Chapter 11

Data Classification: Advanced Concepts

"*Labels are for filing. Labels are for clothing. Labels are not for people.*"—Martina Navratilova

11.1 Introduction

In this chapter, a number of advanced scenarios related to the classification problem will be addressed. These include more difficult special cases of the classification problem and various ways of enhancing classification algorithms with the use of additional inputs or a combination of classifiers. The enhancements discussed in this chapter belong to one of the following two categories:

1. *Difficult classification scenarios:* Many scenarios of the classification problem are much more challenging. These include multiclass scenarios, rare-class scenarios, and cases where the size of the training data is large.

2. *Enhancing classification:* Classification methods can be enhanced with additional data-centric input, user-centric input, or multiple models.

The difficult classification scenarios that are addressed in this chapter are as follows:

1. *Multiclass learning:* Although many classifiers such as decision trees, Bayesian methods, and rule-based classifiers, can be directly used for multiclass learning, some of the models, such as support-vector machines, are naturally designed for binary classification. Therefore, numerous meta-algorithms have been designed for adapting binary classifiers to multiclass learning.

2. *Rare class learning:* The positive and negative examples may be imbalanced. In other words, the data set contains only a small number of positive examples. A direct use of traditional learning models may often result in the classifier assigning all examples to the negative class. Such a classification is not very informative for imbalanced scenarios in which misclassification of the rare class incurs much higher cost than misclassification of the normal class.

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_11

3. *Scalable learning:* The sizes of typical training data sets have increased significantly in recent years. Therefore, it is important to design models that can perform the learning in a scalable way. In cases where the data is not memory resident, it is important to design algorithms that can minimize disk accesses.

4. *Numeric class variables:* Most of the discussion in this book assumes that the class variables are categorical. Suitable modifications are required to classification algorithms, when the class variables are numeric. This problem is also referred to as *regression modeling*.

The addition of more training data or the simultaneous use of a larger number of classification models can improve the learning accuracy. A number of methods have been proposed to enhance classification methods. Examples include the following:

1. *Semisupervised learning:* In these cases, unlabeled examples are used to improve the effectiveness of classifiers. Although unlabeled data does not contain any information about the label distribution, it does contain a significant amount of information about the manifold and clustering structure of the underlying data. Because the classification problem is a supervised version of the clustering problem, this connection can be leveraged to improve the classification accuracy. The core idea is that in most real data sets, labels vary in a smooth way over dense regions of the data. The determination of dense regions in the data only requires unlabeled information.

2. *Active learning:* In real life, it is often expensive to acquire labels. In active learning, the user (or an oracle) is actively involved in determining the most *informative* examples for which the labels need to be acquired. Typically, these are examples that provide the user the more accurate knowledge about the uncertain regions in the data, where the distribution of the class label is unknown.

3. *Ensemble learning:* Similar to the clustering and the outlier detection problems, ensemble learning uses the power of multiple models to provide more robust results for the classification process. The motivation is similar to that for the clustering and outlier detection problems.

This chapter is organized as follows. Multiclass learning is addressed in Sect. 11.2. Rare class learning methods are introduced in Sect. 11.3. Scalable classification methods are introduced in Sect. 11.4. Classification with numeric class variables is discussed in Sect. 11.5. Semisupervised learning methods are introduced in Sect. 11.6. Active learning methods are discussed in Sect. 11.7. Ensemble methods are proposed in Sect. 11.8. Finally, a summary of the chapter is given in Sect. 11.9.

11.2 Multiclass Learning

Some models such as support vector machines (*SVMs*), neural networks, and logistic regression are naturally designed for the binary class scenario. While multiclass generalizations of these methods are available, it is helpful to design generic meta-frameworks that can directly use the binary methods for multiclass classification. These frameworks are designed as meta-algorithms that can take a binary classification algorithm $\mathcal{A}$ as input and use it to make multilabel predictions. Several strategies are possible to convert binary classifiers into multilabel classifiers. In the following discussion, it will be assumed that the number of classes is denoted by k.

The first strategy is the *one-against-rest* approach. In this approach, k different binary classification problems are created, such that one problem corresponds to each class. In the ith problem, the ith class is considered the set of positive examples, whereas all the remaining examples are considered negative examples. The binary classifier $\mathcal{A}$ is applied to each of these training data sets. This creates a total of k models. If the positive class is predicted in the ith problem, then the ith class is rewarded with a vote. Otherwise, each of the remaining classes is rewarded with a vote. The class with the largest number of votes is predicted as the relevant one. In practice, more than one model may predict an example to belong to a positive class. This may result in ties. To avoid ties, one may also use the numeric output of a classifier (e.g., Bayes posterior probability) to weight the corresponding vote. The highest numeric score for a particular class is selected to predict the label. Note that the choice of the numeric score for weighting the votes depends on the classifier at hand. Intuitively, the score represents the "confidence" of that classifier in a particular label.

The second strategy is the *one-against-one* approach. In this strategy, a training data set is constructed for each of the $\binom{k}{2}$ pairs of classes. The algorithm $\mathcal{A}$ is applied to each training data set. This results in a total of $k(k-1)/2$ models. For each model, the prediction provides a vote to the winner. The class label with the most votes is declared as the winner in the end. At first sight, it seems that this approach is computationally more expensive, because it requires us to train $k(k-1)/2$ classifiers, rather than training k classifiers, as in the one-against-rest approach. However, the computational cost is ameliorated by the smaller size of the training data in the one-against-one approach. Specifically, the training data size in the latter case is approximately $2/k$ of the training data size used in the one-against-rest approach on the average. If the running time of each individual classifier scales super-linearly with the number of training points, then the overall running time of this approach may actually be lower than the first approach that requires us to train only k classifiers. This is usually the case for kernel SVM classifiers, in which the running times scale-up more than linearly with the number of data points. Note that the size of the kernel matrix scales up quadratically with the number of data points. The one-against-one approach may also result in ties between different classes that receive the same number of votes. In such cases, the numeric scores output by the classifier may be used to weight the votes for the different classes. As in the previous case, the choice of the numeric score depends on the choice of the base classifier model.

11.3 Rare Class Learning

The class distribution in many applications is not balanced. Consider a scenario in which data points representing credit card activity are labeled as either "normal" or "fraudulent." In such cases, the class distribution is typically very imbalanced. For example, 99 % of the data points may be normal, whereas only 1% of the data points may be fraudulent. The straightforward application of classification algorithms may lead to misleading results because of the preponderance of the normal class.

Consider a test instance $\overline{X}$ whose nearest 100 neighbors contain 49 rare class instances and 51 normal class instances. In such a case, it is evident that the test instance is surrounded by large fraction of rare instances *relative to expectation*. Yet, a k-nearest neighbor classifier with $k = 100$ will categorize instance $\overline{X}$ into the normal class. Such a classifier does not provide informative results, because its behavior approximately mimics a trivial classifier that classifies every instance as normal.

This behavior is not restricted to nearest-neighbor classifiers. A Bayesian classifier will have biased priors that favor the normal class. A decision-tree will find it difficult to separate out instances belonging to the rare class. As a result, most of these classifiers, if not modified appropriately, will classify many rare instances to the majority class. Interestingly, even a trivial classifier that labels all instances as normal might provide a high *absolute* accuracy. However, achieving a high classification accuracy on the rare class is more important in such application domains. This is because the applications associated with rare class detection are typically such that the *consequences* of misclassifying a rare class are much higher than those of misclassifying the normal class. For example, in the credit card scenario, it is much costlier to the credit card company to accept fraudulent activity as normal, rather than warning a customer incorrectly about suspicious activity on their card.

These observations suggest that rare-class learning algorithms need to have an explicit mechanism for emphasizing the greater importance of the rare class. This mechanism is provided by a *cost-matrix* $C(i, j)$ that quantifies the cost of misclassifying the class i to class j where $i \neq j$. In practice, for multiclass problems, it is often difficult to populate the full $k \times k$ matrix of misclassification possibilities. Therefore, a simplification is to associate the misclassification costs with the source class, rather than a source-destination pair. In other words, the cost of misclassifying class i is denoted by $C(i)$, irrespective of the incorrect destination class j to which it is predicted. Typically, the cost of misclassifying a rare class is much larger than that of misclassifying a normal class. Therefore, the goal is to maximize the *cost-weighted accuracy*, rather than the absolute accuracy.

Fortunately, these goals can be achieved by making modest changes to existing classification algorithms. Some examples of these modifications are as follows:

1. *Example reweighting:* The training examples from various classes are reweighted according to their misclassification costs. This approach naturally leads to a bias in classifying rare class examples more accurately than normal class examples. Therefore, classification algorithms need to be modified to work with weighted examples.

2. *Example resampling:* The examples from different classes are resampled to undersample normal classes and/or oversample rare classes. In such cases, unweighted classifiers can be directly used.

Each of these different methods will be discussed in the following sections.

11.3.1 Example Reweighting

In this case, the examples are weighted in proportion to their costs. Because the original classification problem is designed to maximize accuracy, the analogous solution to the weighted problem maximizes cost-weighted accuracy. Therefore, all instances belonging to the ith class are weighted by $C(i)$. Therefore, the existing classification algorithms need to be modified to work with these additional weights. In most cases, the required changes are relatively minor. The following contains a brief description of the required changes to various classification algorithms:

1. *Decision trees:* Weights can be incorporated in decision-tree algorithms easily. The split criterion requires the computation of the Gini index and entropy, all of which can be computed using weights on the examples. Both the Gini index and the entropy are computed as a function of the proportionate class distribution of the training examples. This proportionate class distribution can be computed with the use of

weights on the examples. Tree-pruning can also be modified to measure the impact of removing nodes on the weighted accuracy.

2. *Rule-based classifiers:* Sequential covering algorithms are similar to decision-tree construction. The main difference is in terms of the criteria used to grow rules. Measures such as the Laplace measure and FOIL's information gain use the raw number of positive and negative examples covered by the rule. In this case, the weighted number of examples are used as substitute for the raw number of examples. Rule-pruning uses weighted accuracy to measure the impact of conjunct pruning. For associative classifiers, the weights on the instances need to be used in computation of support and confidence.

3. *Bayes classifiers:* The implementation of Bayes classifiers remains virtually the same as the unweighted case except for one crucial difference in the probability estimation process. The class priors and conditional feature probabilities are now estimated using weights on the instances.

4. *Support vector machines:* Interestingly, the hard-margin support vector machines are not affected by reweighting of examples because the support vectors do not depend on example weights. However, in practice, soft margin is used. In such cases, the slack penalty terms in the objective function are appropriately weighted, and it results in modifications to both the primal and dual methods for soft SVMs (see Exercises 3 and 4). This typically leads to a movement of the boundary of the support-vector machine toward the normal class side of the separation. This ensures that fewer rare class examples are penalized for (the more costly) margin violation, and more normal class examples are penalized. The result is a lower likelihood of incorrectly misclassifying rare class examples but a greater likelihood of misclassifying normal class examples.

5. *Instance-based methods:* Weighted votes are used for the different classes, after determining the m nearest neighbors to a given test instance.

Thus, most classifiers can be made to work with the weighted case with relatively small changes. The advantage of weighting techniques is that they work with the original training data, and are therefore less prone to overfitting than sampling methods that manipulate the training data.

11.3.2 Sampling Methods

In adaptive resampling, the different classes are differentially sampled to enhance the impact of the rare class on the classification model. Sampling can be performed either with or without replacement. The rare class can be oversampled, or the normal class can be undersampled, or both can occur. The classification model is learned on the resampled data. The sampling probabilities are typically chosen in proportion to their misclassification costs. This enhances the proportion of the rare costs in the sample used for learning, and the approach is generally applicable to multiclass scenarios as well. It has generally been observed that undersampling the normal class has a number of advantages over oversampling the rare class. When undersampling is used, the sampled training data is much smaller than the original data set, which leads to better training efficiency.

In some variations, all instances of the rare class are used in combination with a small sample of the normal class. This is also referred to as *one-sided selection*. The logic of this approach is that rare class instances are too valuable as training data to modify any type

of sampling. Undersampling has several advantages with respect to oversampling because of the following reasons:

1. The model construction phase for a smaller training data set requires much less time.
2. The normal class is less important for modeling purposes, and all instances from the more valuable rare class are included for modeling. Therefore, the discarded instances do not impact the modeling effectiveness in a significant way.

11.3.2.1 Relationship Between Weighting and Sampling

Resampling methods can be understood as methods that sample the data in proportion to their weights, and then treat all examples equally. Therefore, the two methods are almost equivalent although sampling methods have greater randomness associated with them. A direct weight-based technique is generally more reliable because of the absence of this randomness. On the other hand, sampling can be more naturally combined with ensemble methods (cf. Sect. 11.8) such as bagging to improve accuracy. Furthermore, sampling has distinct *efficiency* advantages because it works with a much smaller data set. For example, for a data set containing a rare to normal ratio of 1:99, it is possible for a resampling technique to work effectively with 2 % of the original data when the data is resampled into an equal mixture of the normal and anomalous classes. This kind of resampling translates to a performance improvement of a factor of 50.

11.3.2.2 Synthetic Oversampling: SMOTE

One of the problems with oversampling the minority class is that a larger number of samples with replacement leads to repeated samples of the same data point. Repeated samples cause overfitting and reduce classification accuracy. In order to address this issue, a recent approach is to use synthetic oversampling that creates synthetic examples without repetition.

The *SMOTE* approach works as follows. For each minority instance, its k nearest neighbors belonging to the same class are determined. Then, depending on the level of oversampling required, a fraction of them are chosen randomly. For each sampled example-neighbor pair, a synthetic data example is generated on the line segment connecting that minority example to its nearest neighbor. The exact position of the example is chosen uniformly at random along the line segment. These new minority examples are added to the training data, and the classifier is trained with the augmented data. The *SMOTE* algorithm is generally more accurate than a vanilla oversampling approach. This approach forces the decision region of the resampled data to become more general than one in which only members from the rare classes in the *original* training data are oversampled.

11.4 Scalable Classification

In many applications, the training data sizes are rather large. This leads to numerous scalability challenges in building classification models. In such cases, the data will typically not fit in main memory, and therefore the algorithms need to be designed to optimize the accesses to disk. Although the traditional decision-tree algorithms, such as *C4.5*, work well for smaller data sets, they are not optimized to disk-resident data. One solution is to sample the training data, but this has the disadvantage of losing the learning knowledge in the discarded training instances. Some classifiers, such as associative classifiers and

nearest-neighbor methods, can be made faster by using more efficient subroutines for frequent pattern mining and nearest-neighbor indexing, respectively. Other classifiers, such as decision trees and support vector machines, require more careful redesign because they do not rely on any specific computationally intensive subroutines. These two classifiers are also particularly popular, and each of them is used widely in various data domains. Therefore, this chapter will specifically focus on these two classifiers in the context of scalability. An additional scalability challenge is created by *streaming* data, although such algorithms are not discussed in this chapter. The discussion of streaming data is deferred to Chap. 12.

11.4.1 Scalable Decision Trees

The construction of a decision tree can be computationally expensive because the evaluation of a split criterion at a node can sometimes be very slow. In the following, we will discuss two well-known methods for scalable decision tree construction.

11.4.1.1 RainForest

The *RainForest* approach is based on the insight that the evaluation of the split criteria in univariate decision trees do not need access to the data in its multidimensional form. Because each attribute value is analyzed independently in a univariate split, only the *count statistics* of distinct attributes values need to be maintained over different classes. For numeric data, it is assumed that they are discretized into categorical attribute values. The count statistics are collectively referred to as the *AVC-set*. The AVC-set is specific to a decision-tree node, and provides the counts of the distinct values of the attribute in the data records relevant to that node for different classes. Therefore, the size of the AVC-set depends *only* on the number of distinct attribute values and the number of classes. This size is often extremely small in comparison to the number of data records. Therefore, the memory requirement is dependent on the dimensionality of the data, the number of distinct values per dimension, and the number of classes. The larger the base training data set, the greater the proportional savings.

