the two new proposed regions is:

$$\overline{G} = \frac{T^L}{T^L + T^R} G_L + \frac{T^R}{T^L + T^R} G_R \tag{18.5}$$

The split point j and split threshold τ^j are then chosen to create regions A^L and A^R that minimize the average Gini impurity.

This procedure is repeated for other unsplit nodes of the tree, with each additional split creating two new unsplit nodes. By continuing this process recursively, the sample space is divided into smaller and smaller regions. One could allow the recursion to run until we are left with only pure unsplit nodes. A more common choice in practice is to stop splitting nodes when any newly created region would contain a number of observations below some predefined minimum. This minimum number of observations per region is a tuning parameter of the classification tree. Whatever approach is taken, when we arrive at an unsplit node that is not going to be split further, this node becomes one of our final regions A_m^*. In the language of decision trees, this final unsplit node is referred to as a 'leaf.' Algorithm 16 provides a description of the CART classification tree.

Algorithm 16 A Single CART Classification Tree (Breiman, Friedman, Olshen, and Stone (1984))

1: Initialize a single unsplit node to contain the full training sample
2: **for** All unsplit nodes A_u with total observations > threshold **do**
3: **for** $j = 1$ to N and $\tau^j \in \mathcal{T}^{A_u, j}$ **do**
4: Create non-overlapping regions $A_u^L = \{X_t | X_{j,t} < \tau^j, X_t \in A_u\}$ and $A_u^R = \{X_t | X_{j,t} \geq \tau^j, X_t \in A_u\}$ and calculate $\bar{G}$ as in (18.5).
5: **end for**
6: Select j and τ^j to minimize $\bar{G}$ and create the associated nodes A_u^L and A_u^R.
7: Update the set of unsplit nodes to include A_u^L and A_u^R.
8: **end for**
9: For final leaf nodes, A_m^*, form $P_{A_m^*}^c$ as in (18.3), for $c = 1, \ldots, C$ and $m = 1, \ldots M$
10: Form the CART classification tree classifier: $\widehat{S}_{t+h}^c(X_t)$ as in (18.4).

Classification trees have many advantages. They are simple to understand, require no parametric modeling assumptions, and are flexible enough to capture complicated nonlinear and discontinuous relationships between the predictor variables and class indicator. Also, the recursive partitioning algorithm described above scales easily to

large datasets, making classification trees attractive in this setting. Finally, unlike the classifiers we have encountered to this point, a CART classification tree automatically conducts model selection in the process of producing a prediction. Specifically, a variable may never be used as a splitting variable, which leaves it unused in producing a prediction by the classifier. Likewise, another variable may be used multiple times as a splitting variable. These differences result in varying levels of variable importance in a CART classifier. Hastie, Tibshirani, and Friedman (2009) detail a measure of variable importance that can be produced from a CART tree.

CART trees have one significant disadvantage. The sequence of binary splits, and the path dependence this produces, generally produces a high variance forecast. That is, small changes in the training sample can produce very different classification trees and associated predictions. As a result, a number of procedures exist that attempt to retain the benefits of classification trees, while reducing variance. We turn to these procedures next.

18.3.6 Bagging, random forests, and extremely randomized trees

In this section we describe **bagged classification trees**, their close variant, the **random forest**, and a modification to the random forest known as **extremely randomized trees**. Each of these approaches average the predicted classification coming from many classification trees. This allows us to harness the advantages of tree-based methods, while at the same time reducing the variance of the tree-based predictions through averaging. Random forests have been used to identify turning points in economic data by Ward (2017), who uses random forests to identify episodes of financial crises, and Garbellano (2016), who uses random forests to nowcast U.S. recession episodes. Bagging and random forests are discussed in more detail in Chapter 13 of this book.

We begin with bootstrap aggregated (bagged) classification trees, which were introduced in Breiman (1996). Bagged classification trees work by training a large number, B, of CART classification trees and then averaging the class predictions from each of these trees to arrive at a final class prediction. The trees are different because each is trained on a bootstrap training sample of size T, which is created by sampling $\{X_t, S_{t+h}\}$ with replacement from the full training sample. Each tree produces a class prediction, which I label $\widehat{S}^c_{b,t+h}(X_t)$, $b = 1, \ldots, B$. The bagged classifier is then the simple average of the B CART class predictions:

$$\widehat{S}^c_{t+h}(X_t) = \frac{1}{B}\sum_{b=1}^{B} \widehat{S}^c_{b,t+h}(X_t) \tag{18.6}$$

Bagged classification trees are a variance reduction technique that can give substantial gains in accuracy over individual classification trees. As discussed in Breiman (1996), a key determinant of the potential benefits of bagging is the variance of the

individual classification trees across alternative training samples. All else equal, the higher is this variance, the more potential benefit there is from bagging.

As discussed in Breiman (2001), the extent of the variance improvement also depends on the amount of correlation across the individual classification trees that constitute the bagged classification tree, with higher correlation generating lower variance improvements. This correlation could be significant, as the single classification trees used in bagging are trained on overlapping bootstrap training samples. As a result, modifications to the bagged classification tree have been developed that attempt to reduce the correlation across trees, without substantially increasing the bias of the individual classification trees. The most well know of these is the random forest (RF), originally developed in Breiman (2001).[6]

An RF classifier attempts to lower the correlation across trees by adding another layer of randomness into the training of individual classification trees. As with a bagged classification tree, each single classification tree is trained on a bootstrap training sample. However, when training this tree, rather than search over the entire set of predictors $j = 1, \ldots, N$ for the best splitting variable at each node, we instead randomly choose $Q << N$ predictor variables at each node, and search only over these variables for the splitting variable. This procedure is repeated to train B individual classification trees, and the random forest classifier is produced by averaging these classification trees as in equation (18.6). Algorithm 17 provides a description of the RF classifier.

Algorithm 17 Random Forest Classifier (Breiman (2001))

for $b = 1$ to B **do**
  Form a bootstrap training sample by sampling $\{X_t, S_{t+h}\}$ with replacement T times from the training sample observations.
  Initialize a single unsplit node to contain the full bootstrap training sample
  for All unsplit nodes A_u with total observations > threshold **do**
    Randomly select Q predictor variables as possible splitting variables. Denote these predictor variables at time t as $\widetilde{X}_t$
    for $X_{j,t} \in \widetilde{X}_t$ and $\tau^j \in \mathcal{T}^{A_u, j}$ **do**
      Create two non-overlapping regions $A_u^L = \{X_t | X_{j,t} < \tau^j, X_t \in A_u\}$ and $A_R = \{X_t | X_{j,t} \geq \tau^j, X_t \in A_u\}$ and calculate $\bar{G}$ as in (18.5).
    end for
    Select j and τ^j to minimize $\bar{G}$ and create the associated nodes A_u^L and A_u^R.
    Update the set of unsplit nodes to include A_u^L and A_u^R
  end for
  For final leaf nodes, A_m^*, form $P_{A_m^*}^c$ as in (18.3), for $c = 1, \ldots, C$ and $m = 1, \ldots M$
  Form the single tree classifier: $\widehat{S}_{b,t+h}^c(X_t)$ as in (18.4).
end for
Form the Random Forest classifier: $\widehat{S}_{t+h}^c(X_t) = \frac{1}{B} \sum_{b=1}^{B} \widehat{S}_{b,t+h}^c(X_t)$.

[6] Other papers that were influential in the development of random forest methods include Amit and Geman (1997) and Ho (1998).

When implementing a RF classifier, the individual trees are usually allowed to grow to maximum size, meaning that nodes are split until only pure nodes remain. While such a tree should produce a classifier with low bias, it is likely to be unduly influenced by peculiarities of the specific training sample, and thus will have high variance. The averaging of the individual trees lowers this variance, while the additional randomness injected by the random selection of predictor variables helps maximize the variance reduction benefits of averaging. It is worth noting that by searching only over a small random subset of predictors at each node for the optimal splitting variable, the RF classifier also has computational advantages over bagging.

