M

Machine Learning, Ensemble Methods in

SAŠO DŽEROSKI, PANČE PANOV,
BERNARD ŽENKO
Jožef Stefan Institute, Ljubljana, Slovenia

Article Outline

Glossary

Attribute (*also* feature *or* variable) An attribute is an entity that defines a property of an object (or example). It has a domain defined by its type which denotes the values that can be taken by an attribute (e. g., nominal or numeric). For example, apples can have attributes such as weight (with numeric values) and color (with nominal values such as red or green).

Example (*also* instance *or* case) An example is a single object from a problem domain of interest. In machine learning, examples are typically described by a set of attribute values and are used for learning a descriptive and/or predictive model.

Model (*also* classifier) In machine learning, a model is a computer program that attempts to simulate a particular system or its part with the aim of gaining insight into the operation of this system, or to observe its behavior. Strictly speaking, a classifier is a type of model that performs a mapping from a set of unlabeled examples to a set of (discrete) classes. However, in machine learning the term classifier is often used as a synonym for model.

Learning (*also* training) set A learning set is a set of examples that are used for learning a model or a classifier. Examples are typically described in terms of attribute values and have a corresponding output value or class.

Testing set A testing set is a set of examples that, as opposed to examples from the learning set, have not been used in the process of model learning; they are also called unseen examples. They are used for evaluating the learned model.

Ensemble An ensemble in machine learning is a set of predictive models whose predictions are combined into a single prediction. The purpose of learning ensembles is typically to achieve better predictive performance.

Definition of the Subject

Ensemble methods are machine learning methods that construct a set of predictive models and combine their outputs into a single prediction. The purpose of combining several models together is to achieve better predictive performance, and it has been shown in a number of cases that ensembles can be more accurate than single models. While some work on ensemble methods has already been done in the 1970s, it was not until the 1990s, and the introduction of methods such as *bagging* and *boosting*, that ensemble methods started to be more widely used. Today, they represent a standard machine learning method which has to be considered whenever good predictive accuracy is demanded.

Introduction

Most machine learning techniques deal with the problem of learning predictive models of data. The data are usually given as a set of examples where examples represent objects or measurements. Each example can be described in terms of values of several (independent) variables, which are also referred to as attributes, features, inputs or predictors (for example, when talking about cars, possible attributes include the manufacturer, number of seats, horsepower of a car, etc.). Associated with each example is a value of a dependent variable, also referred to as class, output or outcome. The class is some property of special interest (such as the price of the car). The typical machine learning task is to learn a model using a learning data set with the aim of predicting the value of the class for unseen examples (in our car example this would mean that we want to predict the price of a specific car based on its properties). There exist a number of methods, developed within machine learning and statistics, that solve this task more or less successfully (cf., [21,31,43]). Sometimes, however, the performance obtained by these methods (we will call them simple or base methods) is not sufficient.

One of the possibilities to improve predictive performance are ensemble methods, which in the literature are also referred to as multiple classifier systems, committees of classifiers, classifier fusion, combination or aggregation. The main idea is that, just as people often consult several sources when making an important decision, the machine learning model that takes into account several aspects of the problem (or several submodels) should be able to make better predictions. This idea goes in line with the principle of multiple explanations first proposed by the Greek philosopher Epicurus (cf., [28]), which says that for an optimal solution of a concrete problem we have to take into consideration all the hypotheses that are consistent with the input data. Indeed, it has been shown that in a number of cases ensemble methods offer better predictive performance than single models. The performance improvement comes at a price, though. When we humans want to make an informed decision we have to make an extra effort, first to find additional viewpoints on the subject, and second, to compile all this information into a meaningful final decision. The same holds true for ensemble methods; learning the entire set of models and then combining their predictions is computationally more expensive than learning just one simple model. Let us present some of the reasons why ensemble methods might still be preferred over simple methods [33].