These AVC-sets are stored in main memory and used for efficiently evaluating the split criteria at the nodes. The splits are performed at nodes, until the AVC-sets no longer fit in main memory. The data does need to be scanned when the AVC-sets are constructed for newly created nodes. By carefully interleaving the splits and the AVC-set construction, significant computational and disk-access savings can be achieved.

11.4.1.2 BOAT

The *Bootstrapped Optimistic Algorithm for Tree construction (BOAT)* algorithm uses *bootstrapped samples* for decision-tree construction. In bootstrapping, the data is sampled with replacement to create b different bootstrapped samples. These are used to create b different trees denoted by $T_1 \ldots T_b$. Then, it is checked whether the choice of the split attributes and the splitting subsets are identical, at a particular node in the different bootstrapped trees. For nodes where this is not the case, they are deleted along with the corresponding subtrees. The bootstrapping is used to create an *information-coarse splitting criterion* where a confidence interval is imposed on the numeric attribute at each node. The width of this confidence interval can be controlled with the number of bootstrapped samples. At a later stage of the algorithm, the coarse splitting criterion is converted to an exact one by integrating the various confidence intervals of the splits into a crisp criterion. In effect, *BOAT*

uses the trees $T_1 \ldots T_b$ to create a new tree that is very close to one that would have been constructed, even if all the data had been available. The *BOAT* algorithm is faster than *RainForest*, and it requires only two scans over the database. Furthermore, *BOAT* also has the capability of performing incremental decision tree induction and can also handle tuple deletions.

11.4.2 Scalable Support Vector Machines

A major problem with support vector machines is that the size of the optimization problem scales with the number of training data points, and that the memory requirements may scale with the *square* of the number of data points in the case of kernel-based support vector machines. For example, consider the optimization problem for SVM discussed in Sect. 10.6 of Chap. 10. The kernel-based Lagrangian dual of the problem, as adapted from Eq. 10.62 in Chap. 10, may be written as follows:

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j K(\overline{X_i}, \overline{X_j}). \tag{11.1}$$

The number of Lagrangian parameters λ_i (or optimization variables) is equal to the number of training data points n, and the size of the kernel matrix $K(\overline{X_i}, \overline{X_j})$ is $O(n^2)$. As a result, the coefficients of the entire optimization problem cannot even be loaded in main memory for large values of n. The *SVMLight* approach is designed to address this issue. This approach is mainly based on the following two observations:

1. It is not necessary to solve the entire problem at one time. A subset (or *working set*) of the variables $\lambda_1 \ldots \lambda_n$ may be selected for optimization at a given time. Different working sets are selected and optimized iteratively to arrive at the global optimal solution.

2. The support vectors for the SVMs correspond to only a small number of training data points. Even if most of the other training data points were removed, it would have no impact on the decision boundary of the SVM. Therefore, the *early* identification of such data points during the computationally intensive training process is crucial for efficiency maximization.

The following observations discuss how each of the aforementioned observations may be leveraged. In the case of the first observation, an iterative approach is used, in which the set of variables of the optimization problem are improved iteratively by fixing the majority of the variables to their current value, and improving only a small working set of the variables. Note that the size of the *relevant* kernel matrix within each local optimization scales with the square of the size q of the working set S_q, rather than the number n of training points. The *SVMLight* algorithm repeatedly executes the following two iterative steps until global optimality conditions are satisfied:

1. Select q variables as the active working set S_q, and fix the remaining $n - q$ variables to their current value.

2. Solve $L_D(S_q)$, a smaller optimization subproblem, with only q variables.

A key issue is how the working set of size q may be identified in each iteration. Ideally, it is desired to select a working set for which the maximum improvement in the objective

function is achieved. Let $\overline{V}$ be a vector with length equal to the number of Lagrangian variables and at most q nonzero elements. The goal is to determine the optimal choice for the q nonzero elements to determine the working set. An optimization problem is set up for determining $\overline{V}$ in which the dot product of $\overline{V}$ with the gradient of L_D (with respect to the Lagrangian variables) is optimized. This is a separate optimization problem that needs to be solved in each iteration to determine the optimal working set.

The second idea for speeding up support vector machines is that of *shrinking* the training data. In the support vector machine formulation, the focus is primarily on the decision boundary. Training examples that are on the correct size of the margin, and far away from it, have no impact on the solution to the optimization problem, even if they are removed. The early identification of these training examples is required during the optimization process to benefit as much as possible from their removal. A heuristic approach, based on the Lagrangian multiplier estimates, is used in the *SVMLight* approach. The specific details of determining these training examples are beyond the scope of this book but pointers are provided in the bibliographic notes. Another later approach, known as *SVMPerf*, shows how to achieve linear scale-up, but for the case of the linear model only. For some domains, such as text, the linear model works quite well in practice. Furthermore, the *SVMPerf* method has $O(s \cdot n)$ complexity where s is the number of *nonzero features*, and n is the number of training examples. In cases where $s \ll d$, such a classifier is very effective. This is the case for sparse high-dimensional domains such as text and market basket data. Therefore, this approach will be described in Sect. 13.5.3 of Chap. 13 on text data.

11.5 Regression Modeling with Numeric Classes

In many applications, the class variables are numerical. In this case, the goal is to minimize the squared error of prediction of the numeric class variable. This variable is also referred to as the *response variable*, *dependent variable*, or *regressand*. The feature variables are referred to as *explanatory variables*, *input variables*, *predictor variables*, *independent variables*, or *regressors*. The prediction process is referred to as *regression modeling*. This section will discuss a number of such regression modeling algorithms.

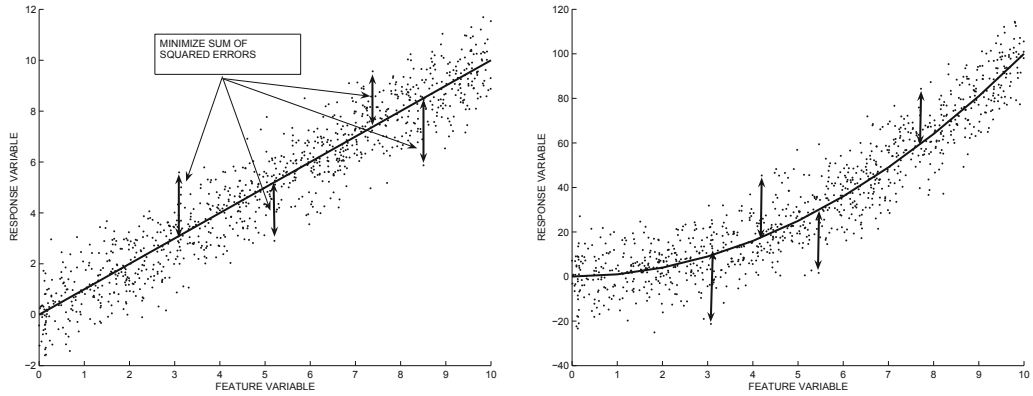
11.5.1 Linear Regression

Let D be an $n \times d$ data matrix whose ith data point (row) is the d-dimensional input feature vector $\overline{X_i}$, and the corresponding response variable is y_i. Let the n-dimensional column-vector of response variables be denoted by $\overline{y} = (y_1, \ldots y_n)^T$. In linear regression, the dependence of each response variable y_i on the corresponding independent variables $\overline{X_i}$ is modeled in the form of a linear relationship:

$$y_i \approx \overline{W} \cdot \overline{X_i} \quad \forall i \in \{1 \ldots n\}. \tag{11.2}$$

Here, $\overline{W} = (w_1 \ldots w_d)$ is a d-dimensional row vector of coefficients that needs to be learned from the training data so as to minimize the unexplained error $\sum_{i=1}^{n} (\overline{W} \cdot \overline{X_i} - y_i)^2$ of modeling. The response values of test instances can be predicted with this linear relationship. Note that a constant (bias) term is not needed on the right-hand side, because we can append an artificial dimension[1] with a value of 1 to each data point to include the constant term within $\overline{W}$. Alternatively, instead of using an artificial dimension, one can mean-center the

[1]Here, we assume that the total number of dimensions is d, including the artificial column.



(a) Linear regression $y = x$ (b) Nonlinear regression $y = x^2$

Figure 11.1: Examples of linear and nonlinear regression

data matrix and the response variable. In such a case, it can be shown that the bias term is not necessary (see Exercise 8). Furthermore, the standard deviations of all columns of the data matrix, except for the artificial column, are assumed to have been scaled to 1. In general, it is common to *standardize* the data in this way to ensure similar scaling and weighting for all attributes. An example of a linear relationship for a 1-dimensional feature variable is illustrated in Fig. 11.1a.

To minimize the squared-error of prediction on the training data, one must determine $\overline{W}$ that minimizes the following objective function O:

$$O = \sum_{i=1}^{n} (\overline{W} \cdot \overline{X_i} - y_i)^2 = ||D\overline{W}^T - \overline{y}||^2. \tag{11.3}$$

Using[2] matrix calculus, the gradient of O with respect to $\overline{W}$ can be shown to be the d-dimensional vector $2D^T(D\overline{W}^T - \overline{y})$. Setting the gradient to 0 yields the following d-dimensional vector of optimization conditions:

$$D^T D\overline{W}^T = D^T \overline{y}. \tag{11.4}$$

If the symmetric matrix $D^T D$ is invertible, then the solution for $\overline{W}$ can be derived from the aforementioned condition as $\overline{W}^T = (D^T D)^{-1} D^T \overline{y}$. The numerical class value of a previously unseen test instance $\overline{T}$ can then be predicted as the dot product between $\overline{W}$ and $\overline{T}$.

It is noteworthy that the matrix $(D^T D)^{-1} D^T$ is also referred to as the *Moore–Penrose pseudoinverse* D^+ of the matrix D. Therefore, the solution to linear regression can also be expressed as $D^+ \overline{y}$. The pseudoinverse is more generally defined even for the case where $D^T D$ is not invertible:

$$D^+ = \lim_{\delta \to 0} (D^T D + \delta^2 I)^{-1} D^T. \tag{11.5}$$

[2]Excluding constant terms, the objective function $O = (D\overline{W}^T - \overline{y})^T (D\overline{W}^T - \overline{y})$ can be expanded to the two additive terms $\overline{W} D^T D \overline{W}^T$ and $-(\overline{W} D^T \overline{y} + \overline{y}^T D \overline{W}^T) = -2\overline{W} D^T \overline{y}$. The gradients of these terms are $2D^T D\overline{W}^T$ and $-2D^T \overline{y}$, respectively. In the event that the Tikhonov regularization term $\lambda ||\overline{W}||^2$ is added to the objective function, an additional term of $2\lambda \overline{W}^T$ will appear in the gradient.

Here, I is a $d \times d$ identity matrix. When the number of training data points is small, all the training examples might lie on a hyperplane with dimensionality less than d. As a result, the $d \times d$ matrix $D^T D$ is not of full rank and therefore not invertible. In other words, the system of equations $D^T D\overline{W}^T = D^T \overline{y}$ is *underdetermined* and has infinitely many solutions. In this case, the general definition of the Moore–Penrose pseudoinverse in Eq. 11.5 is useful. Even though the inverse of $D^T D$ does not exist, the limit of Eq. 11.5 can still be computed. It is possible to compute D^+ using the *SVD* of D (cf. Sect. 2.4.3.4 of Chap. 2). More efficient computation methods are possible with the use of the following matrix identity (see Exercise 15):

$$D^+ = (D^T D)^+ D^T = D^T (DD^T)^+. \tag{11.6}$$

This identity is useful when $d \ll n$ or $n \ll d$. Here, we will show only the case where $d \ll n$ because the other case is very similar. The first step to diagonalize the $d \times d$ symmetric matrix $D^T D$:

$$D^T D = P \Lambda P^T. \tag{11.7}$$

The columns of P are the orthonormal eigenvectors of $D^T D$ and Λ is a diagonal matrix containing the eigenvalues. When the matrix $D^T D$ is of rank $k < d$, the pseudoinverse $(D^T D)^+$ of $D^T D$ is computed as follows:

$$(D^T D)^+ = P \Lambda^+ P^T. \tag{11.8}$$

Λ^+_{ii} is derived from Λ by setting it to $1/\Lambda_{ii}$ for the k nonzero entries, and 0, otherwise. Then, the solution for $\overline{W}$ is defined as follows:

$$\overline{W}^T = (D^T D)^+ D^T \overline{y}. \tag{11.9}$$

Even though the underdetermined system of equations $D^T D\overline{W}^T = D^T \overline{y}$ has infinitely many solutions, the pseudoinverse always provides a solution $\overline{W}$ with the smallest L_2-norm $||\overline{W}||$ among the alternatives. Smaller coefficients are desirable because they reduce overfitting. Overfitting is a significant problem in general, and especially so when the matrix $D^T D$ is not of full rank. A more effective approach is to use *Tikhonov regularization* or *Lasso*. In Tikhonov regularization, also known as *ridge regression*, a penalty term $\lambda||\overline{W}||^2$ is added to the objective function O of Eq. 11.3, where $\lambda > 0$ is a regularization parameter. In that case, the solution for $\overline{W}^T$ becomes $(D^T D + \lambda I)^{-1} D^T \overline{y}$, where I is a $d \times d$ identity matrix. The matrix $(D^T D + \lambda I)$ can be shown to be always positive-definite and therefore invertible. The compact solution provided by the Moore–Penrose pseudoinverse is a special case of Tikhonov regularization in which λ is infinitesimally small (i.e., $\lambda \to 0$). In general, the value of λ should be selected adaptively by cross-validation. In *Lasso*, an L_1-penalty, $\lambda \sum_{i=1}^{d} |w_i|$, is used instead of the L_2-penalty term. The resulting problem does not have a closed form solution, and it is solved using iterative techniques such as *proximal gradient methods* and *coordinate descent* [256]. *Lasso* has the tendency to select sparse solutions (i.e., few nonzero components) for $\overline{W}$, and it is particularly effective for high-dimensional data with many irrelevant features. *Lasso* can also be viewed as an embedded model (cf. Sect. 10.2 of Chap. 10) for feature selection because features with zero coefficients are effectively discarded. The main advantage of *Lasso* over ridge regression is not necessarily one of performance, but that of its highly interpretable feature selection.

Although the use of a penalty term for regularization might seem arbitrary, it creates stability by discouraging very large coefficients. By penalizing all regression coefficients, noisy features are often deemphasized to a greater degree. A common manifestation of overfitting

in linear regression is that the additive contribution of a large coefficient to $\overline{W} \cdot \overline{X}$ may be frequently canceled by another large coefficient in a small training data set. Such features might be noisy. This situation can lead to inaccurate predictions over unseen test instances because the response predictions are very sensitive to small perturbations in feature values. Regularization prevents this situation by penalizing large coefficients. Bayesian interpretations also exist for these regularization methods. For example, Tikhonov regularization assumes Gaussian priors on the parameters $\overline{W}$ and class variable. Such assumptions are helpful in obtaining a unique and probabilistically interpretable solution when the available training data is limited.

