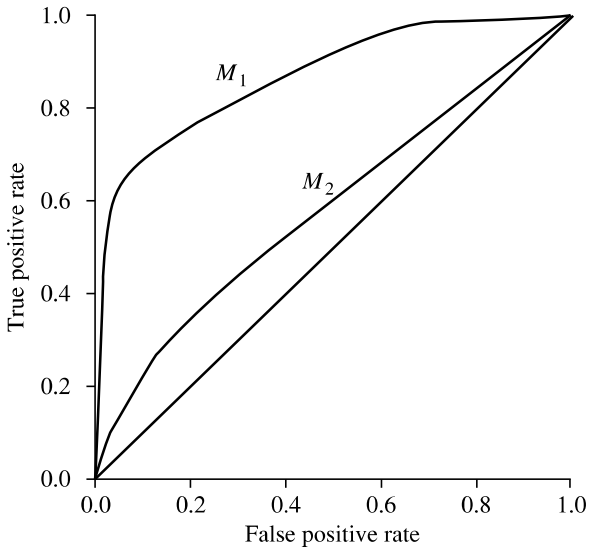

Figure 8.20 ROC curves of two classification models, M_1 and M_2. The diagonal shows where, for every true positive, we are equally likely to encounter a false positive. The closer an ROC curve is to the diagonal line, the less accurate the model is. Thus, $M1$ is more accurate here.

the curve moves steeply up from zero. Later, as we start to encounter fewer and fewer true positives, and more and more false positives, the curve eases off and becomes more horizontal.

To assess the accuracy of a model, we can measure the area under the curve. Several software packages are able to perform such calculation. The closer the area is to 0.5, the less accurate the corresponding model is. A model with perfect accuracy will have an area of 1.0.

8.6 Techniques to Improve Classification Accuracy

In this section, you will learn some tricks for increasing classification accuracy. We focus on *ensemble methods*. An ensemble for classification is a composite model, made up of a combination of classifiers. The individual classifiers vote, and a class label prediction is returned by the ensemble based on the collection of votes. Ensembles tend to be more accurate than their component classifiers. We start off in Section 8.6.1 by introducing ensemble methods in general. Bagging (Section 8.6.2), boosting (Section 8.6.3), and random forests (Section 8.6.4) are popular ensemble methods.

Traditional learning models assume that the data classes are well distributed. In many real-world data domains, however, the data are class-imbalanced, where the main class of interest is represented by only a few tuples. This is known as the *class*

imbalance problem. We also study techniques for improving the classification accuracy of class-imbalanced data. These are presented in Section 8.6.5.

8.6.1 Introducing Ensemble Methods

Bagging, *boosting*, and *random forests* are examples of **ensemble methods** (Figure 8.21). An ensemble combines a series of k learned models (or *base classifiers*), $M_1, M_2, \ldots, M_k$, with the aim of creating an improved composite classification model, $M*$. A given data set, D, is used to create k training sets, $D_1, D_2, \ldots, D_k$, where D_i $(1 \leq i \leq k-1)$ is used to generate classifier M_i. Given a new data tuple to classify, the base classifiers each vote by returning a class prediction. The ensemble returns a class prediction based on the votes of the base classifiers.

An ensemble tends to be more accurate than its base classifiers. For example, consider an ensemble that performs majority voting. That is, given a tuple $\boldsymbol{X}$ to classify, it collects the class label predictions returned from the base classifiers and outputs the class in majority. The base classifiers may make mistakes, but the ensemble will misclassify $\boldsymbol{X}$ only if over half of the base classifiers are in error. Ensembles yield better results when there is significant diversity among the models. That is, ideally, there is little correlation among classifiers. The classifiers should also perform better than random guessing. Each base classifier can be allocated to a different CPU and so ensemble methods are parallelizable.

To help illustrate the power of an ensemble, consider a simple two-class problem described by two attributes, x_1 and x_2. The problem has a linear decision boundary. Figure 8.22(a) shows the decision boundary of a decision tree classifier on the problem. Figure 8.22(b) shows the decision boundary of an ensemble of decision tree classifiers on the same problem. Although the ensemble's decision boundary is still piecewise constant, it has a finer resolution and is better than that of a single tree.