Statistical Reasons

As already mentioned, we learn a model on the learning data, and the resulting model can have more or less good predictive performance on these learning data. However, even if this performance is good, this does not guarantee good performance also on the unseen data. Therefore, when learning single models, we can easily end up with a bad model (although there are evaluation techniques that minimize this risk). By taking into account several models and averaging their predictions we can reduce the risk of selecting a very bad model.

Very Large or Very Small Data Sets

There exist problem domains where the data sets are so large that it is not feasible to learn a model on the entire data set. An alternative and sometimes more efficient approach is to partition the data into smaller parts, learn one model for each part, and combine the outputs of these models into a single prediction.

On the other hand, there exist also many domains where the data sets are very small. As a result, the learned model can be unstable, i. e., it can drastically change if we add or remove just one or two examples. A possible remedy to this problem is to draw several overlapping subsamples from the original data, learn one model for each subsample, and then combine their outputs.

Complex Problem Domains

Sometimes, the problem domain we are modeling is just too complex to be learned by a single learning method. For illustration only, let us assume we are trying to learn a model to discriminate between examples with class '+' and examples with class '–', and the boundary between the two is a circle. If we try to solve this problem using a method that can learn only linear boundaries we will not be able to find an adequate solution. However, if we learn a set of models where each model approximates only a small part of the circular boundary, and then combine these models in an appropriate way, the problem can be solved even with a linear method.

Heterogeneous Data Sources

In some cases, we have data sets from different sources where the same type of objects are described in terms of different attributes. For example, let us assume we have a set of treated cancer patients for which we want to predict whether they will have a relapse or not. For each patient different tests can be performed, such as gene expression analyzes, blood tests, CAT scans, etc., and each of these tests results in a data set with different attributes. It is very difficult to learn a single model with all these attributes. However, we can train a separate model for each test and then combine them. In this way, we can also emphasize the importance of a given test, if we know, for example, that it is more reliable than the others.

In the remainder of the article we first describe the process of learning ensembles and then give an overview of some of the commonly used methods. We conclude with a discussion on potential impacts of ensemble methods on the development of other science areas.

Learning Ensembles

Ensembles of models are sets of (simple) models whose outputs are combined, for instance with majority voting, into a single output or prediction. The problem of learning ensembles attracts a lot of attention in the machine learning community [10], since it is often the case that predictive accuracy of ensembles is better than that of their constituent (base) models. This has also been confirmed by several empirical studies [2,11,15] for both classification (predicting a nominal variable) and regression (predicting a numeric variable) problems. In addition, several theoretical explanations have been proposed to justify the effectiveness of some commonly used ensemble methods [1,27,38].

The learning of ensembles consists of two steps. In the first step we have to learn the base models that make up the ensemble. In the second step we have to figure out how to combine these models (or their predictions) into a single coherent model (or prediction). We will now look more closely into these two steps.

Generating Base Models

When learning base models it makes sense to learn models that are diverse. Combining identical or very similar models clearly does not improve the predictive accuracy of base models. Moreover, it only increases the computational cost of the final model. By diverse models we mean models that make errors on different learning examples, so that when we combine their predictions in some smart way, the resulting prediction will be more accurate. Based on this intuition, many diversity measures have been developed with the purpose of evaluating and guiding the construction of ensembles. However, despite considerable research in this area, it is still not clear whether any of these measures can be used as a practical tool for constructing better ensembles [30]. Instead, several more or less ad hoc approaches are used for generating diverse models. We can group these approaches roughly into two groups. In the first case, the diversity of models is achieved by modifying the learning data, while in the second case, diverse models are learned by changing the learning algorithm.