Extremely Randomized Trees, or 'ExtraTrees,' is another approach to reduce correlation among individual classification trees. Unlike both bagged classification trees and RF, ExtraTrees trains each individual classification tree on the entire training sample rather than bootstrapped samples. As does a random forest, ExtraTrees randomizes the subset of predictor variables considered as possible splitting variables at each node. The innovation with ExtraTrees is that when training the tree, for each possible split variable, only a single value of τ^j is considered as the possible split threshold. For each j, this value is randomly chosen from the uniform interval $\left[min(X_{j,t}|X_{j,t} \in A), max(X_{j,t}|X_{j,t} \in A)\right]$. Thus, ExtraTrees randomizes across both the split variable and the split threshold dimension. ExtraTrees was introduced by Geurts, Ernst, and Wehenkel (2006), who argue that the additional randomization introduced by ExtraTrees should reduce variance more strongly than weaker randomization schemes. Also, the lack of a search over all possible τ^j for each split variable at each node provides additional computational advantages over the RF classifier. Algorithm 18 provides a description of the ExtraTrees classifier:

Algorithm 18 Extremely Randomized Trees (Geurts, Ernst, and Wehenkel (2006))

for $b = 1$ to B **do**
  Initialize a single unsplit node to contain the full training sample
  for All unsplit nodes A_u with total observations > threshold **do**
    Randomly select Q predictor variables as possible splitting variables. Denote these predictor variables at time t as $\widetilde{X}_t$
    for $X_{j,t} \in \widetilde{X}_t$ **do**
      Randomly select a single τ^j from the uniform interval: $\left[min(X_{j,t}|X_{j,t} \in A_u), max(X_{j,t}|X_{j,t} \in A_u)\right]$
      Create two non-overlapping regions $A_u^L = \{X_t|X_{j,t} < \tau^j, X_t \in A_u\}$ and $A_R = \{X_t|X_{j,t} \geq \tau^j, X_t \in A_u\}$ and calculate $\bar{G}$ as in (18.5).
    end for
    Select j to minimize $\bar{G}$ and create the associated nodes A_u^L and A_u^R.
    Update the set of unsplit nodes to include A_u^L and A_u^R
  end for
  For final leaf nodes, A_m^*, form $P_{A_m^*}^c$ as in (18.3), for $c = 1, \ldots, C$ and $m = 1, \ldots M$
  Form the single tree classifier: $\widehat{S}_{b,t+h}^c(X_t)$ as in (18.4).
end for
Form the ExtraTrees classifier: $\widehat{S}_{t+h}^c(X_t) = \frac{1}{B}\sum_{b=1}^{B} \widehat{S}_{b,t+h}^c(X_t)$.

RF and ExtraTrees classifiers have enjoyed a substantial amount of success in empirical applications. They are also particularly well suited for data-rich environments. Application to wide datasets of predictor variables is computationally tractable, requiring a number of scans that at most increase linearly with the number of predictors. Also, because these algorithms are based on classification trees, they automatically conduct model selection as the classifier is trained.

Both RF and ExtraTrees classifiers can be implemented in `R` via the **`caret`** package, using the **`ranger`** method. Implementation involves three tuning parameters. The first is the value of Q and is denoted **`mtry`** in the caret package. The second is **`splitrule`** and indicates whether a random forest (**`splitrule`**$= 0$) or an extremely randomized tree (**`splitrule`**$= 1$) is trained. Finally, **`min.node.size`** indicates the minimum number of observations allowed in the final regions established for each individual classification tree. As discussed above, it is common with random forests and extremely randomized trees to allow trees to be trained until all regions are pure. This can be accomplished by setting **`min.node.size`**$= 1$.

18.3.7 Boosting

Boosting has been described by Hastie et al. (2009) as "one of the most powerful learning ideas introduced in the last twenty years" and by Breiman (1996) as "the best off-the-shelf classifier in the world." Many alternative descriptions and interpretations of boosting exist, and a recent survey and historical perspective is provided in Mayr, Binder, Gefeller, and Schmid (2014). In economics, several recent papers have used boosting to predict expansion and recession episodes, including Ng (2014), Berge (2015) and D opke, Fritsche, and Pierdzioch (2017). Boosting is described in more detail in Chapter 14 of this book.

The central idea of boosting is to recursively apply simple 'base' learners to a training sample, and then combine these base learners to form a strong classifier. In each step of the recursive boosting procedure, the base learner is trained on a weighted sample of the data, where the weighting is done so as to emphasize observations in the sample that to that point had been classified *incorrectly*. The final classifier is formed by combining the sequence of base learners, with better performing base learners getting more weight in this combination.

The first boosting algorithms are credited to Schapire (1990), Freund (1995) and Freund and Schapire (1996) and are referred to as **AdaBoost**. Later work by Friedman, Hastie, and Tibshirani (2000) interpreted AdaBoost as a forward stagewise procedure to fit an additive logistic regression model, while Friedman (2001) showed that boosting algorithms can be interpreted generally as non-parametric function estimation using gradient descent. In the following I will describe boosting in more detail using these later interpretations.

For notational simplicity, and to provide a working example, consider a two class case, where I define the two classes as $S_{t+h} = -1$ and $S_{t+h} = 1$. Define a function $F(X_t) \in \mathbb{R}$ that is meant to model the relationship between our predictor variables,

X_t, and S_{t+h}. Larger values of $F(X_t)$ signal increased evidence for $S_{t+h} = 1$, while smaller values indicate increased evidence for $S_{t+h} = -1$. Finally, define a loss function, $C(S_{t+h}, F(X_t))$, and suppose our goal is to choose $F(X_t)$ such that we minimize the expected loss:

$$E_{S,X} C(S_{t+h}, F(X_t)) \tag{18.7}$$

A common loss function for classification is exponential loss:

$$C(S_{t+h}, F(X_t)) = \exp\left(-S_{t+h} F(X_t)\right)$$

The exponential loss function is smaller if the signs of S_{t+h} and $F(X_t)$ match than if they do not. Also, this loss function rewards (penalizes) larger absolute values of $F(X_t)$ when it is correct (incorrect).

For exponential loss, it is straightforward to show (Friedman et al. (2000)) that the $F(X_t)$ that minimizes equation (18.7) is:

$$F(X_t) = \frac{1}{2} \ln \left[\frac{\Pr\left(S_{t+h} = 1 | X_t\right)}{\Pr\left(S_{t+h} = -1 | X_t\right)} \right]$$

which is simply one-half the log odds ratio. A traditional approach commonly found in economic studies is to assume an approximating parametric model for the log odds ratio. For example, a parametric logistic regression model would specify:

$$\ln \left[\frac{\Pr\left(S_{t+h} = 1 | X_t\right)}{\Pr\left(S_{t+h} = -1 | X_t\right)} \right] = X_t' \beta$$

A boosting algorithm alternatively models $F(X_t)$ as an additive model (Friedman et al. (2000)):

$$F(X_t) = \sum_{j=1}^{J} \alpha_j T_j\left(X_t; \beta_j\right) \tag{18.8}$$

where each $T_j\left(X_t; \beta_j\right)$ is a base learner with parameters β_j. $T_j\left(X_t; \beta_j\right)$ is usually chosen as a simple model or algorithm with only a small number of associated parameters. A very common choice for $T_j\left(X_t; \beta_j\right)$ is a CART regression tree with a small number of splits.