11.5.1.1 Relationship with Fisher's Linear Discriminant

Fisher's linear discriminant for binary classes (cf. Sect. 10.2.1.4 of Chap. 10) can be shown to be a special case of least-squares regression. Consider a problem with two classes, in which the two classes 0 and 1 contain a fraction p_0 and p_1, respectively, of the n data points. Assume that the d-dimensional mean vectors of the two classes are $\overline{\mu_0}$ and $\overline{\mu_1}$, and the covariance matrices are Σ_0 and Σ_1, respectively. Furthermore, it is assumed that the data matrix D is mean-centered. The response variables $\overline{y}$ are set to $-1/p_0$ for class 0 and $+1/p_1$ for class 1. Note that the response variables are also mean-centered as a result. Let us now examine the solution for $\overline{W}$ obtained by least-squares regression. The term $D^T\overline{y}$ is proportional to $\overline{\mu_1}^T - \overline{\mu_0}^T$, because the value of $\overline{y}$ is $-1/p_0$ for a fraction p_0 of the data records belonging to class 0, and it is equal to $1/p_1$ for a fraction p_1 of the data records belonging to class 1. In other words, we have:

$$\begin{aligned}(D^T D)\overline{W}^T &= D^T\overline{y} \\ &\propto \overline{\mu_1}^T - \overline{\mu_0}^T.\end{aligned}$$

For mean-centered data, $\frac{D^T D}{n}$ is equal to the covariance matrix. It can be shown using some simple algebra (see Exercise 21 of Chap. 10) that the covariance matrix is equal to $S_w + p_0 p_1 S_b$, where $S_w = (p_0\Sigma_0 + p_1\Sigma_1)$ and $S_b = (\overline{\mu_1} - \overline{\mu_0})^T(\overline{\mu_1} - \overline{\mu_0})$ are the (scaled) $d \times d$ within-class and between-class scatter matrices, respectively. Therefore, we have:

$$(S_w + p_0 p_1 S_b)\overline{W}^T \propto \overline{\mu_1}^T - \overline{\mu_0}^T. \tag{11.10}$$

Furthermore, the vector $S_b\overline{W}^T$ always points in the direction $\overline{\mu_1}^T - \overline{\mu_0}^T$ because $S_b\overline{W}^T = (\overline{\mu_1}^T - \overline{\mu_0}^T)\left[(\overline{\mu_1} - \overline{\mu_0})\overline{W}^T\right]$. This implies that we can drop the term involving S_b from Eq. 11.10 without affecting the constant of proportionality:

$$\begin{aligned}S_w\overline{W}^T &\propto (\overline{\mu_1}^T - \overline{\mu_0}^T) \\ (p_0\Sigma_0 + p_1\Sigma_1)\overline{W}^T &\propto (\overline{\mu_1}^T - \overline{\mu_0}^T) \\ \overline{W}^T &\propto (p_0\Sigma_0 + p_1\Sigma_1)^{-1}(\overline{\mu_1}^T - \overline{\mu_0}^T).\end{aligned}$$

It is easy to see that the vector $\overline{W}$ is the same as the Fisher's linear discriminant of Sect. 10.2.1.4 in Chap. 10.

11.5.2 Principal Component Regression

Because overfitting is caused by the large number of parameters in $\overline{W}$, a natural approach is to work with a reduced dimensionality data matrix. In principal component regression,

the largest $k \ll d$ principal components of the input data matrix D (cf. Sect. 2.4.3.1 of Chap. 2) with nonzero eigenvalues are determined. These principal components are the top-k eigenvectors of the $d \times d$ covariance matrix of D. Let the top-k eigenvectors be arranged in matrix form as the orthonormal columns of the $d \times k$ matrix P_k. The original $n \times d$ data matrix D is transformed to a new $n \times k$ data matrix $R = DP_k$. The new derived set of k-dimensional input variables $\overline{Z_1} \ldots \overline{Z_n}$, which are rows of R, are used as training data to learn a reduced k-dimensional set of coefficients $\overline{W}$:

$$y_i \approx \overline{W} \cdot \overline{Z_i}. \tag{11.11}$$

In this case, the k-dimensional vector of regression coefficients $\overline{W}$ can be expressed in terms of R as $(R^T R)^{-1} R^T \overline{y}$. This solution is identical to the previous case, except that a smaller and full-rank $k \times k$ matrix $R^T R$ is inverted. Prediction on a test instance $\overline{T}$ is performed after transforming it to this new k-dimensional space as $\overline{T} P_k$. The dot product between $\overline{T} P_k$ and $\overline{W}$ provides the numerical prediction of the test instance. The effectiveness of principal component regression is because of the discarding of the low-variance dimensions, which are either redundant directions (zero eigenvalues) or noisy directions (very small eigenvalues). If all directions are included after PCA-based axis rotation (i.e., $k = d$), then the approach will yield the same results as linear regression on the original data. It is common to standardize the data matrix D to zero mean and unit variance before performing PCA. In such cases, the test instances also need to be scaled and translated in an identical way.

11.5.3 Generalized Linear Models

The implicit assumption in linear models is that a constant change in the ith feature variable leads to a constant change in the response variable, which is proportional to w_i. However, such assumptions are inappropriate in many settings. For example, if the response variable is the height of a person, and the feature variable is the age, the height is not expected to vary linearly with age. Furthermore, the model needs to account for the fact that such variables can never be negative. In other cases, such as customer ratings, the response variables might take on integer values from a bounded range. Nevertheless, the elegant simplicity of linear models can still be leveraged in these settings. In generalized linear models (GLM), each response variable y_i is modeled as an *outcome* of a (typically exponential) probability distribution with mean $f(\overline{W} \cdot \overline{X_i})$ as follows:

$$y_i \sim \text{Probability distribution with mean } f(\overline{W} \cdot \overline{X_i}) \quad \forall i \in \{1 \ldots n\}. \tag{11.12}$$

This function $f(\cdot)$ is referred to as the *mean function*, and its inverse $f^{-1}(\cdot)$ is referred to as the *link function*. Although the same mean/link function can be used with different probability distributions, the selected mean/link functions and probability distributions are usually paired carefully to maximize effectiveness and interpretability of the model. If the observed responses are discrete (e.g., binary), it is possible to use a discrete probability distribution for y_i (e.g., Bernoulli), as long as its mean is $f(\overline{W} \cdot \overline{X_i})$. An example of this scenario is logistic regression. Some common examples of mean functions with their associated probability distribution assumptions are illustrated in the table below:

Link function	Mean function	Distribution assumption
Identity	$\overline{W} \cdot \overline{X}$	Normal
Inverse	$-1/(\overline{W} \cdot \overline{X})$	Exponential, Gamma
Log	$\exp(\overline{W} \cdot \overline{X})$	Poisson
Logit	$1/[1 + \exp(-\overline{W} \cdot \overline{X})]$	Bernoulli, Categorical
Probit	$\Phi(\overline{W} \cdot \overline{X})$	Bernoulli, Categorical

The link function regulates the nature of the response variable and its usability in a specific application. For example, the log, logit, and probit link functions are typically used to model the relative frequency of a discrete or categorical outcome. Because of the probabilistic modeling of the response variable, a maximum likelihood approach is used to determine the optimal parameter set $\overline{W}$, where the product of the probabilities (or probability densities) of the response variable outcomes is maximized. After estimating the parameters in $\overline{W}$, the expected response value of a test instance $\overline{T}$ is estimated as $f(\overline{W} \cdot \overline{T})$. Furthermore, the probability distribution of the response variable (with mean $f(\overline{W} \cdot \overline{T})$) may be used for detailed analysis.

An important special case of *GLM* is least-squares regression. In this case, the probability distribution of the response y_i is the normal distribution with mean $f(\overline{W} \cdot \overline{X_i}) = \overline{W} \cdot \overline{X_i}$ and constant variance σ^2. The relationship $f(\overline{W} \cdot \overline{X_i}) = \overline{W} \cdot \overline{X_i}$ follows from the fact that the link function is the identity function. The likelihood of the training data is as follows:

$$\begin{aligned}\text{Likelihood}(\{y_1 \ldots y_n\}) = \prod_{i=1}^{n} \text{Probability}(y_i) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - f(\overline{W} \cdot \overline{X_i}))^2}{2\sigma^2}\right) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \overline{W} \cdot \overline{X_i})^2}{2\sigma^2}\right) \\ &\propto \exp\left(-\frac{\sum_{i=1}^{n}(y_i - \overline{W} \cdot \overline{X_i})^2}{2\sigma^2}\right).\end{aligned}$$

In this special case, the maximum likelihood approach can be shown to be equivalent to the least-squares approach because the logarithm of the likelihood yields the scaled objective function of linear regression. Another specific example of the process of maximum likelihood estimation with the logit function and Bernoulli distribution is discussed in detail in Sect. 10.6 of Chap. 10. In this case, the *discrete* binary variable y_i is modeled from a Bernoulli distribution with mean function $f(\overline{W} \cdot \overline{X_i}) = 1/[1 + \exp(-\overline{W} \cdot \overline{X_i})]$:

$$y_i = \begin{cases} 1 & \text{with probability } 1/[1 + \exp(-\overline{W} \cdot \overline{X_i})] \\ 0 & \text{with probability } 1/[1 + \exp(\overline{W} \cdot \overline{X_i})]. \end{cases} \tag{11.13}$$

Note that[3] the mean of y_i still satisfies the mean function according to the table above. This special case of GLMs is referred to as *logistic regression*. Logistic regression can also be used for k-way categorical response values. In that case, a k-way categorical distribution is used, and its mean function maps to a k-dimensional vector to represent each outcome of the categorical variable. An added restriction is that the components of the k-dimensional vector must add to 1. Probit regression is a sister family of models to logit regression, in which the cumulative density function (CDF) $\Phi(\cdot)$ of a standard normal distribution

[3] A slightly different convention of $y_i \in \{-1, +1\}$ is used in Chap. 10 for notational convenience. In that case, the mean function would need to be adjusted to $\frac{1-\exp(-\overline{W} \cdot \overline{X})}{1+\exp(-\overline{W} \cdot \overline{X})}$.

is used instead of the logit function. *Ordered* probit regression can model ordered integer values within a range (e.g., ratings) for the response variable by using the quantiles of a standard normal distribution. The key insight of *GLM* is to choose the link function and distribution assumption judiciously depending on the nature of the observed response in a specific application. Generalized linear models can be viewed as a unification of large classes of regression models, such as linear regression, logistic regression, probit regression, and Poisson regression.

11.5.4 Nonlinear and Polynomial Regression

Linear regression cannot capture nonlinear relationships such as those in Fig. 11.1b. The basic linear regression approach can be used for nonlinear regression by using *derived input features*. For example, consider a new set of m features denoted by $h_1(\overline{X_j}) \ldots h_m(\overline{X_j})$ for the jth data point. Here, $h_i(\cdot)$ represents a nonlinear transformation function from the d-dimensional input feature space to 1-dimensional space. This results in a new $n \times m$ input data matrix. By applying linear regression on this derived data matrix, one is able to model relationships of the following form:

$$y = \sum_{i=1}^{m} w_i h_i(\overline{X}). \tag{11.14}$$

For example, in *polynomial regression*, the higher powers of each dimension up to order r are used as a new set of derived features. This approach expands the number of dimensions by a factor of r, but it allows greater expressiveness in terms of nonlinear relationships. The main disadvantage of the approach is that it expands the dimensionality of the parameter set $\overline{W}$, and can therefore result in overfitting. Therefore, it is important to use regularization.

Arbitrary nonlinear relationships can also be captured by methods such as *kernel ridge regression*. In order to use kernels, the main goal is to show that the closed-form solution to linear ridge regression can be expressed in terms of dot products between training and test instances. One way of achieving this goal is by formulating the dual of the linear ridge regression problem [448], and then using the kernel trick as in SVMs. A simpler approach is to make use of a specialized variant of the Sherman–Morrison–Woodbury identity in matrix algebra (see Exercise 14), which is true for any $n \times d$ data matrix D and scalar λ:

$$(D^T D + \lambda I_d)^{-1} D^T = D^T (DD^T + \lambda I_n)^{-1}. \tag{11.15}$$

Note that I_d is a $d \times d$ identity matrix, whereas I_n is an $n \times n$ identity matrix. For an unseen test instance $\overline{Z}$, which is expressed as a row vector, the prediction $F(\overline{Z})$ of linear regression is given by $\overline{Z}\,\overline{W}^T$. By substituting the closed-form solution of ridge regression for $\overline{W}^T$ and then making use of the aforementioned identity, we obtain:

$$F(\overline{Z}) = \overline{Z}(D^T D + \lambda I_d)^{-1} D^T \overline{y} = \overline{Z} D^T (DD^T + \lambda I_n)^{-1} \overline{y}. \tag{11.16}$$

Note that $\overline{Z}D^T$ is an n-dimensional row vector of dot products between the test instance $\overline{Z}$ and the n training instances. According to the kernel trick, we can replace this row vector with a vector $\overline{\kappa}$ containing the n kernel similarities between the test and training instances. Furthermore, the matrix DD^T contains the $n \times n$ dot products between the training instances. We can replace this matrix with the $n \times n$ kernel matrix K constructed on the training instances. Then, the prediction for test instance $\overline{Z}$ is as follows:

$$F(\overline{Z}) = \overline{\kappa}(K + \lambda I_n)^{-1} \overline{y}. \tag{11.17}$$

The kernel trick can also be applied to other variants of linear regression, such as Fisher's discriminant and logistic regression. The extension to Fisher's discriminant is straightforward because it is a special case of linear regression, whereas the derivation for kernel logistic regression uses the dual optimization formulation like SVMs.

11.5.5 From Decision Trees to Regression Trees

Regression trees are designed to model nonlinear relationships between the features and the response variable. If the regression model is constructed at each leaf node in a hierarchical partitioning of the data, locally optimized linear regression models can be obtained within each partition. Even when the relationship between the class variable and feature variables is nonlinear, a *local* linear approximation is quite effective. Each test instance can then be classified with its locally optimized linear regression model by determining its appropriate partition. This hierarchical partitioning is essentially a decision tree because the assigned partition of a test instance is determined by the split criteria at the internal nodes. The overall strategy of constructing a decision tree remains the same as in the case of categorical class variables. Similarly, the splits can use univariate (axis-parallel) splits on the feature variables, as in a traditional decision tree. However, changes need to be made to the splitting and pruning criteria because of the numeric class variable:

1. *Splitting criterion:* In the case of categorical classes, the splitting criterion uses the Gini index or entropy of the class variable as a qualitative measure to decide the splitting attribute. However, in the case of numeric classes, an error-based measure is used. The regression modeling approach of the previous section is applied to each child resulting from a potential split. The *aggregate* squared error of prediction of all the training data points in the different child nodes is computed. The split with the minimum aggregate squared error is selected among all possible splits at a particular node.

   The main *computational* problem with this approach is that a linear regression model needs to be constructed for *each possible* split. An alternative is to not use linear regression in the tree construction phase. The average variance of the numeric class variable in the children nodes resulting from a possible split is used as the quality criterion for split evaluation. In other words, the Gini index splitting criterion for the categorical class variable in traditional decision-tree construction is replaced with the variance of the numeric class variable. The linear regression models are constructed at the leaf nodes for prediction only *after* the entire tree has already been constructed. While this approach will result in larger trees, it is more practical from a computational point of view.

2. *Pruning criterion:* To minimize overfitting, a portion of the training data is not used for constructing the decision tree. This training data is then used for evaluating the squared error of prediction of the decision tree. A similar post-pruning strategy is used as the case of categorical class variables. Leaf nodes are iteratively removed if their removal improves accuracy on the validation set, until no more nodes can be removed.

The main drawback of this approach is that overfitting of the linear regression model is a real possibility when leaf nodes do not contain enough data. Therefore, a sufficient amount of training data is required to begin with. In such cases, regression trees can be very powerful because they can model complex nonlinear relationships.