The majority of ensemble research has focused on methods from the first group, i. e., methods that use different learning data sets. Such data sets can be obtained by resampling techniques such as bootstrapping [14], where learning sets are drawn randomly with replacement from the initial learning data set; this is the approach used in bagging [3] and random forests [5]. An alternative approach is used in boosting [37]. Here we start with a model that is learned on the initial data set. We identify learning examples for which this model performs well. Now we decrease the weights of these examples, since we wish for the next members of the ensemble to focus on examples misclassified by the first model. We iteratively repeat this procedure until enough base models are learned. Yet another approach to learn diverse base models is taken by the random subspaces method [22] where, instead of manipulating examples in the learning set, we each time randomly select a subset of attributes used for describing the learning set examples. These methods are typically coupled with unstable learning algorithms such as decision trees [6] or neural networks [36], for which even a small change in the learning set can produce a significantly different model.

Ensemble methods from the second group, which use different learning algorithms, use two major approaches for achieving diversity. First, if we use a base learning algorithm that depends on some parameters, diverse models can be learned by changing the values of these parameters. Again, because of their instability, decision trees and neural networks are most often employed here. A special case are randomized learning algorithms, where the outcome of learning depends on a seed used for the randomization. The second possibility is to learn each base model with a completely different learning algorithm altogether; for example, we could combine decision trees, neural networks, support vector machines and naive Bayes models into a single ensemble; this approach is used in stacking [44].

Combining Base Models

Once we have generated a sufficiently diverse set of base models, we have to combine them so that a single prediction can be obtained from the ensemble. In general, we have two options, model *selection* or model *fusion* (please note that in the literature a somewhat different definition of these two terms is sometimes used, e. g., [29]). In model selection, we evaluate the performance of all base models, and simply use predictions of the best one as predictions of the ensemble. This approach cannot be strictly regarded as an ensemble method since in the end we are using only one base model for prediction. On one hand, this can be seen as an advantage from the viewpoint that the final model is simpler, more understandable and can be executed fast. On the other hand, it is obvious that the performance of such an ensemble cannot be better than the performance of the best base model. While this seems like a serious drawback it turns out that constructing ensembles that are more accurate than a selected best base model can be a very hard task [13].

In model fusion, we really combine the predictions of all base models into a prediction of the ensemble. By far the most common method for combining predictions is voting; it is used in bagging [3], boosting [37], random forests [5] and many variations of these methods. Voting is a relatively simple combining scheme and can be applied to predictions with nominal or numeric values, or probability distributions over these. A different approach is adopted in stacking [44]. As opposed to voting, where the combining scheme is known in advance and is fixed, stacking tries to learn a so called *meta model* in order to combine base predictions as efficiently as possible. The meta model is learned on data where examples are described in terms of the predictions of the base models and the dependent variable is the final prediction of the ensemble. There are, of course, many other possibilities for combining models, including custom combining schemes specifically tailored for a given problem domain. In the next section we describe some of the most frequently used ensemble methods in more detail.

Frequently Used Ensemble Methods

The use of different schemes for base models generation and their combination, as briefly mentioned in the previous section, gives rise to a large number of possible ensemble methods. We describe here a few of them that are most common, with the exception of the best base model selection approach, which is very straightforward and does not need an additional description.

Voting

Strictly speaking, voting is not an ensemble method, but a method for combining base models, i. e., it is not concerned with the generation of the base models. Still, we include it in this selection of ensemble methods because it can be used for combining models regardless of how these models have been constructed. As mentioned before, voting combines the predictions of base models according to a static voting scheme, which does not depend on the learning data or on the base models. It corresponds to taking a linear combination of the models. The simplest type of voting is the plurality vote (also called majority vote), where each base model casts a vote for its prediction. The prediction that collects most votes is the final prediction of the ensemble. If we are predicting a numeric value, the ensemble prediction is the average of the predictions of the base models.

A more general voting scheme is weighted voting, where different base models can have different influence on the final prediction. Assuming we have some information on the quality of the base models' predictions (provided by the models themselves or through some background knowledge), we can put more weight on the predictions coming from more trustworthy models. Weighted voting predicting nominal values simply means that vote of each base model is multiplied by its weight and the value with the most weighted votes becomes the final prediction of the ensemble. For predicting numeric values we use a weighted average. If d_i and w_i are the prediction of the ith model and its weight, the final prediction is calculated as $Y = \sum_{i=1}^{b} w_i d_i$. Usually we demand that the weights are nonnegative and normalized: $w_i \geq 0\ , \forall\, i\, ; \sum_{i=1}^{b} w_i = 1$.