Boosting algorithms fit equation (18.8) to the training sample in a forward stagewise manner. An additive model fit via forward stagewise iteratively solves for the loss minimizing $\alpha_j T_j\left(X_t; \beta_j\right)$, conditional on the sum of previously fit terms, labeled $F_{j-1}(X_t) = \sum_{i=1}^{j-1} \alpha_i T_i\left(X_t; \beta_i\right)$. Specifically, conditional on an initial $F_0(X_t)$, we iteratively solve the following for $j = 1, \ldots, J$:

$$\{\alpha_j, \beta_j\} = \min_{\alpha_j, \beta_j} \sum_{t=1}^{T} C\left(S_{t+h}, \left[F_{j-1}(X_t) + \alpha_j T_j\left(X_t; \beta_j\right)\right]\right) \tag{18.9}$$

$$F_j\left(X_t\right) = F_{j-1}\left(X_t\right) + \alpha_j T_j\left(X_t; \beta_j\right) \tag{18.10}$$

Gradient boosting finds an approximate solution to equation (18.9)-(18.10) via a two step procedure. First, for each j, compute the 'pseudo-residuals' as the negative gradient of the loss function evaluated at $F_{j-1}\left(X_t\right)$:

$$e_{j,t+h} = -\left[\frac{\partial C(S_{t+h}, F(X_t))}{\partial F(X_t)}\right]_{F(X_t)=F_{j-1}(X_t)}$$

Next, a CART regression tree $T_j\left(X_t; \beta_j\right)$ is fit to the pseudo-residuals. Specifically, a tree is trained on a training sample made up of $\{e_{j,t+h}, X_t\}_{t=1}^{T}$, with the final tree containing M non-overlapping leaves, $A_{m,j}^*$, $m = 1, \ldots, M$. The CART regression tree predicts a constant in each region:

$$T_j\left(X_t; \beta_j\right) = \sum_{m=1}^{M} \bar{e}_{m,j} I(X_t \in A_{m,j}^*)$$

where $\bar{e}_{m,j}$ is the simple average of $e_{j,t+h}$ inside the leaf $A_{m,j}^*$, and β_j represents the parameters of this tree, which would include details such as the split locations and splitting variables. These are chosen as described in Section (18.3.5), but as the pseudo-residuals are continuous, a least squares criterion is minimized to choose β_j rather than the Gini impurity. Notice that because the tree predicts a constant in each regime, the solution to equation (18.9) involves simply a single parameter optimization in each of the $A_{m,j}^*$ regions. Each of these optimizations takes the form:

$$\gamma_{m,j} = \min_{\gamma} \sum_{X_t \in A_{m,j}^*} C\left(S_{t+h}, F_{j-1}\left(X_t\right) + \gamma\right), \quad m = 1, \ldots, M$$

Given this solution, equation (18.10) becomes:

$$F_j\left(X_t\right) = F_{j-1}\left(X_t\right) + \sum_{m=1}^{M} \gamma_{m,j} I(X_t \in A_{m,j}^*) \tag{18.11}$$

As discussed in (Friedman (2001)), gradient boosting is analogous to AdaBoost when the loss function is exponential. However, gradient boosting is more general, and can be implemented for any differentiable loss function. Gradient boosting also helps expose the intuition of boosting. The gradient boosting algorithm approximates the optimal $F(X_t)$ through a series of Newton steps, and in this sense boosting can be interpreted as a numerical minimization of the empirical loss function in the space of the function $F(X_t)$. Each of these Newton steps moves $F_j(X_t)$ in the direction of the negative gradient of the loss function, which is the direction of greatest descent for the loss function in $F(X_t)$ space. Loosely speaking, the negative gradient, or pseudo-residuals, provides us with the residuals from applying $F_{j-1}(X_t)$ to classify S_{t+h}. In this sense, at each step, the boosting algorithm focuses on observations that were classified incorrectly in the previous step.

Finally, Friedman (2001) suggests a modification of equation (18.11) to introduce a shrinkage parameter:

$$F_j(X_t) = F_{j-1}(X_t) + \eta \sum_{m=1}^{M} \gamma_{m,j} I(X_t \in A^*_{m,j}),$$

where $0 < \eta \leq 1$ controls the size of the function steps in the gradient based numerical optimization. In practice, η is a tuning parameter for the gradient boosting algorithm.

Gradient boosting with trees as the base learners is referred to under a variety of names, including a Gradient Boosting Machine, MART (multiple additive regression trees), TreeBoost and a Boosted Regression Tree. The boosting algorithm for our two class example is shown in Algorithm 19.

Algorithm 19 Gradient Boosting with Trees (Friedman (2001))

1: Initialize $F^0(X_t) = \min_{\gamma} \sum_{t=1}^{T} C(S_{t+h}, \gamma)$
2: **for** $j = 1$ to J **do**
3: $\quad e_{j,t+h} = -\left[\frac{\partial C(S_{t+h}, F(X_t))}{\partial F(X_t)}\right]_{F(X_t)=F_{j-1}(X_t)}, t = 1 \ldots T$
4: $\quad$ Fit $T\left(X_t; \beta_j\right)$ to $\{e_{j,t+h}, X_t\}_{t=1}^{T}$ to determine regions $A^*_{m,j}, \quad m = 1, \ldots M$
5: $\quad \gamma_{m,j} = \min_{\gamma} \sum_{X_t \in A^*_{m,j}} C\left(S_{t+h}, F_{j-1}(X_t) + \gamma\right), \quad m = 1, \ldots M$
6: $\quad F_j(X_t) = F_{j-1}(X_t) + \eta \sum_{m=1}^{M} \gamma_{m,j} I\left(X_t \in A^*_{m,j}\right)$
7: **end for**

Upon completion of this algorithm we have $F_J(X_t)$, although in many applications this function is further converted into a more recognizable class prediction. For example, AdaBoost uses the classifier $sign(F_J(X_t))$, which for the two-class example with exponential loss, classifies S_{t+h} according to its highest probability class. In our application, we will instead convert $F_J(X_t)$ to a class probability by inverting the assumed exponential cost function, and use these probabilities as our classifier, $\widehat{S}_{t+h}$. Again, for the two class case with exponential loss:

$$\widehat{S}^{c=1}_{t+h}(X_t) = \frac{exp(2F_J(X_t))}{1 + exp(2F_J(X_t))}$$

$$\widehat{S}^{c=-1}_{t+h}(X_t) = \frac{1}{1 + exp(2F_J(X_t))}$$

Gradient boosting with trees scales very well to data-rich environments. The forward-stagewise gradient boosting algorithms simplify optimization considerably. Further, gradient boosting is commonly implemented with small trees, in part to avoid overfitting. Indeed, a common choice is to use so called 'stumps', which are trees with only a single split. This makes implementation with large sets of predictors

very fast, as at each step in the boosting algorithm, only a small number of scans through the predictor variables is required.

Two final aspects of gradient boosting bear further comment. First, as discussed in Hastie et al. (2009), the algorithm above can be modified to incorporate $K > 2$ classes by assuming a negative multinomial log likelihood cost function. Second, Friedman (2001) suggests a modified version of Algorithm 19 in which, at each step j, a random subsample of the observations is chosen. This modification, known as 'stochastic gradient boosting,' can help prevent overfitting while also improving computational efficiency.

Gradient boosting can be implemented in R via the **caret** package, using the **gbm** method. Implementation of gbm where regression trees are the base learners involves four tuning parameters. The first is **n.trees**, which is the stopping point J for the additive model in equation (18.8). The second is **interaction.depth**, which is the depth (maximum number of consecutive splits) of the regression trees used as weak learners. **shrinkage** is the shrinkage parameter, η in the updating rule equation (18.3.7). Finally, **n.minobsinnode** is the minimum terminal node size for the regression trees.