11.5.6 Assessing Model Effectiveness

The effectiveness of linear regression models can be evaluated with a measure known as the R^2-*statistic*, or the *coefficient of determination*. The term $SSE = \sum_{i=1}^{n}(y_i - g(\overline{X_i}))^2$ yields the sum-of-squared error of prediction of regression. Here, $g(\overline{X})$ represents the linear model used for regression. The squared error of the response variable about its mean (or *total sum of squares*) is $SST = \sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{n}\frac{y_j}{n}\right)^2$. Then the fraction of unexplained variance is given by SSE/SST, and the R^2-statistic is as follows:

$$R^2 = 1 - \frac{SSE}{SST}. \tag{11.18}$$

This statistic always ranges between 0 and 1 for the case of linear models. Higher values are desirable. When the dimensionality is large, the adjusted R^2-statistic provides a more accurate measure:

$$R^2 = 1 - \frac{(n-d)}{(n-1)}\frac{SSE}{SST}. \tag{11.19}$$

The R^2-statistic is appropriate only for the case of linear models. For nonlinear models, it is possible for the R^2-statistic to be highly misleading or even negative. In such cases, one might directly use the SSE as a measure of the error.

11.6 Semisupervised Learning

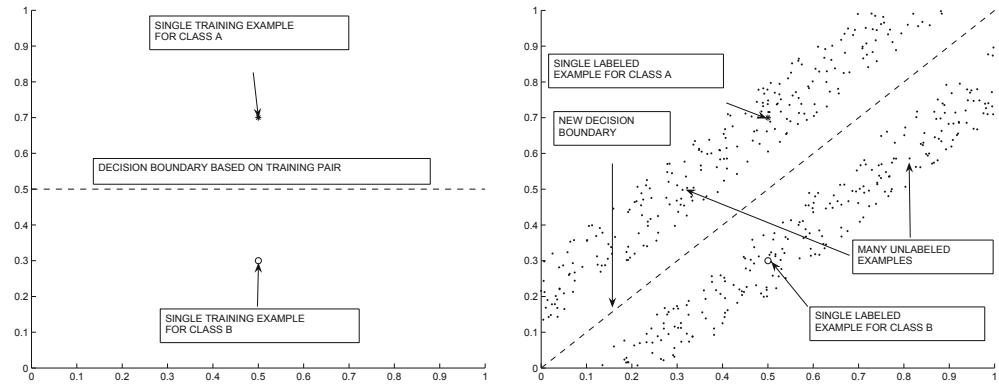
In many applications, labeled data is expensive and hard to acquire. On the other hand, unlabeled data is often copiously available. It turns out that unlabeled data can be used to significantly improve the accuracy of many mining algorithms. Unlabeled data is useful because of the following two reasons:

1. Unlabeled data can be used to estimate the low-dimensional manifold structure of the data. The available variation in label distribution can then be extrapolated on this manifold structure.

2. Unlabeled data can be used to estimate the joint probability distribution of features. The joint probability distributions of features are useful for indirectly relating feature values to labels.

The two aforementioned points are closely related. To explain these points, we will use two examples. In Fig. 11.2, an example has been illustrated where only *two* labeled examples are available. Based only on this training data, a reasonable decision boundary is illustrated in Fig. 11.2a. Note that this is the best decision boundary that one can hope to find with the use of this limited training data. Portions of this decision boundary are in regions of the space where almost no feature values are available. Therefore, the decision boundaries in these regions may not reflect the class behavior of *unseen* test instances.

Now, suppose that a large number of unlabeled examples are added to the training data, as illustrated in Fig. 11.2b. Because of the addition of these unlabeled examples, it becomes immediately evident that the data is distributed along two manifolds, each of which contains one of the training examples. A key assumption here is that the class variables are likely to vary *smoothly* over dense regions of the space, but it may vary significantly over sparse regions of the space. This leads to a new decision boundary that takes the underlying feature correlations into account in addition to the labeled instances. In the particular example of



(a) Labeled data (b) Labeled and unlabeled data

Figure 11.2: Impact of unlabeled data on classification

Fig. 11.2a, if a test instance were provided near the coordinates $(1, 0.7)$ with only the original training data, then almost any classifier, such as the nearest-neighbor classifier, will assign the data points to class A. However, this prediction is not reliable because of few previously seen labeled examples in the locality of the test instance. However, the unlabeled examples could be used to *expand* the labeled examples appropriately, by incrementally labeling the unlabeled examples in each hyperplane of Fig. 11.2b with the appropriate class. At this point, it becomes evident that test instances near the coordinates $(1, 0.7)$ really belong to class B.

A different way of understanding the impact of feature correlation estimation is by examining the intuitively interpretable text domain. Consider a scenario where one were trying to determine whether documents belong to the "Science" category. It is possible, that not enough *labeled* documents may contain the word "Einstein" in the documents. However, the word "Einstein" may often co-occur with other (more common) words such as "Physics" in *unlabeled* documents. At the same time, these more common words may already have been associated with the "Science" category because of their presence in labeled documents. Thus, the unlabeled documents provide the insight that the word "Einstein" is also relevant to the "Science" category. This example shows that unlabeled data can be used to learn *joint* feature distributions that are very relevant to the classification process.

Many of the semisupervised methods are often termed as *transductive* because they cannot handle out-of-sample test instances. In other words, all test instances need to be specified at the time of constructing the training model. New out-of-sample instances cannot be classified after the model has been constructed. This is different from most of the *inductive classifiers* discussed in the previous chapter in which training and testing phases are cleanly separated.

There are two primary types of techniques that are used for semisupervised learning. Some of these methods are *meta-algorithms* that can use any existing classification algorithm as a subroutine, and leverage it to incorporate the impact of unlabeled data. The second type of methods are those in which a number of modifications are incorporated in specific classifiers to account for the impact of unlabeled data. Two examples of the second type of methods are semisupervised Bayes classifiers, and transductive support vector machines. This section will discuss both these classes of techniques.

11.6.1 Generic Meta-algorithms

The goal of generic meta-algorithms is to use existing classification algorithms to enhance the classification process with unlabeled data. The simplest method is *self-training*, in which the smoothness assumption is used to incrementally expand the labeled portions of the training data. The major drawback of this approach is that it might lead to overfitting. One way of avoiding overfitting is by using *co-training*. Co-training partitions the feature space and independently labels instances using classifiers trained on each of these feature spaces. The labeled instances from one classifier are used as feedback to the other, and vice versa.

11.6.1.1 Self-Training

The self-training procedure can use any existing classification algorithm $\mathcal{A}$ as input. The classifier $\mathcal{A}$ is used to incrementally assign labels to unlabeled examples for which it has the most confident prediction. As input, the self-training procedure uses the initial labeled set L, the unlabeled set U, and a user-defined parameter k that may sometimes be set to 1. The self-training procedure iteratively uses the following steps:

1. Use algorithm $\mathcal{A}$ on the current labeled set L to identify the k instances in the unlabeled data U for which the classifier $\mathcal{A}$ is the most confident.

2. Assign labels to the k most confidently predicted instances and add them to L. Remove these instances from U.

It is easy to see that the self-training procedure will work very well for the simple example of Fig. 11.2. However, in practice, the different classes may not be quite as cleanly separated. The major drawback of self-training is that the addition of predicted labels to the training data can lead to propagation of errors in the presence of noise. Another procedure, known as *co-training*, is able to avoid such overfitting more effectively.

11.6.1.2 Co-training

In co-training, it is assumed that the feature set can be partitioned into two *disjoint* groups F_1 and F_2, such that each of them is sufficient to learn the target classification function. It is important to select the two feature subsets so that they are as independent from one another as possible. Two classifiers are constructed, such that one classifier is constructed on each of these groups. These classifiers are not allowed to interact with one another directly for prediction of unlabeled examples though they are used to build up training sets for each other. This is the reason that the approach is referred to as *co-training*.

Let L be the labeled training data and U be the unlabeled data. Let L_1 and L_2 be the labeled sets for each of these classifiers. The sets L_1 and L_2 are initialized to the available labeled data L, except that they are represented in terms of disjoint feature sets F_1 and F_2, respectively. Over the course of the co-training process, as different examples from the initially unlabeled set U are added to L_1 and L_2, respectively, the training instances in L_1 and L_2 may vary from one another. Two classifier models $\mathcal{A}_1$ and $\mathcal{A}_2$ are constructed using the training sets L_1 and L_2, respectively. The following steps are then iteratively applied:

1. Train classifier $\mathcal{A}_1$ using labeled set L_1, and add k most confidently predicted instances from unlabeled set $U - L_2$ to training data set L_2 for classifier $\mathcal{A}_2$.

2. Train classifier $\mathcal{A}_2$ using labeled set L_2, and add k most confidently predicted instances from unlabeled set $U - L_1$ to training data set L_1 for classifier $\mathcal{A}_1$.

In many implementations of the method, the most confidently labeled examples *for each class* are added to the training sets of the other classifier. This procedure is repeated until all instances are labeled. The two classifiers are then retrained with the expanded training data sets. This approach can be used to label not only the unlabeled data set U, but also unseen test instances. At the end of the procedure, two classifiers are returned. For an unseen test instance, each classifier may be used to determine the class label scores. The score for a test instance is determined by combining the scores of the two classifiers. For example, if the Bayes method is used as the base classifier, then the product of the posterior probabilities returned by the two classifiers may be used.

The co-training approach is more robust to noise because of the disjoint feature sets used by the two algorithms. An important assumption is that of *conditional independence* of the features in the two sets with respect to a particular class. In other words, after the class label is fixed, the features in one subset are conditionally independent of the other. The intuition for this is that instances generated by one classifier appear to be randomly distributed to the other, and vice versa. As a result, the approach will generally be more robust to noise than the self-training method.

11.6.2 Specific Variations of Classification Algorithms

The algorithms in the previous section were designed as generic meta-algorithms that can use virtually any known classification algorithm $\mathcal{A}$ for semisupervised learning. A few methods have also been designed that rely on variations of other classification algorithms, such as variations of the Bayes classifier and support vector machines.

11.6.2.1 Semisupervised Bayes Classification with EM

An important observation is that both the EM-clustering algorithm (cf. Sect. 6.5 of Chap. 6) and the naive Bayes classifier (cf. Sect. 10.5.1 of Chap. 10) use the same generative mixture model, wherein examples from each cluster (class) are generated from a predefined distribution, such as the Bernoulli or the Gaussian. In the case of the naive Bayes classifier, the iterative approach of the EM-algorithm is not required because the class memberships of the training data are already fixed, which makes the E-step unnecessary. In the case of semisupervised classification, however, the unlabeled examples need to be assigned to classes in order to expand the training data. Therefore, the iterative approach of the EM-algorithm again becomes essential. Semisupervised Bayes classification can be viewed as a combination of EM clustering and the naive Bayes classifier.

This method was originally proposed in the context of text data, although this discussion will assume categorical data for simplicity. Note that the binary representation of text data may also be considered categorical data. The naive Bayes algorithm requires the estimation of the conditional probabilities of the feature values for each class. Specifically, Eq. 10.22 in Sect. 10.5.1 of Chap. 10 requires the estimation of $P(x_j = a_j|C = c)$. This expression represents the conditional probability of the feature value, given the class and is estimated from the training data. The estimation cannot be performed accurately, if the number of training examples is small. Consider the case of the text domain. If only five to ten labeled documents are available for a particular class, and x_j is a binary variable corresponding to the presence or absence of a particular word j, then this estimation cannot be performed robustly. As discussed earlier, the joint distribution of features with labeled and unlabeled data can be very helpful in this respect.

Intuitively, the idea is to use the EM clustering algorithm to determine the clusters of documents most similar to the labeled classes. A partially supervised EM clustering method associates each cluster with a particular class. The conditional feature distributions in these clusters are used as a more robust proxy for the feature distributions of the corresponding classes.

The basic idea is to use a generative model to create semisupervised clusters from the data. A one-to-one correspondence between the mixture components and the classes is retained in this case. The use of EM algorithms for clustering categorical data and its semisupervised variant are discussed in Sects. 7.2.3 and 7.5.1, respectively, of Chap. 7. The reader is advised to revisit these sections for the relevant background before reading further.

For initialization, the labeled examples are used as the seeds for the EM algorithm, and the number of mixture components is set to the number of classes. A Bayes classifier is used to assign documents to clusters (classes) in the E-step. In the first iteration, the Bayes classifier uses only the labeled data to determine the initial set of posterior cluster (class) membership probabilities, as in a standard Bayes classifier. This results in a set of "soft" clusters, in which the (unlabeled) data point $\overline{X}$ has a weight $w(\overline{X}, c)$ in the range $(0, 1)$ associated with each class c, corresponding to its posterior Bayes membership probability. Only labeled documents have binary weights that are either 0 or 1 for each class, depending on their fixed assignments. The value of $P(x_j = a_j|C = c)$ is now estimated using a weighted variant of Eq. 10.22 in Chap. 10 that leverages *both* the labeled *and* the unlabeled documents.

$$P(x_j = a_j|C = c) = \frac{\sum_{\overline{X} \in \mathcal{L} \cup \mathcal{U}} w(\overline{X}, c) I(x_j, a_j)}{\sum_{\overline{X} \in \mathcal{L} \cup \mathcal{U}} w(\overline{X}, c)} \tag{11.20}$$

Here, $I(x_j, a_j)$ is an indicator variable that takes on the value of 1, if the jth feature x_j of $\overline{X}$ is a_j, and 0 otherwise. The major difference from Eq. 10.22 is that the posterior Bayes estimates of unlabeled documents are *also* used to estimate class-conditional feature distributions. As in the standard Bayes method, the same Laplacian smoothing approach may be incorporated to reduce overfitting. The prior probabilities $P(C = c)$ for each cluster may also be estimated by computing the average assignment probability of the data points to the corresponding class. This is the M-step of the EM algorithm. The next E-step uses these modified values of $P(x_j = a_j|C = c)$ and the prior probability to derive the posterior Bayes probability with a standard Bayes classifier. Therefore, the Bayes classifier implicitly incorporates the impact of unlabeled data. The algorithm may be summarized by the following two iterative steps that are continually repeated to convergence:

1. (E-step) Estimate posterior probability of membership of data points to clusters (classes) using Bayes rule.

$$P(C = c|\overline{X}) \propto P(C = c) \prod_{j=1}^{d} P(x_j = a_j|C = c) \tag{11.21}$$

2. (M-step) Estimate conditional distribution of features for different clusters (classes), using the current estimated posterior probabilities (unlabeled data) and known memberships (labeled data) of data points to clusters (classes).

One challenge with the use of the approach is that the clustering structure may sometimes not correspond to the class distribution very well. In such cases, the use of unlabeled data can harm the classification accuracy, as the clusters found by the EM algorithm

drift away from the true class structure. After all, unlabeled data are plentiful compared to labeled data, and therefore the estimation of $P(x_j = a_j|C = c)$ in Eq. 11.20 will be dominated by the unlabeled data. To ameliorate this effect, the labeled and unlabeled data are weighted differently during the estimation of $P(x_j = a_j|C = c)$. The unlabeled data are weighted down by a predefined discount factor $\mu < 1$ to ensure better correspondence between the clustering structure and the class distribution. In other words, the value of $w(\overline{X}, c)$ is multiplied with μ for only the unlabeled examples before estimating $P(x_j = a_j|C = c)$ in Eq. 11.20. The EM-approach for semisupervised classification is particularly remarkable because it demonstrates the link between semisupervised clustering and semisupervised classification, even though these two kinds of semisupervision are motivated by different application scenarios.