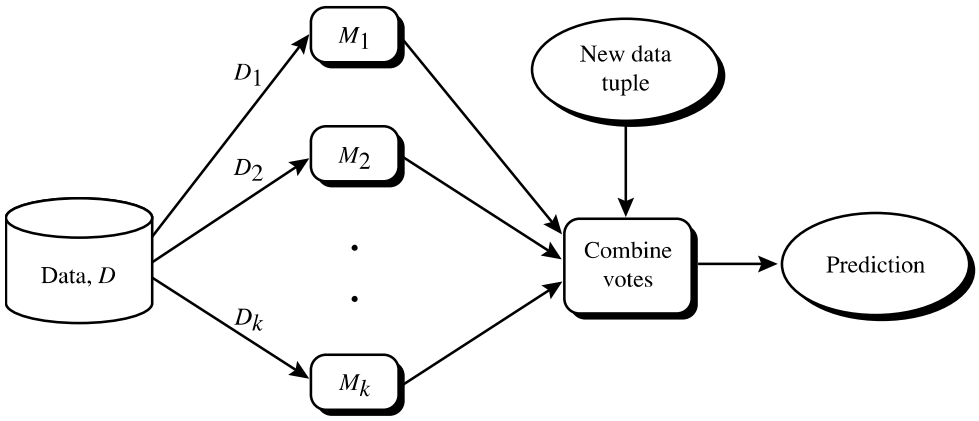

Figure 8.21 Increasing classifier accuracy: Ensemble methods generate a set of classification models, $M_1, M_2, \ldots, M_k$. Given a new data tuple to classify, each classifier "votes" for the class label of that tuple. The ensemble combines the votes to return a class prediction.

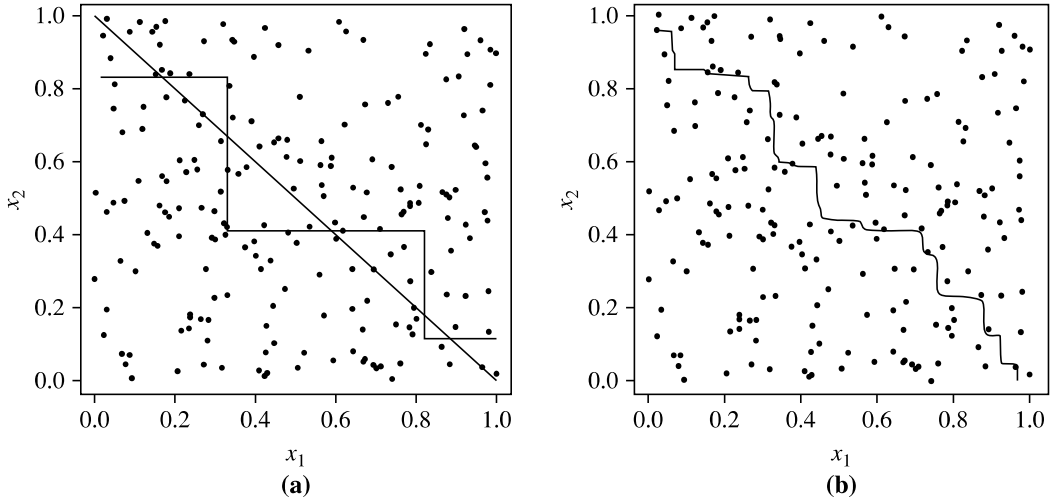

Figure 8.22 Decision boundary by (a) a single decision tree and (b) an ensemble of decision trees for a linearly separable problem (i.e., where the actual decision boundary is a straight line). The decision tree struggles with approximating a linear boundary. The decision boundary of the ensemble is closer to the true boundary. *Source:* From Seni and Elder [SE10]. © 2010 Morgan & Claypool Publishers; used with permission.

http://stats.stackexchange.com/questions/1889/1/bagging-boosting-and-stacking-in-machine-learning

ز لینک بالا یک توضیح خیلی [illegible] بصورت جمع بندی از سه الگوریتم bagging,bootstrap,stacking کرده

8.6.2 Bagging

نکته اصلی در bagging این هس که درواقع ما از طریق عمل bootstap با جایگزاری داده نمونه تولید میکنیم سپس مدل را با آن آموزش میدهیم

بعدش از بین همه این مدل ها اونیکه اکثرا به اون تخمین رسیدند جواب ما هست

Bagging = use Doctor to resolve your patient

We now take an intuitive look at how bagging works as a method of increasing accuracy. Suppose that you are a patient and would like to have a diagnosis made based on your symptoms. Instead of asking one doctor, you may choose to ask several. If a certain diagnosis occurs more than any other, you may choose this as the final or best diagnosis. That is, the final diagnosis is made based on a majority vote, where each doctor gets an equal vote. Now replace each doctor by a classifier, and you have the basic idea behind bagging. Intuitively, a majority vote made by a large group of doctors may be more reliable than a majority vote made by a small group.