18.4 Markov-Switching Models

In this section we describe Markov-switching (MS) models, which are a popular approach for both historical and real-time classification of economic data. In contrast to the machine learning algorithms presented in the previous section, MS models are unsupervised, meaning that a historical time series indicating the class is not required. Instead, MS models assume a parametric structure for the evolution of the class, as well as for the interaction of the class with observed data. This structure allows for statistical inference on which class is, or will be, active. In data-rich environments, Markov-switching can be combined with dynamic factor models to capture the information contained in datasets with many predictors. Obviously, MS models are particularly attractive when a historical class indicator is not available, and thus supervised approaches cannot be implemented. However, MS models have also been used quite effectively for real-time classification in settings where a historical indicator is available. We will see an example of this in Section 18.5.

MS models are parametric time-series models in which parameters are allowed to take on different values in each of C regimes, which for our purposes correspond to the classes of interest. A fundamental difference from the supervised approaches we have already discussed is that these regimes are not assumed to be observed in the training sample. Instead, a stochastic process assumed to have generated the regime shifts is included as part of the model, which allows for both in-sample historical inference on which regime is active, as well as out-of-sample forecasts of regimes. In the MS model, introduced to econometrics by Goldfeld and Quandt (1973), Cosslett and Lee (1985), and Hamilton (1989), the stochastic process assumed is a C-state Markov process. Also, and in contrast to the non-parametric approaches we have

already seen, a specific parametric structure is assumed to link the observed X_t to the regimes. Following Hamilton (1989), this linking model is usually an autoregressive time-series model with parameters that differ in the C regimes. The primary use of these models in the applied economics literature has been to describe changes in the dynamic behavior of macroeconomic and financial time series.

The parametric structure of MS models comes with some benefits for classification. First, by specifying a stochastic process for the regimes, one can allow for dynamic features that may help with both historical and out-of-sample classification. For example, most economic regimes of interest display substantial levels of persistence. In an MS model, this persistence is captured by the assumed Markov process for the regimes. Second, by assuming a parametric model linking X_t to the classes, the model allows the researcher to focus the classification exercise on the object of interest. For example, if one is interested in identifying high and low volatility regimes, a model that allows for switching in only conditional variance of an AR model could be specified.[7]

Since the seminal work of Hamilton (1989), MS models have become a very popular modeling tool for applied work in economics. Of particular note are regime-switching models of measures of economic output, such as real Gross Domestic Product (GDP), which have been used to model and identify the phases of the business cycle. Examples of such models include Hamilton (1989), Chauvet (1998), Kim and Nelson (1999a), Kim and Nelson (1999b), and Kim, Morley, and Piger (2005). A sampling of other applications include modeling regime shifts in time series of inflation and interest rates (Evans and Wachtel (1993); Garcia and Perron (1996); Ang and Bekaert (2002)), high and low volatility regimes in equity returns (Turner, Startz, and Nelson (1989); Hamilton and Susmel (1994); Hamilton and Lin (1996); Dueker (1997); Guidolin and Timmermann (2005)), shifts in the Federal Reserve's policy "rule" (Kim (2004); Sims and Zha (2006)), and time variation in the response of economic output to monetary policy actions (Garcia and Schaller (2002); Kaufmann (2002); Ravn and Sola (2004); Lo and Piger (2005)). Hamilton and Raj (2002), Hamilton (2008) and Piger (2009) provide surveys of MS models, while Hamilton (1994) and Kim and Nelson (1999c) provide textbook treatments.

Following Hamilton (1989), early work on MS models focused on univariate models. In this case, X_t is scalar, and a common modeling choice is a p^{th}-order autoregressive model with Markov-switching parameters:

$$X_t = \mu_{S_{t+h}} + \phi_{1,S_{t+h}}\left(X_{t-1} - \mu_{S_{t+h-1}}\right) + \cdots + \phi_{p,S_{t+h}}\left(X_{t-p} - \mu_{S_{t+h-p}}\right) + \varepsilon_t$$
$$\varepsilon_t \sim N\left(0, \sigma^2_{S_{t+h}}\right) \tag{18.12}$$

[7] MS models generally require a normalization in order to properly define the regimes. For example, in a two regime example where the regimes are high and low volatility, we could specify that $S_{t+h} = 1$ is the low variance regime and $S_{t+h} = 2$ is the high variance regime. In practice this is enforced by restricting the variance in $S_{t+h} = 2$ to be larger than that in $S_{t+h} = 1$. See Hamilton, Waggoner, and Zha (2007) for an extensive discussion of normalization in the MS model.

where $S_{t+h} \in \{1, \ldots, C\}$ indicates the regime and is assumed to be unobserved, even in the training sample. In this model, each of the mean, autoregressive parameters and conditional variance parameters are allowed to change in each of the C different regimes. Hamilton (1989) develops a recursive filter that can be used to construct the likelihood function for this MS autoregressive model, and thus estimate the parameters of the model via maximum likelihood.

A subsequent literature explored Markov-switching in multivariate settings. In the context of identifying business cycle regimes, Diebold and Rudebusch (1996) argue that considering multivariate information in the form of a factor structure can drastically improve statistical identification of the regimes. Chauvet (1998) operationalizes this idea by developing a statistical model that incorporates both a dynamic factor model and Markov switching, now commonly called a dynamic factor Markov-switching (DFMS) model. Specifically, if X_t is multivariate, we assume that X_t is driven by a single-index dynamic factor structure, where the dynamic factor is itself driven by a Markov-switching process. A typical example of such a model is as follows:

$$X_t^{std} = \begin{bmatrix} \lambda_1(L) \\ \lambda_2(L) \\ \vdots \\ \lambda_N(L) \end{bmatrix} F_t + v_t$$

where X_t^{std} is the demeaned and standardized vector of predictor variables, $\lambda_i(L)$ is a lag polynomial, and $v_t = \left(v_{1,t}, v_{2,t}, \ldots, v_{N,t}\right)'$ is a zero-mean disturbance vector meant to capture idiosyncratic variation in the series. v_t is allowed to be serially correlated, but its cross-correlations are limited. In the so-called 'exact' factor model we assume that $E(v_{i,t}v_{j,t}) = 0$, while in the 'approximate' factor model v_t is allowed to have weak cross correlations. Finally, F_t is the unobserved, scalar, 'dynamic factor.' We assume that F_t follows a Markov-switching autoregressive process as in equation (18.12), with X_t replaced by F_t.

Chauvet (1998) specifies a version of this DFMS model where the number of predictors is $N = 4$ and shows how the parameters of both the dynamic factor process and the MS process can be estimated jointly via the approximate maximum likelihood estimator developed in Kim (1994). Kim and Nelson (1998) develop a Bayesian Gibbs-sampling approach to estimate a similar model. Finally, Camacho et al. (2018) develop modifications of the DFMS framework that are useful for real-time monitoring of economic activity, including mixed-frequency data and unbalanced panels. Chapter 2 of this book presents additional discussion of the DFMS model.

As discussed in Camacho, Perez-Quiros, and Poncela (2015), in data-rich environments the joint estimation of the DFMS model can become computationally unwieldy. In these cases, an alternative, two-step, approach to estimation of model parameters can provide significant computational savings. Specifically, in the first step, the dynamic factor F_t is estimated using the non-parametric principal compo-

nents estimator of Stock and Watson (2002). Specifically, $\widehat{F}_t$ is set equal to the first principal component of X_t^{std}. In a second step, $\widehat{F}_t$ is fit to a univariate MS model as in equation (18.12). The performance of this two-step approach relative to the one-step approach was evaluated by Camacho et al. (2015), and the two step approach was used by Fossati (2016) for the task of identifying U.S. business cycle phases in real time.

For the purposes of this chapter, we are primarily interested in the ability of MS models to produce a class prediction, $\widehat{S}_{t+h}^c$. In a MS model, this prediction comes in the form of a 'smoothed' conditional probability: $\widehat{S}_{t+h}^c = \Pr\left(S_{t+h} = c|\widetilde{X}_T\right), c = 1,\ldots,C$, where $\widetilde{X}_T$ denotes the entire training sample, $\widetilde{X}_T = \{X_t\}_{t=1}^T$. Bayesian estimation approaches of MS models are particularly useful here, as they produce this conditional probability while integrating out uncertainty regarding model parameters, rather than conditioning on estimates of these parameters.