11.6.2.2 Transductive Support Vector Machines

The general assumption for most of the semisupervised methods is that the label values of unsupervised examples do not vary abruptly at densely populated regions of the data. In transductive support vector machines, this assumption is implicitly encoded by assigning labels to unsupervised examples that maximize the margin of the support vector machine. To understand this point, consider the example of Fig 11.2b. In this case, the margin of the SVM will be optimized only when the labels of the examples in the cluster containing the single example for class A, are also set to the same value A. The same is true for the unlabeled examples in the cluster containing the single label for class B. Therefore, the SVM formulation now needs to be modified to incorporate additional margin constraints, and binary decision variables for each unlabeled example. Recall from the discussion in Sect. 10.6 of Chap. 10 that the original SVM formulation was to minimize the objective function $\frac{||W||^2}{2} + C\sum_{i=1}^{n} \xi_i$, subject to the following constraints:

$$y_i(\overline{W} \cdot \overline{X_i} + b) \geq 1 - \xi_i \quad \forall i. \tag{11.22}$$

In addition, the nonnegativity constraint $\xi_i \geq 0$ on the slack variables is observed. Note that the value of y_i is *known*, because the training examples are labeled. For the case of unlabeled examples, binary decision *variables* $z_i \in \{-1, +1\}$ (with corresponding slack penalties) are incorporated for each *unlabeled* training example $\overline{X_i} \in \mathcal{U}$. These decision variables correspond to the assignment of the unlabeled examples to a particular class. The following constraint is added to the optimization problem:

$$z_i(\overline{W} \cdot \overline{X_i} + b) \geq 1 - \xi_i \quad \forall i : \overline{X_i} \in \mathcal{U}. \tag{11.23}$$

The slack penalties for the unlabeled examples can also be included in the optimization objective function. Note that, unlike y_i, the value of z_i is not known, and it is a binary *integer* variable that becomes a part of the optimization problem. Furthermore, the modified optimization formulation is an integer program, which is far more difficult than the original convex optimization problem for support vector machines.

A number of techniques have, therefore, been designed to approximately solve this problem with iterative mechanisms. One of these methods starts by labeling the most confidently predicted examples and iteratively expanding them. The number of positive examples initially labeled from the unlabeled instances, is based on the required trade-off between precision and recall. This ratio of positive to negative examples is maintained throughout the iterative algorithm. In each iteration, one positive example is changed to negative, and one negative example to positive to improve the soft margin of the classifier as much as possible. The bibliographic notes contain a discussion of the methods commonly used in this context.

11.6.3 Graph-Based Semisupervised Learning

The conversion of arbitrary data types to graphs is discussed in Sect. 2.2.2.9 of Chap. 2. Therefore, one advantage of this approach is that it can be used for semisupervised classification of arbitrary data types, as long as a distance function is available for quantifying proximity between data objects. This is a property that graph-based methods inherit from their origins in nearest-neighbor classification. The steps in graph-based semisupervised learning are as follows:

1. Construct a similarity graph on both the labeled and the unlabeled data records. Each data object O_i is associated with a node in the similarity graph. Each object is connected to its k-nearest neighbors.

2. The weight w_{ij} of the edge (i, j) is equal to a kernelized function of the distance $d(O_i, O_j)$ between the objects O_i and O_j, so that larger weights indicate greater similarity. A typical example of the weight is based on the *heat kernel* [90]:

$$w_{ij} = e^{-d(O_i,O_j)^2/t^2}. \tag{11.24}$$

   Here, t is a user-defined parameter.

This problem is one where we have a graph containing both labeled and unlabeled nodes. It is now desired to infer the labels of the unlabeled nodes with the use of these proximity relationships. This problem is exactly identical to the *collective classification problem* introduced in Sect. 19.4 of Chap. 19. Readers are advised to refer to the methods discussed in that section.

Graph-based semisupervised learning may be viewed as a semisupervised extension of nearest-neighbor classifiers. The only difference of graph-based semisupervised methods from nearest-neighbor classifiers is the way in which similarity graphs are constructed. Nearest-neighbor methods can be conceptually viewed as collective classification methods on similarity graphs in which edges are added only between pairs of labeled and unlabeled instances. Nearest-neighbor classification simply selects the dominant label from the labeled nodes incident on an unlabeled node. In the semisupervised case, edges can be added between any pair of nodes, whether they are labeled or unlabeled. The addition of these extra edges is necessary in semisupervised learning because of the scarcity of the labeled nodes in the similarity graph. Such edges are able to associate unlabeled clusters of arbitrary shape to their closest labeled instances more effectively. The reader is referred to Sect. 19.4 of Chap. 19 for discussions on collective classification.

11.6.4 Discussion of Semisupervised Learning

An important question in semisupervised learning is whether unlabeled data always helps in improving classification accuracy. Semisupervised learning depends on the inherent class structure of the underlying data. For semisupervised learning to be effective, the class structure of the data should approximately match its clustering structure. This assumption is obvious in the case of the semisupervised EM algorithm. The assumption is, however, implicitly used by other methods as well.

In practice, semisupervised learning is most effective when the number of labeled examples is extremely small, and there is no realistic way of making confident predictions about scarcely populated regions of the space. In some domains, such as node classification of graphs, this is almost always true. Therefore, in such domains, the transductive setting is

the only way in which classification can be performed. These methods will be discussed in detail in Sect. 19.4 of Chap. 19. On the other hand, when a lot of labeled data is already available, then the unlabeled examples do not provide much advantage to the learner, and can, in fact, be harmful in some cases.

11.7 Active Learning

From the discussion in the previous section on semisupervised classification, it is evident that labeled data are often scarce in real applications. While labeled data are often expensive to obtain, the cost of procuring labeled data can often be quantified. Some examples of costly labeling mechanisms are as follows:

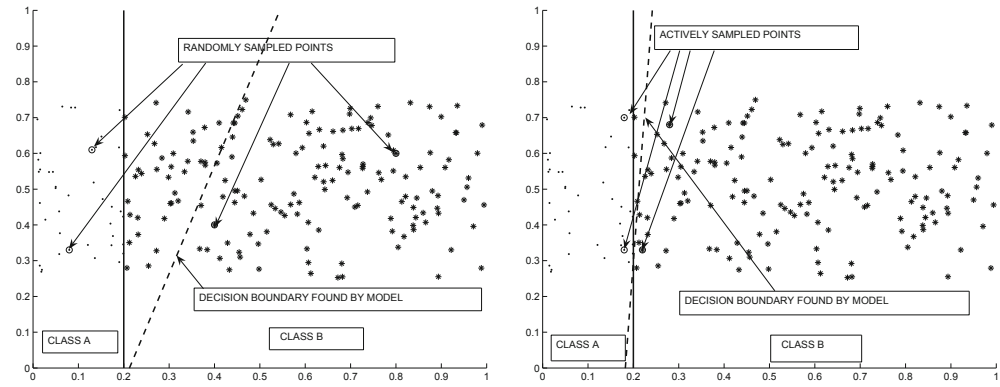
- *Document collections:* Large amounts of document data, which are usually unlabeled, are available on the Web. A common approach is to manually label the documents, which is a slow, painstaking, and laborious process. Alternatively, crowdsourcing mechanisms, such as Amazon Mechanical Turk, may be used. However, such mechanisms typically incur a dollar-cost on a per-instance basis.

- *Privacy-constrained data sets:* In many scenarios, the labels on records may be sensitive information that can be acquired at a significant query cost (e.g., obtaining permission from the relevant entity). In such cases, costs are harder to quantify explicitly, but can nevertheless be estimated through modeling.

- *Social networks:* In social networks, it may be desirable to identify nodes with specific properties. For example, an advertising company may desire to identify social network nodes that are interested in "cosmetics." However, it is rare that labels will be explicitly associated with the nodes. Identification of relevant nodes may require either manual examination of social network posts or user surveys. Both processes are time-consuming and costly.

It is clear from the aforementioned examples that the acquisition of labels should be viewed as a *cost-centric process* that helps improve modeling accuracy. The goal in active learning is to maximize the accuracy of classification at a specific cost of label acquisition. Therefore, active learning *integrates label acquisition and model construction*. This is different from all the other algorithms discussed in this book, where it is assumed that training data labels are already available.

Not all training examples are equally *informative*. To illustrate this point, consider the two-class problem illustrated in Fig. 11.3. The two classes, labeled by A and B, respectively, have a vertical decision boundary separating them. Suppose that the acquisition of labels is so costly that one is only allowed to acquire the labels of four examples from the entire data set and use this set of four examples to train a model. Clearly, this is a very small number of training examples, and the wrong choice of training examples may lead to significant overfitting. For example, in the case of Fig. 11.3a, the four examples have been randomly sampled from the data set. A typical linear classifier, such as logistic regression, may determine a decision boundary, corresponding to the dashed line in Fig. 11.3a. It is evident that this decision boundary is a poor representation of the true (vertical) decision boundary. On the other hand, in the case of Fig. 11.3b, the sampled examples are chosen more carefully to align along the true decision boundary. This set of labeled examples will result in a much better classification model for the entire data set. The goal in active learning is to integrate the labeling and classification process in a single framework to create



(a) Random sampling (b) Active sampling

Figure 11.3: Impact of active sampling on decision boundary

robust models. In practice, the determination of the correct choice of query instances is a very challenging problem. The key is to use the knowledge gained from the labels already acquired to "guess" the most informative regions in which to query the labels. Such an approach can help discover the true shape of the decision boundary as quickly as possible. Therefore, the key question in active learning is as follows:

How do we select instances to label to create the most accurate model at a given cost?

In some scenarios, the labeling cost may be instance-specific cost, although most models use the simplifying assumption of equal costs over all instances. Every active learning system has two primary components, one of which is already given:

1. *Oracle:* The oracle provides the responses to the underlying query in the form of labels of specified test instances. The oracle may be a human labeler, or a cost-driven data-acquisition system, such as Amazon Mechanical Turk. In general, for modeling purposes, the oracle is viewed as a black-box that is part of the input to the process.

2. *Query system:* The job of the query system is to pose queries to the oracle for labels of specific records. The querying strategy typically uses the distribution of currently known set of training instance labels to determine the most informative regions for querying.

The design of the query system may depend on the application at hand. For example, some query systems use *selective sampling*, in which a sequence of examples are presented to the user who makes a decision about whether or not to query them. The pool-based sampling approach assumes the availability of a base "pool" of instances from which to query the labels of data points. The task of the learner is to, therefore, determine (informative) instances one by one from this pool for querying.

The pool-based approach is the most common scenario for active learning, and will therefore be discussed in this chapter. The overall approach in the procedure is an iterative one. In each iteration, a number of interesting instances are identified, for which the addition of labels would be most informative for further classification. These are considered the "important" instances. The identification of the important instances is the job of the query-system, whereas the determination of the labels of queried instances is the job of the oracle,

which, in some cases, might be a human expert. The iterative process is repeated until either the cost budget is exhausted or the classification accuracy no longer improves with further addition of labels.

It is evident that the crucial part of active learning is the choice of the querying strategy. How should this querying be performed? From the example of Fig. 11.3, it is evident that the most effective querying strategies can map out the *boundaries* of separation most clearly. Because the boundary regions often contain instances of multiple classes, they are characterized by class label uncertainty or disagreements between different learners about the class label. This is, of course, not always true because uncertain regions may sometimes contain unrepresentative outliers. Therefore, the various models work with different assumptions about the most appropriate methodology for identifying the most informative query points.

1. *Heterogeneity-based models:* These models attempt to sample regions of the space that are uncertain, heterogeneous, or dissimilar to what has already been seen so far. Examples of such models include *uncertainty sampling*, *query-by-committee*, and *expected model change*. These models are based on the assumption that regions near the decision boundary are more likely to be heterogeneous and instances in these regions are more valuable for learning the decision boundary.

2. *Performance-based models:* These models directly use performance measures of classifiers such as expected error or variance reduction. Therefore, these models quantify the impact of adding the queried instance to the classifier performance on remaining unlabeled instances.

3. *Representativeness-based models:* These models attempt to create data, that is as representative as possible, of the underlying population of training instances. For example, it may be desired that the density distribution of the queried instances matches that of the training data. However, a heterogeneity criterion is often retained within the query model.

In the following, a brief discussion of each of these different types of models is provided.

11.7.1 Heterogeneity-Based Models

The goal in these models is to determine regions of greatest heterogeneity. The typical approach is to use the current set of training labels to examine the classification uncertainty of unseen instances with respect to available labels. This heterogeneity may be quantified in various ways, such as by measuring the uncertainty of classification, dissimilarity with the current model, or disagreement between a committee of classifiers.

11.7.1.1 Uncertainty Sampling

In uncertainty sampling, the learner attempts to label those instances for which the value of the label is the least certain. For example, the posterior probability of a Bayes classifier may be used to quantify its uncertainty. The Bayes classifier is trained on instances whose labels are already available. A binary-label instance is deemed as uncertain when its posterior class probabilities are as close to 0.5 as possible. The corresponding criterion may be formalized as follows:

$$\text{Certain}(\overline{X}) = \sum_{i=1}^{k} ||p_i - 0.5||. \tag{11.25}$$

The value lies in the range $(0, 1)$, and lower values are indicative of greater uncertainty. In the multiclass scenario, a formal entropy measure may be used to quantify uncertainty. If the Bayes posterior probabilities of the k classes are $p_1 \ldots p_k$, respectively, based on the current set of labeled instances, then the entropy measure Entropy$(\overline{X})$ is defined as follows:

$$\text{Entropy}(\overline{X}) = -\sum_{i=1}^{k} p_i \log(p_i). \tag{11.26}$$

In this case, larger values of the entropy indicate greater uncertainty and are more desirable for label acquisition.

11.7.1.2 Query-by-Committee

In this case, the heterogeneity is measured in terms of the disagreement of different classifiers rather than the posterior probabilities of a single classifier over different labels. This criterion, however, tries to achieve the same intuitive goal, but in a different way. Intuitively, when the posterior probability of a Bayes classifier is the same across different classes, a significant disagreement may exist between different classification models about the predicted label. Therefore, this approach uses a committee of different classifiers that are trained on the current set of labeled instances. These classifiers are then used to predict the class label of each unlabeled instance. The instance for which the classifiers disagree the most is selected as the relevant one in this scenario.

At an intuitive level, the query-by-committee method achieves similar heterogeneity goals as the uncertainty sampling method. Different classifiers are more likely to disagree on the class label for instances near the true decision boundary. The mathematical formula for quantifying the disagreement is also the same as uncertainty sampling. In particular, the posterior probability p_i of each class i in Eq. 11.26 is replaced with the fraction of votes received by each class i. It is particularly beneficial to use diverse classifiers that use fundamentally different modeling methodologies.

11.7.1.3 Expected Model Change

In this approach, the instance with the greatest expected change from the current classification model by adding a particular instance to the training data is selected. In many optimization-based classification models, such as discriminative probabilistic models, the gradient of the model objective function with respect to the model parameters can be quantified. By adding a queried instance to the training data, the gradient will change as well. The instance with the greatest change in the gradient when the queried instance is added to set of labeled instances. The intuition is that such an instance is likely to be very different from the model constructed using already labeled instances. Let $\delta g_i(\overline{X})$ be the change in the gradient with respect to the model parameters, conditional on the fact that the correct training label of the candidate instance $\overline{X}$ is the ith class. In other words, if the current labeled training set is L and $\overline{\nabla G(L)}$ is the gradient of the objective function with respect to model parameters, we have:

$$\delta g_i(\overline{X}) = ||\overline{\nabla G(L \cup (\overline{X}, i))} - \overline{\nabla G(L)}||. \tag{11.27}$$

Of course, we do not yet know the training label of $\overline{X}$, but we can only estimate the posterior probability of each label with a Bayes classifier. Let p_i be the posterior probability of the

class i with respect to the current label set of known labels in the training data. Then, the expected model change $C(\overline{X})$ with respect to the instance $\overline{X}$ is defined as follows:

$$C(\overline{X}) = \sum_{i=1}^{k} p_i \cdot \delta g_i(\overline{X}).$$

The instance $\overline{X}$ with the largest value of $C(\overline{X})$ is queried for the label.