Given a set, D, of d tuples, **bagging** works as follows. For iteration i ($i = 1, 2, \ldots, k$), a training set, D_i, of d tuples is sampled with replacement from the original set of tuples, D. Note that the term *bagging* stands for *bootstrap aggregation*. Each training set is a bootstrap sample, as described in Section 8.5.4. Because sampling with replacement is used, some of the original tuples of D may not be included in D_i, whereas others may occur more than once. A classifier model, M_i, is learned for each training set, D_i. To classify an unknown tuple, $\boldsymbol{X}$, each classifier, M_i, returns its class prediction, which counts as one vote. The bagged classifier, $M*$, counts the votes and assigns the class with the most votes to $\boldsymbol{X}$. Bagging can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple. The algorithm is summarized in Figure 8.23.

Training set = Bootstrap sample

The bagged classifier often has significantly greater accuracy than a single classifier derived from D, the original training data. It will not be considerably worse and is more

Algorithm: Bagging. The bagging algorithm—create an ensemble of classification models for a learning scheme where each model gives an equally weighted prediction.

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme (decision tree algorithm, naïve Bayesian, etc.).

Output: The ensemble—a composite model, $M*$.

Method:

(1) **for** $i = 1$ to k **do** // create k models:
(2) create bootstrap sample, D_i, by sampling D with replacement;
(3) use D_i and the learning scheme to derive a model, M_i;
(4) **endfor**

To use the ensemble to classify a tuple, X:

let each of the k models classify $\boldsymbol{X}$ and return the majority vote;

Figure 8.23 Bagging.

robust to the effects of noisy data and overfitting. The increased accuracy occurs because the composite model reduces the variance of the individual classifiers.

8.6.3 Boosting and AdaBoost

We now look at the ensemble method of boosting. As in the previous section, suppose that as a patient, you have certain symptoms. Instead of consulting one doctor, you choose to consult several. Suppose you assign weights to the value or worth of each doctor's diagnosis, based on the accuracies of previous diagnoses they have made. The final diagnosis is then a combination of the weighted diagnoses. This is the essence behind boosting.

در بوستینگ
وزن به هرکدام از دادههای آموزشی منتسب میشه
توجه به نکته هایلایت شده

In **boosting**, weights are also assigned to each training tuple. A series of k classifiers is iteratively learned. After a classifier, M_i, is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to "pay more attention" to the training tuples that were misclassified by M_i. The final boosted classifier, $M*$, combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy.

در آدابوست
به همه داده های آموزشی وزن یکسان 1/d را میدهد.
نیاز به k مرحله برای تولید نهایی مقدار k کلاسیفایر دارد.

AdaBoost (short for Adaptive Boosting) is a popular boosting algorithm. Suppose we want to boost the accuracy of a learning method. We are given D, a data set of d class-labeled tuples, $(\boldsymbol{X_1}, y_1), (\boldsymbol{X_2}, y_2), \ldots, (\boldsymbol{X_d}, y_d)$, where y_i is the class label of tuple $\boldsymbol{X_i}$. Initially, AdaBoost assigns each training tuple an equal weight of $1/d$. Generating k classifiers for the ensemble requires k rounds through the rest of the algorithm. In round i, the tuples from D are sampled to form a training set, D_i, of size d. Sampling

در مرحله i ام
تاپلها از D ب فرم مجموعه داده آموزشی با سایز d نمونه میشود
نمونه برداری با جایگزینی هست

شانس هر تاپل برای انتخاب به وزنش ربط داره

with replacement is used—the same tuple may be selected more than once. Each tuple's chance of being selected is based on its weight. A classifier model, M_i, is derived from the training tuples of D_i. Its error is then calculated using D_i as a test set. The weights of the training tuples are then adjusted according to how they were classified.

اگر تاپل اشتباه کلاسیفای بشه وزنش زیاد میشه

وزن زیاد = اکثر اوقات اشتباه کلاسیفای شده misclassify

If a tuple was incorrectly classified, its weight is increased. If a tuple was correctly classified, its weight is decreased. A tuple's weight reflects how difficult it is to classify—the higher the weight, the more often it has been misclassified. These weights will be used to generate the training samples for the classifier of the next round. The basic idea is that when we build a classifier, we want it to focus more on the misclassified tuples of the previous round. Some classifiers may be better at classifying some "difficult" tuples than others. In this way, we build a series of classifiers that complement each other. The algorithm is summarized in Figure 8.24.