In the application presented in Section 18.5 I will consider two versions of the DFMS model for classification. First, for cases with a small number of predictor variables, we estimate the parameters of the DFMS model jointly using the Bayesian sampler of Kim and Nelson (1998). Second, for cases where the number of predictor variables is large, we use the two step approach described above, where we estimate the univariate MS model for $\widehat{F}_t$ via Bayesian techniques. Both of these approaches produce a classifier in the form of the smoothed conditional probability of the class. A complete description of the Bayesian samplers used in estimation is beyond the scope of this chapter. I refer the interested reader to Kim and Nelson (1999c), where detailed descriptions of Bayesian samplers for MS models can be found.

18.5 Application

In this section I present an application of the classification techniques presented above to nowcasting U.S. expansion and recession phases at the monthly frequency. In this case, $S_{t+h} \in \{1,2\}$, where $S_{t+h} = 1$ indicates a month that is a recession and $S_{t+h} = 2$ indicates a month that is an expansion. As I am interested in nowcasting, I set $h = 0$. As the measure of expansion and recession regimes, I use the NBER business cycle dates, which are determined by the NBER's Business Cycle Dating Committee. I will evaluate the ability of the alternative classifiers to accurately classify out-of-sample months that have not yet been classified by the NBER, and also to provide timely identification of turning points between expansion and recession (peaks) and recession and expansion (troughs) in real time.

Providing improved nowcasts of business cycle phases and associated turning points is of significant importance because there are many examples of turning points that were not predicted ex-ante. This leaves policymakers, financial markets, firms, and individuals to try to determine if a new business cycle phase has already begun. Even this is a difficult task, with new turning points usually not identified until many months after they occur. For example, the NBER has historically announced new

turning points with a lag of between 4 and 21 months. Statistical models improve on the NBER's timeliness considerably, with little difference in the timing of the turning point dates established.[8] However, these models still generally identify turning points only after several months have passed. For example, Hamilton (2011) surveys a wide range of statistical models that were in place to identify business cycle turning points in real time, and finds that such models did not send a definitive signal regarding the December 2007 NBER peak until late 2008.

There have been a number of existing studies that evaluate the performance of individual classifiers to nowcast U.S. business cycle dates. In this chapter I contribute to this literature by providing a comparison of a broad range of classifiers, including several that have not yet been evaluated for the purpose of nowcasting business cycles. In doing so, I also evaluate the ability of these classifiers to provide improved nowcasts using large N vs. small N datasets. Most of the existing literature has focused on small N datasets, usually consisting of four coincident monthly series highlighted by the NBER's Business Cycle Dating Committee as important in their decisions. Notable exceptions are Fossati (2016), Davig and Smalter Hall (2016) and Berge (2015), each of which uses larger datasets to classify NBER recessions in real time.

For predictor variables, I begin with the FRED-MD dataset, which is a monthly dataset on a large number of macroeconomic and financial variables maintained by the Federal Reserve Bank of St. Louis. The development of FRED-MD is described in McCracken and Ng (2015). I use the most recent version of this dataset available, which as of the writing of this chapter was the vintage released at the end of November 2018. This vintage provides data for 128 monthly series covering months from a maximum of January 1959 through October 2018. I then delete six series that are not regularly available over the sample period, and add seven series on manufacturing activity from the National Association of Purchasing Managers, obtained from Quandl (www.quandl.com.) I also add seven indices of 'news implied volatility' as constructed in Manela and Moreira (2017). The addition of these series is motivated by Karnizova and Li (2014), who show that uncertainty measures have predictive power for forecasting U.S. recessions. Finally, I restrict all series to begin in January 1960, which eliminates missing values during 1959 for a number of series.

For all series that are from the original FRED-MD dataset, I transform the series to be stationary using the transformation suggested in McCracken and Ng (2015), implemented using the Matlab code available from Michael McCracken's website. For the seven NAPM series and NVIX series, I leave the series without transformation. In some cases, the transformation involves differencing, which uses up the initial observation. To have a common starting point for our sample, I begin measuring all series in February 1960. The final raw dataset then consists of 136 series, where all series begin in February 1960 and extend to a maximum of October 2018.

In the analysis I consider three alternative datasets. The first, labeled *Narrow* in the tables below, is a small dataset that uses four coincident indicators that have been the focus of much of the U.S. business cycle dating literature. These series are

[8] See, e.g., Chauvet and Piger (2008), Chauvet and Hamilton (2006) and Giusto and Piger (2017).

the growth rate of non-farm payroll employment, the growth rate of the industrial production index, the growth rate of real personal income excluding transfer receipts, and the growth rate of real manufacturing and trade sales. The second dataset, labeled *Real Activity*, is a larger dataset consisting of the 70 variables that are in the groupings 'output and income', 'labor market', 'housing' and 'consumption, orders and inventories' as defined in McCracken and Ng (2015). These variables define the real activity variables in the dataset, and as such target the most obvious variables with which to date turning points in real economic activity. The third dataset, labeled *Broad*, consists of all the variables in the dataset.

To evaluate the performance of the classifiers using these datasets, I conduct a pseudo out-of-sample nowcasting exercise that covers the last two decades. Specifically, consider an analyst applying a classification technique in real time at the end of each month from January 2000 to November 2018. For each of these months, I assume the analyst has a dataset covering the time period that would have been available in real time. That is, I accurately replicate the real-time data reporting lags the analyst would face. The reason this is a pseudo out-of-sample exercise is that I do not use the vintage of each dataset that would have been available in real time, as such vintage data is not readily available for all the variables in our dataset. Chauvet and Piger (2008) show that data revisions do not cause significant inaccuracies in real time dating of turning points. That said, interesting future work would replicate this analysis with a fully vintage dataset.

In each month, the analyst uses the available dataset to train the supervised classifiers over a period for which the NBER classification of S_t is assumed known. At each month, the lag with which the NBER classification is assumed known is allowed to vary, and is set using the approach taken in Giusto and Piger (2017). Specifically, I assume that: 1) The date of a new peak or trough is assumed to be known once it is announced by the NBER. 2) If the NBER does not announce a new peak within twelve months of a date, then it is assumed that a new peak did not occur at that date. Twelve months is the longest historical lag the NBER has taken in announcing a new business cycle peak. 3) Once the date of a new turning point is announced by the NBER, the new NBER business cycle phase (expansion or recession) is assumed to last at least six months. Since the unsupervised DFMS classifier does not require knowledge of the NBER classification, I estimate the parameters of this model over the full period for which the predictor data is available to the analyst. After training, the analyst then uses the classifier to classify those months through the end of the relevant sample of predictor variables for which the NBER dates are not known.

Somewhat more formally, suppose the data sample of predictor variables available to the analyst ends in time H, and the NBER dates are known through time $H - J$. Then the supervised classifiers would be trained on data through $H - J$, and the DFMS model would be estimated on data through H. After training and estimation, all classifiers will be used to classify the unknown NBER dates from month $H - J + 1$ through H, and the accuracy of these monthly classifications will be evaluated. I will also use these out-of-sample classifications to identify new business cycle turning points in real time.