11.7.2 Performance-Based Models

Although the motivation of heterogeneity-based models is that uncertain regions are the most informative by virtue of their proximity to decision boundaries, they have a drawback as well. Querying uncertain regions can inadvertently lead to the addition of unrepresentative outliers to the training data. Performance-based classifiers are therefore focused directly on the classification objective function. Therefore, these methods evaluate the accuracy of classification on the *remaining unlabeled instances*.

11.7.2.1 Expected Error Reduction

For the purpose of discussion, the remaining set of instances that has not yet been labeled is denoted by V. This set is used as the validation set on which the expected error reduction is computed. This approach is related to uncertainty sampling in a complementary way. Whereas uncertainty sampling *maximizes* the label uncertainty of the *queried* instance, the expected error reduction *minimizes* the expected label uncertainty of the *remaining* instances V when the queried instance is added to the training data. Thus, in the case of a binary-classification problem, the predicted posterior probabilities of the instances in V should be as far away from 0.5 as possible *after adding the queried instance*. The idea here is that greater certainty in prediction of class labels *of the remaining unlabeled instances*, will eventually result in a lower error rate on an unseen test set as well. Thus, error-reduction models can also be considered as *greatest certainty* models, except that the certainty criterion is applied to the instances in V rather than the query instance itself. Let $p_i(\overline{X})$ denote the posterior probability of the label i for the query candidate instance $\overline{X}$ with a Bayes model trained on the current set of labeled instances. Let $P_j^{(\overline{X},i)}(\overline{Z})$ be the posterior probability of class label j, when the instance-label combination $(\overline{X}, i)$ is added to the current set of labeled instances. Then, the error objective function $E(\overline{X}, V)$ for the *binary* class problem (i.e., $k = 2$) is defined as follows:

$$E(\overline{X}, V) = \sum_{i=1}^{k} p_i(\overline{X}) \left(\sum_{j=1}^{k} \sum_{\overline{Z} \in V} ||P_j^{(\overline{X},i)}(\overline{Z}) - 0.5|| \right). \tag{11.28}$$

The objective function can be interpreted as the expected label certainty of *remaining test instances*. Therefore, the objective function is maximized rather than minimized, as in the case of uncertainty-based models.

This result can easily be extended to the case of k-way models by using the same entropy criterion that was discussed for uncertainty-based models. In that case, the aforementioned expression is modified to replace $||P_j^{(\overline{X},i)}(\overline{Z}) - 0.5||$ with the class-specific entropy term $-P_j^{(\overline{X},i)}(\overline{Z})\log(P_j^{(\overline{X},i)}(\overline{Z}))$. Furthermore, this criterion needs to be minimized.

11.7.2.2 Expected Variance Reduction

One observation about the aforementioned error-reduction method of Eq. 11.28 is that it needs to be computed in terms of the entire set of unlabeled instances in V, and a new model needs to be trained incrementally to test the effect of adding a new instance. This can be computationally expensive. It should be pointed out that when the error of an instance set reduces, the corresponding variance also typically reduces. The overall generalization error can be expressed[4] as a sum of the true label noise, model bias, and variance. Of these, only the last term is highly dependent on the choice of instances selected. Therefore, it is possible to reduce the variance instead of the error, and the main advantage of doing so is the reduction in computational requirements. The main advantage of these techniques is the ability to express the variance in *closed form*, and therefore achieve greater computational efficiency. A detailed description of this class of methods is beyond the scope of this book. Refer to the bibliographic notes.

11.7.3 Representativeness-Based Models

The main advantage of performance-based models over heterogeneity-based models is that they intend to improve the error behavior on the *aggregate* set of unlabeled instances, rather than evaluating the uncertainty behavior of the *queried* instance. Therefore, unrepresentative or outlier-like queries are avoided. In some models, the representativeness itself becomes a part of the criterion for querying. One way of measuring representativeness is with the use of a density-based criterion, in which the density of a region in the space is used to weight the querying criterion. This weight is combined with a heterogeneity-based query criterion. Therefore, such methods can be considered a variation of the heterogeneity-based model, but with a representativeness weighting to ensure that outliers are not selected.

Therefore, these methods *combine* the heterogeneity behavior of the queried instance with a representativeness function from the unlabeled set V to decide on the queried instance. The representativeness function weights dense regions of the input space. The objective function $O(\overline{X}, V)$ of such a model is expressed as the product of a heterogeneity component $H(\overline{X})$ and a representativeness component $R(\overline{X}, V)$.

$$O(\overline{X}, V) = H(\overline{X})R(\overline{X}, V)$$

The value of $H(\overline{X})$ (assumed to be a maximization function) can be any of the heterogeneity criteria (transformed appropriately for maximization), such as the entropy criterion from uncertainty sampling, or the expected model change criterion. The representativeness criterion $R(\overline{X}, V)$ is simply a measure of the density of $\overline{X}$ with respect to the instances in V. A simple version of this density is the average similarity of $\overline{X}$ to the instances in V. Many other sophisticated variations of this simple measure are used. The reader is referred to the bibliographic notes for a discussion of the available measures.

11.8 Ensemble Methods

Ensemble methods are motivated by the fact that different classifiers may make different predictions on test instances due to the specific characteristics of the classifier, or their sensitivity to the random artifacts in the training data. An ensemble method is an approach to increase the prediction accuracy by combining the results from multiple classifiers. The

[4]This theoretical concept is discussed in detail in the next section.

Algorithm *EnsembleClassify*(Training Data Set: $\mathcal{D}$
Base Algorithms: $\mathcal{A}_1 \ldots \mathcal{A}_r$, Test Instances: $\mathcal{T}$)
begin
$j = 1$;
repeat
Select an algorithm $\mathcal{Q}_j$ from $\mathcal{A}_1 \ldots \mathcal{A}_r$;
Create a new training data set $f_j(\mathcal{D})$ from $\mathcal{D}$;
Apply $\mathcal{Q}_j$ to $f_j(\mathcal{D})$ to learn model $\mathcal{M}_j$;
$j = j + 1$;
until(termination);
report labels of each $T \in \mathcal{T}$ based on combination of
predictions from all learned models $\mathcal{M}_j$;
end

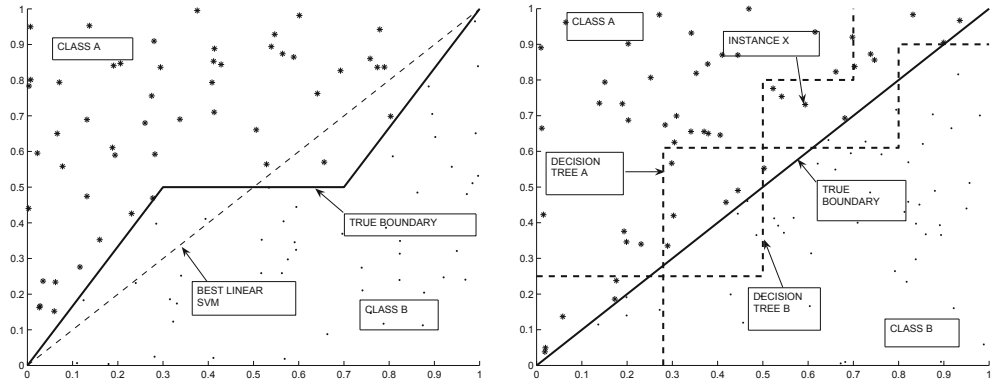
Figure 11.4: The generic ensemble framework

basic approach of ensemble analysis is to apply the *base ensemble learners* multiple times by using either different models, or by using the same model on different subsets of the training data. The results from different classifiers are then combined into a single robust prediction.

Although there are significant differences in how the individual learners are constructed and combined by various ensemble models, we start with a very generic description of ensemble algorithms. Later in this section, we will discuss specific *instantiations* of this broad framework, such as bagging, boosting, and random decision trees. The ensemble approach uses a set of base classification algorithms $\mathcal{A}_1 \ldots \mathcal{A}_r$. Note that these learners might be completely different algorithms, such as decision trees, SVMs, or the Bayes classifier. In some types of ensembles, such as boosting and bagging, a single learning algorithm is used but with different choices of training data. Different learners are used to leverage the greater robustness of different algorithms in different regions of the data. Let the learning algorithm selected in the jth iteration be denoted by $\mathcal{Q}_j$. It is assumed that $\mathcal{Q}_j$ is selected from the base learners. At this point, a *derivative training data set* $f_j(\mathcal{D})$ from the base training data is selected. This may be a random sample of the training data, as in bagging, or it may be based on the results of the past execution of ensemble components, as in boosting. A model $\mathcal{M}_j$ is learned in the jth iteration by applying the selected learning algorithm $\mathcal{Q}_j$ to $f_j(\mathcal{D})$. For each test instance T, a prediction is made by combining the results of different models $\mathcal{M}_j$ on T. This combination may be performed in various ways. Examples include the use of simple averaging, the use of a weighted vote, or the treatment of the model combination process as a learning problem. The overall ensemble framework is illustrated in Fig. 11.4.

The description of Fig. 11.4 is very generic, and allows significant flexibility in terms of how the ensemble components may be learned and the combination may be performed. The two primary types of ensembles are special cases of the description of Fig. 11.4:

1. *Data-centered ensembles:* A single base learning algorithm (e.g., an SVM or a decision tree) is used, and the primary variation is in terms of how the derivative data set $f_j(\mathcal{D})$ for the jth ensemble component is constructed. In this case, the input to the algorithm contains only a single learning algorithm $\mathcal{A}_1$. The data set $f_j(\mathcal{D})$ for the jth component of the ensemble may be constructed by sampling the data, focusing on incorrectly classified portions of the training data in previously executed ensemble components, manipulating the features of the data, or manipulating the class labels in the data.



(a) bias (b) variance

Figure 11.5: Impact of bias and variance on classification accuracy

2. *Model-centered ensembles:* Different algorithms $\mathcal{Q}_j$ are used in each ensemble iteration. In these cases, the data set $f_j(\mathcal{D})$ for each ensemble component is the same as the original data set $\mathcal{D}$. The rationale for these methods is that different models may work better in different regions of the data, and therefore the combination of the models may be more effective for any given test instance, as long as the specific errors of a classification algorithm are not reflected by the majority of the ensemble components on any particular test instance.

The following discussion introduces the rationale for ensemble analysis before presenting specific instantiations.

11.8.1 Why Does Ensemble Analysis Work?

The rationale for ensemble analysis can be best understood by examining the different components of the error of a classifier, as discussed in statistical learning theory. There are three primary components to the error of a classifier:

1. *Bias:* Every classifier makes its own modeling assumptions about the nature of the decision boundary between classes. For example, a linear SVM classifier assumes that the two classes may be separated by a linear decision boundary. This is, of course, not true in practice. For example, in Fig. 11.5a, the decision boundary between the different classes is clearly not linear. The correct decision boundary is shown by the solid line. Therefore, no (linear) SVM classifier can classify all the possible test instances correctly even if the best possible SVM model is constructed with a very large training data set. Although the SVM classifier in Fig. 11.5a seems to be the best possible approximation, it obviously cannot match the correct decision boundary and therefore has an inherent error. In other words, any given linear SVM model will have an inherent *bias*. When a classifier has high bias, it will make *consistently incorrect* predictions over particular choices of test instances near the incorrectly modeled decision-boundary, even when different samples of the training data are used for the learning process.

2. *Variance:* Random variations in the choices of the training data will lead to different models. Consider the example illustrated in Fig. 11.5b. In this case, the true decision

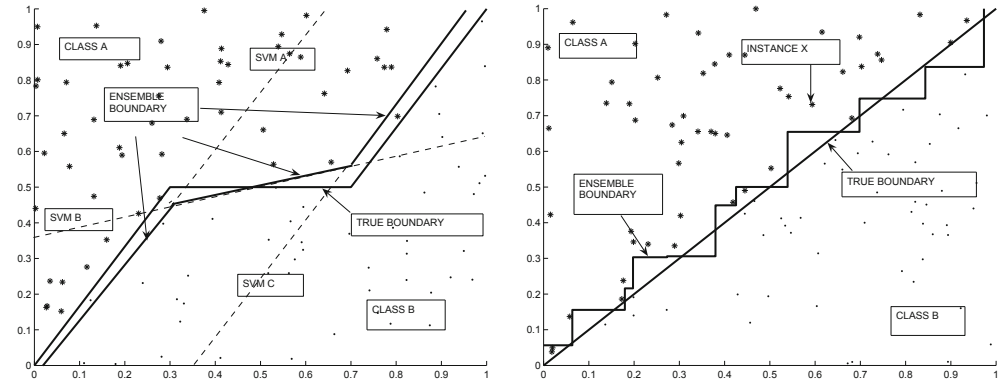
boundary is linear. A *sufficiently deep* univariate decision tree can approximate a linear boundary quite well with axis-parallel piecewise approximations. However, *with limited training data*, even when the trees are grown to full depth without pruning, the piecewise approximations will be coarse like the boundaries illustrated for hypothetical decision trees A and B in Fig. 11.5b. Different choices of training data might lead to different split choices, as a result of which the decision boundaries of trees A and B are very different. Therefore, (test) instances such as X are *inconsistently classified* by decision trees which were created by different choices of training data sets. This is a manifestation of model *variance*. Model variance is closely related to overfitting. When a classifier has an overfitting tendency, it will make *inconsistent* predictions for the same test instance over different training data sets.

3. *Noise:* The noise refers to the intrinsic errors in the target class labeling. Because this is an intrinsic aspect of data quality, there is little that one can do to correct it. Therefore, the focus of ensemble analysis is generally on reducing bias and variance.

Note that the design choices of a classifier often reflect a trade-off between the bias and the variance. For example, pruning a decision tree results in a more stable classifier and therefore reduces the variance. On the other hand, because the pruned decision tree makes stronger assumptions about the simplicity of the decision boundary than the unpruned tree, the former leads to greater bias. Similarly, using a larger number of neighbors for a nearest-neighbor classifier will lead to larger bias but lower variance. In general, simplified assumptions about the decision boundary lead to greater bias but lower variance. On the other hand, complex assumptions reduce bias but are harder to robustly estimate with limited data. The bias and variance are affected by virtually every design choice of the model, such as the choice of the base algorithm or the choice of model parameters.

Ensemble analysis can often be used to reduce both the bias and variance of the classification process. For example, consider the case of the example illustrated in Fig. 11.5a, in which the decision boundary is not linear, and therefore any linear SVM classifier will not find the correct decision boundary. However, by using different choices of model parameters, or data subset selection, it is possible to create three different linear SVM hyperplanes A, B, and C, as illustrated in Fig. 11.6a. Note that these different classifiers tend to work well in different parts of the data and have different *directions* of bias in any particular part of the data. This kind of differential performance on different parts of the data is sometimes artificially induced in ensemble components in some methods, such as boosting. In other cases, it may be a natural result of using ensemble model components that are very different from one another (e.g., decision trees and Bayes classifiers). Now consider a new ensemble classifier that is created using the majority vote of the three aforementioned classifiers corresponding to hyperplanes A, B, and C. The decision boundary of this ensemble classifier is illustrated in Fig. 11.6a as well. *This decision boundary is not linear and has lower bias with respect to the true decision boundary.* The reason for this is that different classifiers have different levels and directions of bias in different parts of the training data, and the majority vote across the different classifiers is able to obtain results that are generally less biased in any specific region than each of the component classifiers.