Another interesting aspect of voting is that, because of its simplicity, it allows for some theoretical analyzes of its efficiency. For example, when modeling a binary problem (a problem with two possible values, e. g., *positive* and *negative*) it has been shown that, if we have an ensemble with independent base models each with success probability (accuracy) greater than 1/2, i. e., better than random guessing, the accuracy of the ensemble increases as the number of base models increases (cf., [20,37,41]).

Bagging

Bagging (short for bootstrap aggregation) [3] is a voting method where base models are learned on different variants of the learning data set which are generated with bootstrapping (bootstrap sampling) [14]. Bootstrapping is a technique for sampling with replacement; from the initial learning data set we randomly select examples for a new learning (sub)set, where each example can be selected more than once. If we generate a set with the same number of examples as the original learning set, the new one will on average contain only 63.2% different examples from the original set, while the remaining 36.8% will be multiple copies. This technique is often used for estimating properties of a variable, such as its variance, by measuring those properties on the samples obtained in this manner.

Using these sampled sets, a collection of base models is learned and their predictions are combined by simple majority voting. Such an ensemble often gives better results than its individual base models because it combines the advantages of individual models. Bagging has to be used together with an unstable learning algorithm (e. g., decision trees or neural networks), where small changes in the learning set result in largely different classifiers. Another benefit of the sampling technique is that it is less likely

```
Input: Learning set $S$, Ensemble size $B$
Output: Ensemble $E$
  $E = \emptyset$
  for $i = 1$ to $B$ do
    $S^i$ = BootstrapSample($S$)
    $C^i$ = ConstructBaseModel($S^i$)
    $E = E \cup \{C^i\}$
  end for
  return $E$
```

Machine Learning, Ensemble Methods in, Algorithm 1
Learning ensembles with bagging

that (many) outliers in the learning set show up also in the bootstrap sample. As a result, base models and the ensemble as a whole should be less sensitive to data outliers. The bagging algorithm is presented in Algorithm 1. Bagging can be used both for classification and regression problems. In the case of regression the individual predictions are combined by averaging.

Boosting

Boosting [15] comprises a whole family of similar methods that, just as bagging, use voting to combine the predictions of base models learned by a single learning algorithm. The difference between the two approaches is that in bagging the complementarity of the constructed base models is left to chance, while in boosting we try to generate complementary base models by learning subsequent models, taking into account the mistakes of previous models. The procedure starts by learning the first base model on the entire learning set with equally weighted examples. For the next base models, we want them to correctly predict the examples that have not been correctly predicted by previous base models. Therefore, we increase the weights of these examples (or decrease the weights of the correctly predicted examples) and learn a new base model. We stop learning new base models when some stopping criterion is satisfied (like when the accuracy of the new base model is less then or equal to 0.5). The prediction of the ensemble is obtained by weighted voting, where more weight is given to more accurate base models; the weights of all classifiers that vote for a specific class are summed and the class with the highest total vote is predicted.