Before discussing the results, there are several details of the implementation to discuss. First, when the analyst applies the classifiers to predict the NBER date out of sample, the most recent data to be classified will be incomplete due to differential reporting lags across series. In general, I handle these 'ragged edges' by filling in the missing values using kNN imputation, as discussed in Section 18.3.3, prior to performing any subsequent analysis. Second, for the supervised classifiers, I classify S_t on the basis of the contemporaneous values of the predictor variables (month t) and the first lag (month $t-1$). That is, the X_t vector contains both the contemporaneous and first lag of all variables in the relevant sample. For the unsupervised DFMS approach, all available values of X_t are used in forming the smoothed posterior probability for each class. Third, for each dataset, I replace outliers, defined as datapoints that are greater than four standard deviations from the mean, with the median of a six quarter window on either side of the outlier. Finally, as is typical in the classification literature, I standardize and demean all variables prior to analysis.[9]

For each of the supervised classifiers, I use the **caret** package in R to train and form predictions. In each case, repeated, stratified k-fold cross validation is used for tuning parameters, with k set equal to 10, and the number of repeats also set equal to 10.[10] The objective function used in the cross validation exercise was AUROC. Default ranges from `caret` were used in tuning parameters. Note that both the kNN imputation for missing values, as well as outlier detection, were done separately on each fold of the cross-validation exercise, which prevents data from outside the fold from informing the within-fold training.

For the unsupervised DFMS model, we must specify a specific version of the Markov-switching equation to apply to the factor F_t. In order to provide a comparison to the existing literature, I use specifications that most closely match those in existing studies. Specifically, for the model applied to the narrow dataset, for which the DFMS model is estimated jointly, I follow Chauvet (1998) and Kim and Nelson (1999b) and allow the factor to follow an AR(2) process with regime switching in mean:

$$F_t = \mu_{S_t} + \phi_1 \left(X_{t-1} - \mu_{S_{t-1}}\right) + \phi_2 \left(X_{t-2} - \mu_{S_{t-2}}\right) + \varepsilon_t$$
$$\varepsilon_t \sim N\left(0, \sigma^2\right)$$

For the DFMS model applied to the real activity dataset, for which the DFMS model is estimated via a two-step procedure, I follow Camacho et al. (2015) and use a simple AR(0) process with a switching mean:

[9] In unreported results, I also considered a version of each supervised classifier that classified based on predictor variables formed as principal components from the relevant dataset. The performance of this version of the classifier was similar in all cases to the results applied to the full dataset of individual predictors.

[10] This is a relatively small number of repeats, and was chosen to reduce the computational burden of the recursive out-of-sample nowcasting exercise. In unreported results, I confirmed the robustness of several randomly chosen reported results to a larger number of repeats (100).

$$F_t = \mu_{S_t} + \varepsilon_t$$
$$\varepsilon_t \sim N\left(0, \sigma^2\right)$$

I do not consider the broad dataset for the DFMS model, as the diversity of series in this dataset is likely not well described by only a single factor as is assumed by the DFMS model.

Table 18.1 presents the QPS and AUROC statistics for each classifier applied to each dataset, calculated over all the out-of-sample observations in the recursive nowcasting exercise. There are several conclusions that can be drawn from these results. First of all, in general, the AUROC statistics are very high, suggesting that each of these classifiers has substantial ability to classify expansion and recession months out of sample. Second, there are only relatively small differences in these statistics across classifiers. The DFMS model applied to the narrow dataset provides the highest AUROC at 0.997, which is very close to perfect classification ability, while the kNN classifier applied to the narrow dataset produces the lowest QPS. However, most classifiers produce AUROCs and QPS values that are reasonably close to these best performing values.

Third, Table 18.1 suggests that the out-of-sample evaluation metrics are only moderately improved, it at all, by considering predictor variables beyond the narrow dataset. The differences in the evaluation statistics that result from changing dataset size are small, and in many cases these changes are not in a consistent direction across the QPS vs. AUROC. Overall, these results do not suggest that considering a larger number of predictors over the narrow dataset is clearly advantageous for nowcasting business cycle phases in U.S. data. Note that some of this result likely comes because there is limited room for improvement over the narrow dataset.

Table 18.1 also presents results for a simple ensemble classifier, which is formed as the average of the alternative classifiers. The ensemble classifier averages the classification from all six classifiers for the narrow and real activity dataset, and averages the classification of the five supervised classifiers for the broad dataset. By averaging across approximately unbiased classifier that are not perfectly correlated, an ensemble classifier holds out the possibility of lower variance forecasts than is produced by the individual classifiers. Interestingly, the ensemble classifiers perform well in this setting, with the ensemble classifier applied to the real activity dataset having the lowest QPS and second highest AUROC of any classifier / dataset combination in the table.

The results in Table 18.1 do not speak directly to the question of identifying turning points (peaks and troughs) in real time. To evaluate the ability of the classifiers to identify turning points, we require a rule to transform the classifier output into turning point predictions. Here I employ a simple rule to identify a new turning point, which can be described as follows: If the most recent known NBER classified month is an expansion month, a business cycle peak is established if $\widehat{S}_t^1(X_t) \geq 0.5$ for the final month in the out-of-sample period. Similarly, if the most recent known NBER classified month is a recession month, a business cycle trough is established if $\widehat{S}_t^1(X_t) < 0.5$ for the final month in the out-of-sample period. This is a rather

Table 18.1 Out-of-Sample Evaluation Metrics for Alternative Classifiers

Classifier	QPS	AUROC
Naïve Bayes		
Narrow	0.058	0.990
Real Activity	0.064	0.974
Broad	0.074	0.968
kNN		
Narrow	0.030	0.989
Real Activity	0.033	0.978
Broad	0.055	0.990
Random Forest / Extra Trees		
Narrow	0.034	0.988
Real Activity	0.032	0.988
Broad	0.036	0.989
Boosting		
Narrow	0.043	0.980
Real Activity	0.037	0.978
Broad	0.039	0.982
LVQ		
Narrow	0.043	0.938
Real Activity	0.046	0.930
Broad	0.038	0.952
DFMS		
Narrow	0.041	0.997
Real Activity	0.047	0.992
Ensemble		
Narrow	0.034	0.992
Real Activity	0.029	0.993
Broad	0.031	0.993

Notes: This table shows the quadratic probability score (QPS) and the area under the ROC curve (AUROC) for out-of-sample nowcasts produced from January 2000 to October 2018 by the supervised and unsupervised classifiers discussed in Sections 18.3 and 18.4.

aggressive rule, which puts a high value on speed of detection. As such, we will be particularly interested in the tendency of this rule to identify false turning points.

Table 18.2 shows the performance of each classifier for identifying the four U.S. business cycle turning points over the 2000-2018 time period. In the table, the dates shown are the first month in which the analyst would have been able to identify a turning point in the vicinity of the relevant turning point. For example, consider the column for the December 2007 business cycle peak. An entry of 'Mar 2008' means that an analyst applying the classifiers at the end of March 2008 would have detected a business cycle peak in the vicinity of the December 2007 peak. Because there is a minimum one month lag in data reporting for all series in the FRED-MD dataset, the analyst would have been using a dataset that extended through February 2008 to identify this turning point. An entry of 'NA' means that the relevant turning point was not identified prior to the NBER Business Cycle Dating Committee making the announcement of a new business cycle turning point.

I begin with the two business cycle peaks in the out-of-sample period. If we focus on the narrow dataset, we see that most of the classifiers identify the March 2001 business cycle peak by the end of May 2001, and the December 2007 peak by the end of May 2008. This is a very timely identification of both of these turning points. For the March 2001 peak, identification at the end of May 2001 means that the classifiers identified this recession using data through April 2001, which was the very first month of the recession. For the December 2007 peak, Hamilton (2011) reports that other real-time approaches in use at the time did not identify a business cycle peak until late 2008 or early 2009, and the NBER business cycle dating committee announced the December 2007 peak in December 2008. Thus, identification at the end of May 2008 is relatively very fast. This timely performance does come with a single false business cycle peak being called for several, although not all, of the classifiers. The date of this false peak was in September 2005 for most classifiers.