Now, let's look at some of the math that's involved in the algorithm. To compute the error rate of model M_i, we sum the weights of each of the tuples in D_i that M_i misclassified. That is,

misclassification error : 1=> missclassified 0=> o.w

نرخ خطا برای مدل Mi

$$error(M_i) = \sum_{j=1}^{d} w_j \times err(\mathbf{X_j}), \tag{8.34}$$

where $err(\mathbf{X_j})$ is the misclassification error of tuple $\mathbf{X_j}$: If the tuple was misclassified, then $err(\mathbf{X_j})$ is 1; otherwise, it is 0. If the performance of classifier M_i is so poor that its error exceeds 0.5, then we abandon it. Instead, we try again by generating a new D_i training set, from which we derive a new M_i.

نرخ خطای Mi نشان دهنده این هست که ... **نحوه چگونگی آپدیت تاپلهای آموزشی**

The error rate of M_i affects how the weights of the training tuples are updated. If a tuple in round i was correctly classified, its weight is multiplied by $error(M_i)/(1 - error(M_i))$. Once the weights of all the correctly classified tuples are updated, the weights for all tuples (including the misclassified ones) are normalized so that their sum remains the same as it was before. To normalize a weight, we multiply it by the sum of the old weights, divided by the sum of the new weights. As a result, the weights of misclassified tuples are increased and the weights of correctly classified tuples are decreased, as described before.

نحوه آپدیت

wi=wi* e(Mi)/(1-error(Mi)

wi=wi*(sum(old w)/sum(new))

چگونه تاپل x را کلاسیفای کنیم ؟؟؟؟

برخلاف bagging که به هر کلاسیفایر میزان vote یکسان میداد

آدابوست => هرچه نرخ خطای کلاسیفایر پایین تر باشد امتیاز صحت بیشتری قائل است

"*Once boosting is complete, how is the ensemble of classifiers used to predict the class label of a tuple, X?*" Unlike bagging, where each classifier was assigned an equal vote, boosting assigns a weight to each classifier's vote, based on how well the classifier performed. The lower a classifier's error rate, the more accurate it is, and therefore, the higher its weight for voting should be. The weight of classifier M_i's vote is

$$log \frac{1 - error(M_i)}{error(M_i)}. \tag{8.35}$$

کلاسی ک بیشترین تکرر را داشت جوابه

For each class, c, we sum the weights of each classifier that assigned class c to $\mathbf{X}$. The class with the highest sum is the "winner" and is returned as the class prediction for tuple $\mathbf{X}$.

"*How does boosting compare with bagging?*" Because of the way boosting focuses on the misclassified tuples, it risks overfitting the resulting composite model to such data.

Algorithm: AdaBoost. A boosting algorithm—create an ensemble of classifiers. Each one gives a weighted vote.

Input:

- D, a set of d class-labeled training tuples;
- k, the number of rounds (one classifier is generated per round);
- a classification learning scheme.

Output: A composite model.

Method:

(1) initialize the weight of each tuple in D to $1/d$;
(2) **for** $i = 1$ to k **do** // for each round:
(3) sample D with replacement according to the tuple weights to obtain D_i;
(4) use training set D_i to derive a model, M_i;
(5) compute $error(M_i)$, the error rate of M_i (Eq. 8.34)
(6) **if** $error(M_i) > 0.5$ **then**
(7) go back to step 3 and try again;
(8) **endif**
(9) **for** each tuple in D_i that was correctly classified **do**
(10) multiply the weight of the tuple by $error(M_i)/(1 - error(M_i))$; // update weights
(11) normalize the weight of each tuple; wi=wi* e(Mi)/(1-error(Mi)
(12) **endfor**

To use the ensemble to classify tuple, X:

(1) initialize weight of each class to 0;
(2) **for** $i = 1$ to k **do** // for each classifier:
(3) $w_i = log \frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
(4) $c = M_i(\boldsymbol{X})$; // get class prediction for $\boldsymbol{X}$ from M_i
(5) add w_i to weight for class c
(6) **endfor**
(7) return the class with the largest weight;

Figure 8.24 AdaBoost, a boosting algorithm.

بوستینگ بهتر دوست داره صحت رو بالا ببره ولی خب... بعث overfitting میشه

Therefore, sometimes the resulting "boosted" model may be less accurate than a single model derived from the same data. Bagging is less susceptible to model overfitting. While both can significantly improve accuracy in comparison to a single model, boosting tends to achieve greater accuracy.

8.6.4 Random Forests

We now present another ensemble method called **random forests**. Imagine that each of the classifiers in the ensemble is a *decision tree* classifier so that the collection of classifiers