A similar argument applies to the variance example illustrated in Fig. 11.5b. Although instances such as X are inconsistently classified because of model variance, they will often be classified correctly when the model bias is low. As a result, by using the aggregation over sufficiently independent classifiers, it becomes increasingly likely that instances close to the decision boundary, such as X, will be correctly classified. For example, a majority vote of just three *independent* trees, each of which classifies X correctly with 80 % probability,



(a) bias (b) variance

Figure 11.6: Ensemble decision boundaries are more refined than those of component classifiers

will be correct $(0.8^3 + \binom{3}{2} \times 0.8^2 \times 0.2) \times 100 \approx 90\%$ of the time. In other words, the ensemble decision boundary of the majority classifier will be much closer to the true decision boundary than that of *any* of its component classifiers. In fact, a realistic example of how an ensemble boundary might look like after combining a set of relatively coarse decision trees, is illustrated in Fig. 11.6b. Note that the ensemble boundary is much closer to the true boundary because it is not regulated by the unpredictable variations in decision-tree behavior for a training data set of limited size. Such an ensemble is, therefore, better able to make use of the knowledge in the training data.

In general, different classification models have different sources of bias and variance. Models that are too simple (such as a linear SVM or shallow decision tree) make too many assumptions about the shape of the decision boundary, and will therefore have high bias. Models that are too complex (such as a deep decision tree) will overfit the data, and will therefore have high variance. Sometimes a different parameter setting in the same classifier will favor different parts of the bias-variance trade-off curve. For example, a small value of k in a nearest-neighbor classifier will result in lower bias but higher variance. Because different kinds of ensembles learners have different impacts on bias and variance, it is important to choose the component classifiers, so as to optimize the impact on the bias-variance trade-off. An overview of the impact of different models on bias and variance is provided in Table 11.1.

11.8.2 Formal Statement of Bias-Variance Trade-off

In the following, a formal statement of the bias-variance trade-off will be provided. Consider a classification problem with a training data set $\mathcal{D}$. The classification problem can be viewed as that of learning the function $f(\overline{X})$ between the feature variables $\overline{X}$ and the binary class variable y:

$$y = f(\overline{X}) + \epsilon. \tag{11.29}$$

Here, $f(\overline{X})$ is the function representing the true (but unknown) relationship between the feature variables and the class variable, and ϵ is the intrinsic error in the data that cannot be modeled. Therefore, over the n test instances $(\overline{X_1}, y_1) \ldots (\overline{X_n}, y_n)$, the intrinsic noise ϵ_a^2

Table 11.1: Impact of different techniques on bias-variance trade-off

Technique	Source/level of bias	Source/level of variance
Simple models	Oversimplification increases bias in decision boundary	Low variance. Simple models do not overfit
Complex models	Generally lower than simple models. Complex boundary can be modeled	High variance. Complex assumptions will be overly sensitive to data variation
Shallow decision trees	High bias. Shallow tree will ignore many relevant split predicates	Low variance. The top split levels do not depend on minor data variations
Deep decision trees	Lower bias than shallow decision tree. Deep levels model complex boundary	High variance because of overfitting at lower levels
Rules	Bias increases with fewer antecedents per rule	Variance increases with more antecedents per rule
Naive Bayes	High bias from simplified model (e.g., Bernoulli) and naive assumption	Variance in estimation of model parameters. More parameters increase variance
Linear models	High bias. Correct boundary may not be linear	Low variance. Linear separator can be modeled robustly
Kernel SVM	Bias lower than linear SVM. Choice of kernel function	Variance higher than linear SVM
k-NN model	Simplified distance function such as Euclidean causes bias. Increases with k	Complex distance function such as local discriminant causes variance. Decreases with k
Regularization	Increases bias	Reduces variance

may be estimated as follows:

$$\epsilon_a^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\overline{X}_i))^2. \tag{11.30}$$

Because the precise form of the function $f(\overline{X})$ is unknown, most classification algorithms construct a model $g(\overline{X}, \mathcal{D})$ based on certain *modeling assumptions*. An example of such a modeling assumption is the linear decision boundary in SVM. This function $g(\overline{X}, \mathcal{D})$ may be defined either algorithmically (e.g., decision tree), or it may be defined in closed form, such as the SVM classifier, discussed in Sect. 10.6 of Chap. 10. In the latter case, the function $g(\overline{X}, \mathcal{D})$ is defined as follows:

$$g(\overline{X}, \mathcal{D}) = \text{sign}\{\overline{W} \cdot \overline{X} + b\}. \tag{11.31}$$

Note that the coefficients $\overline{W}$ and b can only be *estimated* from the training data set $\mathcal{D}$. The notation $\mathcal{D}$ occurs as an argument of $g(\overline{X}, \mathcal{D})$ because the training data set $\mathcal{D}$ is required to estimate model coefficients such as $\overline{W}$ and b. For a training data set $\mathcal{D}$ of limited size, it is often not possible to estimate $g(\overline{X}, \mathcal{D})$ very accurately, as in the case of the coarse decision boundaries of Fig. 11.5b. Therefore, the specific impact of the training data set on the estimated result $g(\overline{X}, \mathcal{D})$ can be quantified by comparing it with its *expected value* $E_{\mathcal{D}}[g(\overline{X}, \mathcal{D})]$ over all possible outcomes of training data sets $\mathcal{D}$.

Aside from the intrinsic error ϵ_a^2, which is *data-set specific*, there are two primary sources of error in the modeling and estimation process:

1. The *modeling assumptions* about $g(\overline{X}, \mathcal{D})$ may not reflect the true model. Consider the case where the linear SVM modeling assumption is used for $g(\overline{X}, \mathcal{D})$, and the true boundary between the classes is not linear. This would result in a *bias* in the model. In

practice, the bias usually arises from oversimplification in the modeling assumptions. Even if we had a very large training data set and could somehow estimate the expected value of $g(\overline{X}, \mathcal{D})$, the value of $(f(\overline{X}) - E_{\mathcal{D}}[g(\overline{X}, \mathcal{D})])^2$ would be nonzero because of the *bias* arising from the difference between the true model and assumed model. This is referred to as the (squared) *bias*. Note that it is often possible to reduce the bias by assuming a more complex form for $g(\overline{X}, \mathcal{D})$, such as using a kernel SVM instead of a linear SVM in Fig. 11.5a.

2. Even if our assumptions about $g(\overline{X}, \mathcal{D})$ were correct, it is not possible to *exactly* estimate $E_{\mathcal{D}}[g(\overline{X}, \mathcal{D})]$ with any given training data set $\mathcal{D}$. Over different instantiations of a training data set $\mathcal{D}$ and fixed test instance $\overline{X}$, the predicted class label $g(\overline{X}, \mathcal{D})$ would be different. This is the model *variance*, which corresponds to $E_{\mathcal{D}}[(g(\overline{X}, \mathcal{D}) - E_{\mathcal{D}}[g(\overline{X}, \mathcal{D})])^2]$. Note that the expectation function $E_{\mathcal{D}}[g(\overline{X}, \mathcal{D})]$ defines a decision boundary which is usually much closer to the true decision boundary (e.g., ensemble boundary estimate in Fig. 11.6b) as compared to that defined by a specific instantiation $\mathcal{D}$ of the training data (e.g., boundaries A and B in Fig. 11.5b).

It can be shown that the expected mean squared error of the prediction $E_{\mathcal{D}}[MSE] = \frac{1}{n}\sum_{i=1}^{n} E_{\mathcal{D}}[(y_i - g(\overline{X_i}, \mathcal{D}))^2]$ of test data points $(\overline{X_1}, y_1) \ldots (\overline{X_n}, y_n)$ over different choices of training data set $\mathcal{D}$ can be decomposed into the bias, variance, and the intrinsic error as follows:

$$E_{\mathcal{D}}[MSE] = \frac{1}{n}\sum_{i=1}^{n} \left(\underbrace{(f(\overline{X_i}) - E_{\mathcal{D}}[g(\overline{X_i}, \mathcal{D})])^2}_{\text{Bias}^2} + \underbrace{E_{\mathcal{D}}[(g(\overline{X_i}, \mathcal{D}) - E_{\mathcal{D}}[g(\overline{X_i}, \mathcal{D})])^2]}_{\text{Variance}} \right) + \underbrace{\epsilon_a^2}_{\text{Error}}.$$

The ensemble method reduces the classification error by reducing the bias and variance of the combined model. By carefully choosing the component models with specific bias-variance properties and combining them appropriately, it is often possible to reduce both the bias and the variance over an individual model.

11.8.3 Specific Instantiations of Ensemble Learning

A number of ensemble approaches have been designed to increase accuracy by reducing bias, variance, or a combination of both. In the following, a discussion of selected models is provided.

11.8.3.1 Bagging

Bagging, which is also referred to as bootstrapped aggregating, is an approach that attempts to reduce the variance of the prediction. It is based on the idea that if the variance of a prediction is σ^2, then the variance of the average of k independent and identically distributed (i.i.d.) predictions is reduced to $\frac{\sigma^2}{k}$. Given sufficiently independent predictors, such an approach of averaging will reduce the variance significantly.

So how does one approximate i.i.d. predictors? In bagging, data points are sampled uniformly from the original data with replacement. This sampling approach is referred to as *bootstrapping* and is also used for model evaluation. A sample of approximately the same size as the original data is drawn. This sample may contain duplicates, and typically contains approximately $1 - (1 - 1/n)^n \approx 1 - 1/e$ fraction of the *distinct* data points in the original data. Here, e represents the base of the natural logarithm. This result is easy to show because

the probability of a data point *not* being included in the sample is given by $(1 - 1/n)^n$. A total of k different bootstrapped samples, each of size n, are drawn independently, and the classifier is trained on each of them. For a given test instance, the predicted class label is reported by the ensemble as the dominant vote of the different classifiers.

The primary advantage of bagging is that it reduces the model variance. However, bagging does not reduce the model bias. Therefore, this approach will not be effective for the example of Fig. 11.5a, but it will be effective for the example of Fig. 11.5b. In Fig. 11.5b, the different decision tree boundaries are created by the random variations in the bootstrapped samples. The *majority* vote of these bootstrapped samples will, however, perform better than a model constructed on the full data set because of a reduction in the variance component. For cases in which the bias is the primary component of error, the bootstrapping approach may show a slight degradation in accuracy. Therefore, while designing a bootstrapping approach, it is advisable to design the individual components to reduce the bias at the possible expense of variance, because bagging will take care of the latter. Such a choice optimizes the bias-variance trade-off. For example, one might want to grow a deep decision tree with 100 % class purity at the leaves. In fact, decision trees are an ideal choice for bagging, because they have low bias and high variance when they are grown sufficiently deep. The main problem with bagging is that the i.i.d. assumption is usually not satisfied because of correlations between ensemble components.

11.8.3.2 Random Forests

Bagging works best when the individual predictions from various ensemble components satisfy the i.i.d. property. If the k different predictors, each with variance σ^2, have a positive pairwise correlation of ρ between them, then the variance of the averaged prediction can be shown to be $\rho \cdot \sigma^2 + \frac{(1-\rho) \cdot \sigma^2}{k}$. The term $\rho \cdot \sigma^2$ is invariant to the number of components k in the ensemble. This term limits the performance gains from bagging. As we will discuss below, the predictions from bootstrapped decision trees are usually positively correlated.

Random forests can be viewed as a generalization of the basic bagging method, as applied to decision trees. Random forests are defined as an ensemble of decision trees, in which randomness has explicitly been inserted into the model building process of each decision tree. While the bootstrapped sampling approach of bagging is also an indirect way of adding randomness to model-building, there are some disadvantages of doing so. The main drawback of using decision-trees directly with bagging is that the split choices at the top levels of the tree are statistically likely to remain approximately invariant to bootstrapped sampling. Therefore, the trees are more correlated, which limits the amount of error reduction obtained from bagging. In such cases, it makes sense to directly increase the diversity of the component decision-tree models. The idea is to use a randomized decision tree model with less correlation between the different ensemble components. The underlying variability can then be more effectively reduced by an averaging approach. The final results are often more accurate than a direct application of bagging on decision trees. Figure 11.6b provides a typical example of the effect of the approach on the decision boundary, which is similar to that of bagging, but it is usually more pronounced.

The *random-split selection* approach directly introduces randomness into the split criterion. An integer parameter $q \leq d$ is used to regulate the amount of randomness introduced in split selection. The split selection at each node is preceded by the randomized selection of a subset S of attributes of size q. The splits at that node are then executed using only this subset. Larger values of q will result in correlated trees that are similar to a tree without any injected randomness. By selecting small values of q relative to the full dimensionality

d, the correlations across different components of the random forest can be reduced. The resulting trees can also be grown more efficiently, because a smaller number of attributes need to be considered at each node. On the other hand, when a larger size q of the selected feature set S is used, then each individual ensemble component will have better accuracy, which is also desirable. Therefore, the goal is to select a good trade-off point. It has been shown that, when the number of input attributes is d, a total of $q = \log_2(d) + 1$ attributes should be selected to achieve the best trade-off. Interestingly, even a choice of $q = 1$ seems to work well in terms of accuracy, though it requires the use of a larger number of ensemble components. After the ensemble has been constructed, the dominant label from the various predictions of a test instance is reported. This approach is referred to as *Forest-RI* because it is based on *random input selection*.

This approach does not work well when the overall dimensionality d is small, and therefore it is no longer possible to use values of q much smaller than d. In such cases, a value $L \leq d$ is specified, which corresponds to the number of input features that are *combined* together. At each node, L features are randomly selected, and combined linearly with coefficients generated uniformly at random from the range $[-1, 1]$. A total of q such combinations are generated in order to create a new subset S of multivariate attributes. As in the previous case, the splits are executed using only the attribute set S. This results in multivariate random splits. This approach is referred to as *Forest-RC* because it uses *random linear combinations*.

In the random forest approach, deep trees are grown without pruning. Each tree is grown on a bootstrapped sample of the training data to increase the variance reduction. As in bagging, the variance is reduced significantly by the random forest approach. However, the component classifiers may have higher bias because of the restricted split selection of each component classifier. This can sometimes cause a problem when the fraction of informative features in the training data is small. Therefore, the gains in random forests are a result of variance reduction. In practice, the random forests approach usually performs better than bagging and comparable to boosting. It is also resistant to noise and outliers.

11.8.3.3 Boosting

In boosting, a weight is associated with each training instance, and the different classifiers are trained with the use of these weights. The weights are modified iteratively based on classifier performance. In other words, the future models constructed are dependent on the results from previous models. Thus, each classifier in this model is constructed using a the same algorithm $\mathcal{A}$ on a weighted training data set. The basic idea is to focus on the incorrectly classified instances in future iterations by increasing the relative weight of these instances. The hypothesis is that the errors in these misclassified instances are caused by classifier bias. Therefore, increasing the instance weight of misclassified instances will result in a new classifier that corrects for the bias on these *particular* instances. By iteratively using this approach and creating a weighted combination of the various classifiers, it is possible to create a classifier with lower *overall* bias. For example, in Fig. 11.6a, each individual SVM is not globally optimized and is accurate only near specific portions of the decision boundary, but the combination ensemble provides a very accurate decision boundary.

The most well-known approach to boosting is the *AdaBoost* algorithm. For simplicity, the following discussion will assume the binary class scenario. It is assumed that the class labels are drawn from $\{-1, +1\}$. This algorithm works by associating each training example with a *weight* that is updated in each iteration, depending on the results of the classification in the last iteration. The base classifiers therefore need to be able to work with weighted

Algorithm *AdaBoost*(Data Set: $\mathcal{D}$, Base Classifier: $\mathcal{A}$, Maximum Rounds: T)
begin
  $t = 0$;
  for each i initialize $W_1(i) = 1/n$;
  repeat
    $t = t + 1$;
    Determine weighted error rate ϵ_t on $\mathcal{D}$ when base algorithm $\mathcal{A}$
      is applied to weighted data set with weights $W_t(\cdot)$;
    $\alpha_t = \frac{1}{2}\log_e((1 - \epsilon_t)/\epsilon_t)$;
    for each misclassified $\overline{X_i} \in \mathcal{D}$ **do** $W_{t+1}(i) = W_t(i)e^{\alpha_t}$;
      else (correctly classified instance) **do** $W_{t+1}(i) = W_t(i)e^{-\alpha_t}$;
    for each instance $\overline{X_i}$ **do** normalize $W_{t+1}(i) = W_{t+1}(i)/[\sum_{j=1}^{n} W_{t+1}(j)]$;
  until $((t \geq T)$ OR $(\epsilon_t = 0)$ OR $(\epsilon_t \geq 0.5))$;
  Use ensemble components with weights α_t for test instance classification;
end

Figure 11.7: The *AdaBoost* algorithm

instances. Weights can be incorporated either by direct modification of training models, or by (biased) bootstrap sampling of the training data. The reader should revisit the section on rare class learning for a discussion on this topic. Instances that are misclassified are given higher weights in successive iterations. Note that this corresponds to intentionally biasing the classifier in later iterations with respect to the *global* training data, but reducing the bias in certain *local* regions that are deemed "difficult" to classify by the specific model $\mathcal{A}$.