If we move to the larger real activity and broad datasets, in most cases the performance of the classifiers for identifying business cycle peaks deteriorates. Two of the classifiers, kNN and Random Forests, fail to identify the 2001 business cycle peak, while several classifiers identify peaks more slowly when using the larger datasets than the narrow dataset. There are some cases where moving from the narrow to the real activity dataset does improve with detection. The primary example is the DFMS model, where three of the four turning points are identified more quickly when using the real activity dataset, and the false peak that occurs under the narrow dataset is eliminated. Overall, a reasonable conclusion is that there are limited gains from using larger datasets to identify business cycle peaks in real time, with the gains that do occur coming from the use of the real activity dataset with certain classifiers.

Moving to troughs, the five supervised classification techniques identify troughs very quickly in real time when applied to the narrow dataset. For the November 2001 trough, these classifiers identify the trough by January or February of 2002, while the June 2009 trough is identified by August or September of 2009. This is impressive considering the very slow nature of the recovery following these recessions. As an example, the NBER business cycle dating committee didn't identify the November

Table 18.2 Out-of-Sample Turning Point Identification for Alternative Classifiers

	Peaks		*Troughs*		
Classifier	**Mar 2001**	**Dec 2007**	**Nov 2001**	**Jun 2009**	**False Turning Points**
Naïve Bayes					
Narrow	Feb 2001	May 2008	Jan 2002	Sep 2009	False Peak: Sep 2005
Real Activity	Feb 2001	Mar 2008	Mar 2002	Feb 2010	False Peak: Nov 2010
Broad	Oct 2001	Mar 2008	Jan 2002	Feb 2010	None
kNN					
Narrow	May 2001	May 2008	Jan 2002	Aug 2009	None
Real Activity	NA	Sep 2008	Jan 2002	Aug 2009	None
Broad	NA	NA	Nov 2001	Jul 2009	None
Random Forest / Extra Trees					
Narrow	May 2001	May 2008	Jan 2002	Aug 2009	None
Real Activity	NA	May 2008	Mar 2002	Aug 2009	None
Broad	NA	Oct 2008	Jan 2002	Aug 2009	None
Boosting					
Narrow	May 2001	May 2008	Jan 2002	Aug 2009	False Trough: Nov 2008
Real Activity	May 2001	Nov 2008	Dec 2001	Sep 2009	None
Broad	May 2001	Nov 2008	Dec 2001	May 2009	None
LVQ					
Narrow	May 2001	May 2008	Feb 2002	Aug 2009	False Peak: Sep 2005
Real Activity	Sep 2001	Mar 2008	Mar 2002	Aug 2009	False Peak: Oct 2010
Broad	May 2001	Mar 2008	Feb 2002	Aug 2009	None
DFMS					
Narrow	May 2001	May 2008	Jun 2002	Sep 2009	False Peak: Sep 2005
Real Activity	Apr 2001	April 2008	Apr 2002	May 2010	None
Ensemble					
Narrow	May 2001	May 2008	Jan 2002	Aug 2009	False Peak: Sep 2005
Real Activity	Jul 2001	May 2008	Mar 2002	Sep 2009	None
Broad	Mar 2001	May 2008	Mar 2002	Sep 2009	None

Notes: This table shows the earliest month that an analyst would have identified the four NBER business cycle turning points over the January 2000 to October 2018 out-of-sample period using the supervised and unsupervised classifiers discussed in Sections 18.3 and 18.4. An entry of 'NA' means the relevant turning point was not identified prior to the NBER Business Cycle Dating Committee making the announcement of a new business cycle turning point.

2001 troughs until July 2003 and the June 2009 trough until September 2010. This performance was achieved with only a single false trough identified by one algorithm, Boosted Classification Trees. The DFMS classifier was somewhat slower to detect these troughs than the other classifiers, although still substantially faster than the NBER announcement. Finally, consistent with the results for peaks, larger datasets did not substantially improve the timeliness with which troughs were identified on average.

Given the small number of turning points in the out-of-sample period, it is hard to distinguish definitively between the performance of the individual classifiers. If one were forced to choose a single classifier for the purpose of identifying turning points, both the kNN classifier and the random forest classifier applied to the narrow dataset were quick to identify turning points while producing no false positives. The other classifiers had similar performance, but produced a single false positive. That said, the kNN classifier and random forest classifier both failed to identify business cycle peaks when applied to the larger datasets, which may give us some pause as to the robustness of these classifiers. If one looks more holistically across the various datasets, the boosting algorithm emerges as a possible favorite, as it identifies all four turning points for all four datasets, and does so with speed comparable to the top set of performers for each dataset. Finally, the ensemble classifier has overall performance similar to the boosting algorithm. We might also expect, given the potential effect of averaging on reducing classifier variance, that the ensemble classifier will be the most robust classifier across alternative out-of-sample periods.

18.6 Conclusion

In this chapter I have surveyed a variety of approaches for real-time classification of economic time-series data. Special attention was paid to the case where classification is conducted in a data-rich environment. Much of the discussion was focused on machine learning supervised classification techniques that are common to the statistical classification literature, but have only recently begun to be widely used in economics. I also presented a review of Markov-switching models, which is an unsupervised classification approach that has been commonly used in economics for both historical and real-time classification. Finally, I presented an application to real-time identification of U.S. business cycle turning points based on a wide dataset of 136 macroeconomic and financial time-series.

References

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

Ang, A., & Bekaert, G. (2002). Regime switches in interest rates. *Journal of Business and Economic Statistics*, *20*, 163–182.

Berge, T. (2015). Predicting recessions with leading indicators: Model averaging and selection over the business cycle. *Journal of Forecasting*, *34*(6), 455–471.

Berge, T., & Jordá, O. (2011). The classification of economic activity into expansions and recessions. *American Economic Journal: Macroeconomics*, *3*(2), 246–277.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *26*(2), 123–140.

Breiman, L. (2001). Random Forests. In *Machine learning* (pp. 5–32).

Camacho, M., Perez-Quiros, G., & Poncela, P. (2015). Extracting nonlinear signals from several economic indicators. *Journal of Applied Econometrics*, *30*(7), 1073–1089.

Camacho, M., Perez-Quiros, G., & Poncela, P. (2018). Markov-switching dynamic factor models in real time. *International Journal of Forecasting*, *34*, 598–611.

Chauvet, M. (1998). An econometric characterization of business cycle dynamics with factor structure and regime switching. *International Economic Review*, *39*, 969–996.

Chauvet, M., & Hamilton, J. D. (2006). Dating business cycle turning points. In P. R. Costas Milas & D. van Dijk (Eds.), *Nonlinear time series analysis of business cycles* (pp. 1–53). Elsevier, North Holland.

Chauvet, M., & Piger, J. (2008). A comparison of the real-time performance of business cycle dating methods. *Journal of Business and Economic Statistics*, *26*(1), 42–49.

Cosslett, S., & Lee, L.-F. (1985). Serial correlation in discrete variable models. *Journal of Econometrics*, *27*, 79–97.

D opke, J., Fritsche, U., & Pierdzioch, C. (2017). Predicting recessions with boosted regression trees. *International Journal of Forecasting*, *33*, 745–759.

Davig, T., & Smalter Hall, A. (2016). *Recession forecasting using Bayesian classification*. Federal Reserve Bank of Kansas City Research Working paper no. 1606.

Diebold, F. X., & Rudebusch, G. D. (1996). Measuring business cycles: A modern perspective. *The Review of Economics and Statistics*, *78*(1), 67–77.

Dueker, M. (1997). Markov switching in garch processes and mean-reverting stock-market volatility. *Journal of Business and Economic Statistics*, *15*, 26–34.

Estrella, A., & Mishkin, F. S. (1998). Predicting U.S. recessions: Financial variables as leading indicators. *The Review of Economics and Statistics*, *80*(1), 45–61.

Estrella, A., Rodrigues, A. P., & Schich, S. (2003). How stable is the predictive power of the yield curve? Evidence from Germany and the United States. *The Review of Economics and Statistics*, *85*(3), 629–644.