In the tth round, the weight of the ith instance is $W_t(i)$. The algorithm starts with equal weight of $1/n$ for each of the n instances, and updates them in each iteration. In the event that the ith instance is misclassified, then its (relative) weight is increased to $W_{t+1}(i) = W_t(i)e^{\alpha_t}$, whereas in the case of a correct classification, the weight is decreased to $W_{t+1}(i) = W_t(i)e^{-\alpha_t}$. Here α_t is chosen as the function $\frac{1}{2}\log_e((1-\epsilon_t)/\epsilon_t)$, where ϵ_t is the fraction of incorrectly predicted training instances (computed after weighting with $W_t(i)$) by the model in the tth iteration. The approach terminates when the classifier achieves 100 % accuracy on the training data ($\epsilon_t = 0$), or it performs worse than a random (binary) classifier ($\epsilon_t \geq 0.5$). An additional termination criterion is that the number of boosting rounds is bounded above by a user-defined parameter T. The overall training portion of the algorithm is illustrated in Fig. 11.7.

It remains to be explained how a particular test instance is classified with the ensemble learner. Each of the models induced in the different rounds of boosting is applied to the test instance. The prediction $p_t \in \{-1, +1\}$ of the test instance for the tth round is weighted with α_t and these weighted predictions are aggregated. The sign of this aggregation $\sum_t p_t \alpha_t$ provides the class label prediction of the test instance. Note that less accurate components are weighted less by this approach.

An error rate of $\epsilon_t \geq 0.5$ is as bad or worse than the expected error rate of a random (binary) classifier. This is the reason that this case is also used as a termination criterion. In some implementations of boosting, the weights $W_t(i)$ are reset to $1/n$ whenever $\epsilon_t \geq 0.5$, and the boosting process is continued with the reset weights. In other implementations, ϵ_t is allowed to increase beyond 0.5, and therefore some of the prediction results p_t for a test instance are effectively inverted with negative values of the weight $\alpha_t = \log_e((1 - \epsilon_t)/\epsilon_t)$.

Boosting primarily focuses on reducing the bias. The bias component of the error is reduced because of the greater focus on misclassified instances. The combination decision boundary is a complex combination of the simpler decision boundaries, which are each optimized to specific parts of the training data. An example of how simpler decision boundaries combine to provide a complex decision boundary was already illustrated in Fig. 11.6a. Because of its focus on the bias of classifier models, such an approach is capable of combining many weak (high bias) learners to create a strong learner. Therefore, the approach should generally be used with simpler (high bias) learners with low variance in the individual ensemble components. In spite of its focus on bias, boosting can also reduce the variance slightly when reweighting is implemented with sampling. This reduction is because of the repeated construction of models on randomly sampled, albeit reweighted, instances. The amount of variance reduction depends on the reweighting scheme used. Modifying the weights less aggressively between rounds will lead to better variance reduction. For example, if the weights are not modified at all between boosting rounds, then the boosting approach defaults to bagging, which only reduces variance. Therefore, it is possible to leverage variants of boosting to explore the bias-variance trade-off in various ways.

Boosting is vulnerable to data sets with significant noise in them. This is because boosting assumes that misclassification is caused by the bias component of instances near the *incorrectly modeled decision boundary*, whereas it might simply be a result of the mislabeling of the *data*. This is the noise component that is intrinsic to the *data*, rather than the *model*. In such cases, boosting inappropriately overtrains the classifier to low-quality portions of the data. Indeed, there are many noisy real-world data sets where boosting does not perform well. Its accuracy is typically superior to bagging in scenarios where the data sets are not excessively noisy.

11.8.3.4 Bucket of Models

The bucket of models is based on the intuition that it is often difficult to know *a priori*, which classifier will work well for a particular data set. For example, a particular data set may be suited to decision trees, whereas another data set may be well suited to support vector machines. Therefore, *model selection* methods are required. Given a data set, how does one determine, which algorithm to use? The idea is to first divide the data set into two subsets A and B. Each algorithm is trained on subset A. The set B is then used to evaluate the performance of each of these models. The winner in this "bake-off" contest is selected. Then, the winner is retrained using the full data set. If desired, cross-validation can be used for the contest, instead of the "hold-out" approach of dividing the training data into two subsets.

Note that the accuracy of the bucket of models is no better than the accuracy of the best classifier for a *particular* data set. However, over *many* data sets, the approach has the advantage of being able to use the best model that is suited to *each* data set, because different classifiers may work differently on different data sets. The bucket of models is used commonly for model selection and parameter tuning in classification algorithms. Each individual model is the same classifier over a different choice of the parameters. The winner therefore provides the optimal parameter choice across all models.

The bucket of models approach is based on the idea that different classifiers will have different kinds of bias on different data sets. This is because the "correct" decision boundary varies with the data set. By using a "winner-takes-all" contest, the classifier with the most accurate decision boundary will be selected for each data set. Because the bucket of models evaluates the classifier accuracy based on *overall* accuracy, it will also tend to select a model with lower variance. Therefore, the approach can reduce both the bias and the variance.

11.8.3.5 Stacking

The stacking approach is a very general one, in which two levels of classification are used. As in the case of the bucket of models approach, the training data is divided into two subsets A and B. The subset A is used for the first-level classifiers that are the ensemble components. The subset B is used for the second-level classifier that combines the results from different ensemble components in the previous phase. These two steps are described as follows:

1. Train a set of k classifiers (ensemble components) on the training data subset A. These k ensemble components can be generated in various ways, such as drawing k bootstrapped samples (bagging) on data subset A, k rounds of boosting on data subset A, k different random decision trees on data subset A, or simply training k heterogeneous classifiers on data subset A.

2. Determine the k outputs of each of the classifiers on the training data subset B. Create a new set of k features, in which each feature value is the output of one of these k classifiers. Thus, each point in training data subset B is transformed to this k-dimensional space based on the prediction of the k first-level classifiers. Its class label is its (known) ground-truth value. The second-level classifier is trained on this new representation of subset B.

The result is a set of k first-level models used to transform the feature space, and a combiner classifier at the second-level. For a test instance, the first-level models are used to create a new k-dimensional representation. The second-level classifier is then used to predict the test instance. In many implementations of stacking, the original features of data subset B are retained along with the k new features for learning the second-level classifier. It is also possible to use class probabilities as features rather than the class label predictions. To prevent loss of training data in the first-level and second-level models, this method can be combined with m-way cross-validation. In this approach, a new feature set is derived for *each* training data point by iteratively using $(m-1)$ segments for training the first-level classifier, and using it to derive the features of the remainder. The second-level classifier is trained on the newly created data set, which represents *all* the training data points. Furthermore, the first-level classifiers are re-trained on the full training data in order to enable more robust feature transformations of test instances during classification.

The stacking approach is able to reduce both bias and variance, because its combiner learns from the errors of different ensemble components. Many other ensemble methods can be viewed as special cases of stacking in which a data-independent model combination algorithm, such as a majority vote, is used. The main advantage of stacking is the flexible learning approach of its combiner, which makes it potentially more powerful than other ensemble methods.

11.9 Summary

In this chapter, we studied several advanced topics in data classification, such as multiclass learning, scalable learning, and rare class learning. These are more challenging scenarios for data classification that require dedicated methods. Classification can often be enhanced with additional unlabeled data in semisupervised learning, or by selective acquisition of the user, as in active learning. Ensemble methods can also be used to significantly improve classification accuracy.

In multiclass learning methods, binary classifiers are combined in a meta-algorithm framework. Typically, either a one-against-rest, or a one-against-one approach is used. The voting from different classifiers is used to provide a final result. In many scenarios, the one-against-one approach is more efficient than the one-against-rest approach. Many scalable methods have been designed for data classification. For decision trees, two scalable methods include *RainForest* and *BOAT*. Numerous fast variations of SVM classifiers have also been designed.

The rare class learning problem is very common, especially because class distributions are very imbalanced in many real scenarios. Typically, the objective function for the classification problem is changed with the use of cost-weighting. Cost-weighting can be achieved either with example weighting, or with example resampling. Typically, the normal class is undersampled in example resampling, which results in better training efficiency.

The paucity of training data is common in real domains. Semisupervised learning is one way of addressing the paucity of training data. In these methods, the copiously available unlabeled data is used to make estimations about class distributions in regions where little labeled data is available. A second way of leveraging the copious unlabeled data, is by actively assigning labels so that the most informative labels are determined for classification.

Ensemble methods improve classifier accuracy by reducing their bias and variance. Some ensemble methods, such as bagging and random forests, are designed only to reduce the variance, whereas other ensemble methods, such as boosting and stacking, can help reduce both the bias and variance. In some cases, such as boosting, it is possible for the ensemble to overfit the noise in the data and thereby lead to lower accuracy.

11.10 Bibliographic Notes

Multiclass strategies are used for those classifiers that are designed to be binary. A typical example of such a classifier is the support vector machine. The one-against-rest strategy is introduced in [106]. The one-against-one strategy is discussed in [318].

Some examples of scalable decision tree methods include *SLIQ* [381], *BOAT* [227], and *RainForest* [228]. Some early parallel implementations of decision trees include the *SPRINT* method [458]. An example of a scalable SVM method is *SVMLight* [291]. Other methods, such as *SVMPerf* [292], reformulate the SVM optimization to reduce the number of slack variables, and increase the number of constraints. A cutting plane approach that works with a small subset of constraints at a time is used to make the SVM classifier scalable. This approach is discussed in detail in Chap. 13 on text mining.

Detailed discussions on imbalance and cost-sensitive learning may be found in [136, 139, 193]. A variety of general methods have been proposed for cost-sensitive learning, such as *MetaCost* [174], weighting [531], and sampling [136, 531]. The *SMOTE* method is discussed in [137]. Boosting algorithms have also been studied for the problem of cost-sensitive learning. The *AdaCost* algorithm was proposed in [203]. Boosting techniques can also be combined with sampling methods, as in the case of the *SMOTEBoost* algorithm [138]. An evaluation of boosting algorithms for rare class detection is provided in [296]. Discussions of linear regression models and regression trees may be found in [110, 256, 391].

Recently, the semisupervised and active learning problems have been studied to use external information for better supervision. The co-training method was discussed in [100]. The EM algorithm for combining labeled and unlabeled data was proposed in [410]. Transductive SVM methods are proposed in [293, 496]. The method in [293] is a scalable SVM method that uses an iterative approach. Graph-based methods for semisupervised learn-

ing are discussed in [101, 294]. Surveys on semisupervised classification may be found in [33, 555].

A detailed survey on active learning may be found in [13, 454]. Methods for uncertainty sampling [345], query-by-committee [457], greatest model change [157], greatest error reduction [158], and greatest variance reduction [158] have been proposed. Representativeness-based models have been discussed in [455]. Another form of active learning queries the data *vertically*. In other words, instead of examples, it is learned which *attributes* to collect to minimize the error at a given cost level [382].

The problem of meta-algorithm analysis has recently become very important because of its significant impact on improving the accuracy of classification algorithms. The bagging and random forests methods were proposed in [111, 112]. The boosting method was proposed in [215]. Bayesian model averaging and combination methods are proposed in [175]. The stacking method is discussed in [491, 513], and the bucket-of-models approach is explained in [541].

11.11 Exercises

1. Suppose that a classification training algorithm requires $O(n^r)$ time for training on a data set of size n. Here r is assumed to be larger than 1. Consider a data set $\mathcal{D}$ with an exactly even distribution across k different classes. Compare the running time of the one-against-rest approach with that of the one-against-one approach.

2. Discuss some general meta-strategies for speeding up classifiers. Discuss some strategies that you might use to scale up (a) nearest-neighbor classifiers, and (b) associative classifiers.

3. Describe the changes required to the dual formulation of the soft SVM classifier with hinge loss to make it a weighted classifier for rare-class learning.

4. Describe the changes required to the primal formulation of the soft SVM classifier with hinge loss to make it a weighted classifier for rare-class learning.

5. Implement the one-against-rest and one-against-one multiclass approach. Use the nearest-neighbor algorithm as the base classifier.

6. Design a semisupervised classification algorithm with the use of a supervised modification of the k-means algorithm. How is this algorithm related to the EM-based semisupervised approach?

7. Suppose that your data was distributed into two thin and separated concentric rings, each of which belonged to one of the two classes. Suppose that you had only a small number of labeled examples from each of the rings but you had a lot of unlabeled examples. Would your rather use the (a) EM-algorithm, (b) transductive SVM algorithm, or the (c) graph-based approach for semisupervised classification? Why?

8. Write the optimization formulation for least-squares regression of the form $y = \overline{W} \cdot \overline{X} + b$ with a bias term b. Provide a closed-form solution for the optimal values of $\overline{W}$ and b in terms of the data matrix D and response variable vector $\overline{y}$. Show that the optimal value of the bias term b always evaluates to 0 when the data matrix D and response variable vector $\overline{y}$ are both mean-centered.

9. Design a modification of the uncertainty sampling approach in which the dollar-costs of querying various instances are known to be different. Assume that the cost of querying instance i is known to be c_i.

10. Consider a situation where a classifier gives very consistent class-label predictions when trained on samples of the (training) data. Which ensemble method should you not use? Why?

11. Design a heuristic variant of the *AdaBoost* algorithm, which will perform better than *AdaBoost* in terms of reducing the variance component of the error. Does this mean that the overall error of this ensemble variant will be lower than that of *AdaBoost*?

12. Would you rather use a linear SVM to create the ensemble component in bagging or a kernel SVM? What would you do in the case of boosting?

13. Consider a d-dimensional data set. Suppose that you used the 1-nearest neighbor class label in a randomly chosen subspace with dimensionality $d/2$ as a classification model. This classifier is repeatedly used on a test instance to create a majority-vote prediction. Discuss the bias-variance mechanism with which such a classifier will reduce error.

14. For any $d \times n$ matrix A and scalar λ, use its singular value decomposition to show that the following is always true:

$$(AA^T + \lambda I_d)^{-1} A = A(A^T A + \lambda I_n)^{-1}.$$

Here, I_d and I_n are $d \times d$ and $n \times n$ identity matrices, respectively.

15. Let the singular value decomposition of an $n \times d$ matrix D be $Q\Sigma P^T$. According to Chap. 2, its pseudoinverse is $P\Sigma^+ Q^T$. Here, Σ^+ is obtained by inverting the nonzero diagonal entries of the $n \times d$ matrix Σ and then transposing the resulting matrix.

(a) Use this result to show that:

$$D^+ = (D^T D)^+ D^T.$$

(b) Show that an alternative way of computing the pseudoinverse is as follows:

$$D^+ = D^T (DD^T)^+.$$

(c) Discuss the efficiency of various methods of computing the pseudoinverse of D with varying values of n and d.

(d) Discuss the usefulness of any of the aforementioned methods for computing the pseudoinverse in the context of incorporating the kernel trick in linear regression.