Evans, M., & Wachtel, P. (1993). Inflation regimes and the sources of inflation uncertainty. *Journal of Money, Credit and Banking*, *25*, 475–511.

Fossati, S. (2016). Dating U.S. business cycles with macro factors. *Studies in Nonlinear Dynamics and Econometrics*, *20*, 529–547.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, *121*(2), 256–285.

Freund, Y., & Schapire, R. (1996). Experiments with a new boosting algorithm. *Proceedings of ICML*, *13*, 148–156.

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

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, *28*(2), 337–407.

Fushing, H., Chen, S.-C., Berge, T., & Jordá, O. (2010). *A chronology of international business cycles through non-parametric decoding*. SSRN Working Paper no: 1705758.

Garbellano, J. (2016). *Nowcasting recessions with machine learning: New tools for predicting the business cycle* (Bachelor's Thesis, University of Oregon).

Garcia, R., & Schaller, H. (2002). Are the effects of monetary policy asymmetric? *Economic Inquiry*, *40*, 102–119.

Garcia, R., & Perron, P. (1996). An analysis of the real interest rate under regime shifts. *Review of Economics and Statistics*, *78*(1), 111–125.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*(1), 3–42.

Giusto, A., & Piger, J. (2017). Identifying business cycle turning points in real time with vector quantization. *International Journal of Forecasting*, *33*, 174–184.

Goldfeld, S. M., & Quandt, R. E. (1973). A markov model for switching regressions. *Journal of Econometrics*, *1*(1), 3–16.

Guidolin, M., & Timmermann, A. (2005). Economic implications of bull and bear regimes in uk stock and bond returns. *Economic Journal*, *115*(500), 111–143.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica*, *57*(2), 357–384.

Hamilton, J. D. (1994). *Time series analysis*. Princeton, NJ: Princeton University Press.

Hamilton, J. D. (2008). Regime switching models. In S. N. Durlauf & L. E. Blume (Eds.), *New palgrave dictionary of economics, 2nd edition*, Palgrave MacMillan.

Hamilton, J. D. (2011). Calling recessions in real time. *International Journal of Forecasting*, *27*(4), 1006–1026.

Hamilton, J. D., & Lin, G. (1996). Stock market volatility and the business cycle. *Journal of Applied Econometrics*, *11*, 573–593.

Hamilton, J. D., & Raj, B. (2002). New directions in business cycle research and financial analysis. *Empirical Economics*, *27*(2), 149–162.

Hamilton, J. D., & Susmel, R. (1994). Autoregressive conditional heteroskedasticity and changes in regime. *Journal of Econometrics*, *64*, 307–333.

Hamilton, J. D., Waggoner, D. F., & Zha, T. (2007). Normalization in econometrics. *Econometric Reviews*, *26*(2-4), 221–252.

Hammer, B., & Villmann, T. (2002). Generalized relevance learning vector quantization. *Neural Networks*, *15*(8-9), 1059–1068.

Harding, D., & Pagan, A. (2006). Synchronization of cycles. *Journal of Econometrics*, *132*(1), 59–79.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning. data mining, inference and prediction*. New York, NY: Springer.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(8), 832–844.

Karnizova, L., & Li, J. (2014). Economic policy uncertainty, financial markets and probability of U.S. recessions. *Economics Letters*, *125*(2), 261–265.

Kaufmann, S. (2002). Is there an asymmetric effect of monetary policy over time? A bayesian analysis using austrian data. *Empirical Economics*, *27*, 277–297.

Kauppi, H., & Saikkonen, P. (2008). Predicting u.s. recessions with dynamic binary response models. *The Review of Economics and Statistics*, *90*(4), 777–791.

Kim, C.-J. (1994). Dynamic linear models with markov switching. *Journal of Econometrics*, *60*(1-2), 1–22.

Kim, C.-J. (2004). Markov-switching models with endogenous explanatory variables. *Journal of Econometrics*, *122*, 127–136.

Kim, C.-J., Morley, J., & Piger, J. (2005). Nonlinearity and the permanent effects of recessions. *Journal of Applied Econometrics*, *20*(2), 291–309.

Kim, C.-J., & Nelson, C. R. (1998). Business cycle turning points, a new coincident index, and tests of duration dependence based on a dynamic factor model with regime switching. *Review of Economics and Statistics*, *80*(2), 188–201.

Kim, C.-J., & Nelson, C. R. (1999a). Friedman's plucking model of business fluctuations: Tests and estimates of permanent and transitory components. *Journal of Money, Credit and Banking*, *31*, 317–334.

Kim, C.-J., & Nelson, C. R. (1999b). Has the U.S. economy become more stable? A bayesian approach based on a Markov-switching model of the business cycle. *Review of Economics and Statistics*, *81*(4), 608–616.

Kim, C.-J., & Nelson, C. R. (1999c). *State-space models with regime switching*. Cambridge, MA: The MIT Press.

Kohonen, T. (2001). *Self-organizing maps*. Berlin: Spring-Verlag.

Lo, M. C., & Piger, J. (2005). Is the response of output to monetary policy asymmetric? Evidence from a regime-switching coefficients model. *Journal of Money, Credit and Banking*, *37*, 865–887.

Manela, A., & Moreira, A. (2017). News implied volatility and disaster concerns. *Journal of Financial Economics*, *123*(1), 137–162.

Mayr, A., Binder, H., Gefeller, O., & Schmid, M. (2014). The evolution of boosting algorithms: From machine learning to statistical modelling. *Methods of Information in Medicine*, *6*(1), 419–427.

McCracken, M. W., & Ng, S. (2015). *Fred-md: A monthly database for macroeconomic research*. St. Louis Federal Reserve Bank Working Paper no. 2015-012B.

Ng, S. (2014). Boosting recessions. *Canadian Journal of Economics*, *47*(1), 1–34.

Owyang, M. T., Piger, J., & Wall, H. J. (2015). Forecasting national recessions using state-level data. *Journal of Money, Credit and Banking*, *47*(5), 847–866.

Piger, J. (2009). Econometrics: Models of regime changes. In R. A. Meyers (Ed.), *Encyclopedia of complexity and system science* (pp. 2744–2757). Springer.

Qi, M. (2001). Predicting u.s. recessions with leading indicators via neural network models. *International Journal of Forecasting*, *17*, 383–401.

Ravn, M., & Sola, M. (2004). Asymmetric effects of monetary policy in the united states. *Federal Reserve Bank of St. Louis Review*, *86*, 41–60.

Rudebusch, G., & Williams, J. (2009). Forecasting recessions: The puzzle of the enduring power of the yield curve. *Journal of Business and Economic Statistics*, *27*(4), 492–503.

Sato, A., & Yamada, K. (1995). Generalized learning vector quantization. In D. T. G. Tesauro & T. Leen (Eds.), *Advances in neural information processing systems* (pp. 423–429). Elsevier, North Holland.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*, 197–227.

Sims, C. A., & Zha, T. (2006). Were there regime switches in U.S. monetary policy? *American Economic Review*, *96*(1), 54–81.

Stock, J. H., & Watson, M. W. (2002). Forecasting using principal components from a large number of predictors. *Journal of the American Statistical Association*, *97*, 1167–1179.

Stock, J. H., & Watson, M. W. (2014). Estimating turning points using large data sets. *Journal of Econometrics*, *178*(1), 368–381.

Turner, C. M., Startz, R., & Nelson, C. R. (1989). A Markov model of heteroskedasticity, risk, and learning in the stock market. *Journal of Financial Economics*, *25*(1), 3–22.

Vishwakarma, K. (1994). Recognizing business cycle turning points by means of a neural network. *Computational Economics*, *7*, 175–185.

Ward, F. (2017). Spotting the danger zone: Forecasting financial crises with classification tree ensembles and many predictors. *Journal of Applied Econometrics*, *32*, 359–378.