An interesting property of some boosting methods is that they provide a theoretical guarantee of the accuracy [15,26]. We can show that the predictive error of the ensemble on the learning data quickly decreases as we increase the number of base models within the ensemble. The only precondition for error decrease is that the error of the individual members of the ensemble is less than 0.5. For binary classification problems this condition is usually easy to fulfill. While the guarantee of a small error on the learning set is not a guarantee of a small error on unseen examples, boosting methods are known to frequently improve the predictive performance of the base algorithms [39]. Just as bagging, boosting should also be used together with unstable learning methods such as decision trees or neural networks. The most widely used boosting method is AdaBoost.M1 [15] presented in Algorithm 2 (together with the exact example reweighting scheme used in this algorithm), which was designed for learning with binary classification problems. Nevertheless, there exist also modifications of the original method that work on classification problems with more than two possible values (multiclass) [40] and even on regression problems [34,35]. An alternative name often used for boosting methods is *arcing* (adaptively resample and combine) [4], although strictly speaking, boosting methods are a subset of arcing methods, i. e., boosting methods are the ones for which it can be shown that they can achieve an arbitrarily small error on the learning data set.

```
Input: Learning set $S$, Ensemble size $B$
Output: Ensemble $E$
  $E = \emptyset$
  $W$ = AssignEqualWeights($S$)
  for $i = 1$ to $B$ do
    $C^i$ = ConstructModel($S$, $W$)
    $Err$ = ApplyModel($C^i$, $S$)
    if $(Err = 0) \vee (Err \geq 0.5)$ then
      TerminateModelGeneration
      return $E$
    end if
    for $j = 1$ to NumberOfExamples($S$) do
      if CorrectlyClassified($S_j$, $C^i$) then
        $W_j = W_j \frac{Err}{1-Err}$
      end if
    end for
    $W$ = NormalizeWeights($W$)
    $E = E \cup \{C^i\}$
  end for
  return $E$
```

Machine Learning, Ensemble Methods in, Algorithm 2
The AdaBoost.M1 algorithm for learning ensembles with boosting

```
Input: Learning set $S$, Ensemble size $B$,
       Proportion of attributes considered $f$
Output: Ensemble $E$
  $E = \emptyset$
  for $i = 1$ to $B$ do
    $S^i = \text{BootstrapSample}(S)$
    $C^i = \text{BuildRandomTreeModel}(S^i, f)$
    $E = E \cup \{C^i\}$
  end for
  return $E$
```

Machine Learning, Ensemble Methods in, Algorithm 3
Learning random forests

Random Forests

Random forests [5] is a method for combining models learned with a randomized version of a decision tree algorithm. Random forests can be seen as an implementation of bagging in which each model is learned with a modified version of the CART decision tree algorithm [6]; namely, when searching for an optimal attribute split in a tree, rather than considering all possible splits, only a small subset of randomly selected splits is tested (i. e., a random subset of attributes), and the best one is chosen from this subset. There are two sources of diversity when learning the trees, and both are random: the selection of a bootstrap sample for learning each tree, and the selection of attributes to on which to split at every node of the tree. Random forests are a robust and typically very accurate ensemble method applicable to classification and regression problems. The algorithm for learning random forests is presented in Algorithm 3.

Stacking

Stacking or stacked generalization [44] is a method for combining heterogeneous base models, i. e., models learned with different learning algorithms such as the nearest neighbor method, decision trees, naive Bayes, etc. Base models are not combined with a fixed scheme such as voting, but rather an additional model called *meta* (or *level 1*) model is learned and used for combining base (or *level 0*) models. The procedure has two steps. First, we generate the meta learning data set using the predictions of the base models. Second, using the meta learning set we learn the meta model which can combine predictions of base models into a final prediction.

Let $L_1, \ldots, L_N$ be the base learning algorithms, and S be the learning data set, which consists of examples $\mathbf{s}_i = (\mathbf{x}_i, y_i)$, i. e., pairs of attribute vectors $\mathbf{x}_i$ and their classifications y_i. Generation of the meta learning data set is done using a leave-one-out, or in general, a K-fold cross-validation procedure. The initial learning set S with n examples is split into K proper subsets S_k of roughly equal size and class value distribution. For each of the subsets a group of base models $C_1^k, C_2^k, \ldots, C_N^k$ is learned ($C_j^k = L_j(S - S_k), \forall j = 1, \ldots, N, \forall k = 1, \ldots, K$). These models are now used for predicting examples that were not included in their learning set: $\hat{y}_i^j = C_j^k(x_i)$, $x_i \in S_k$. These predictions are collected into a meta learning set S^m. Each example from the original learning set S has a corresponding example in S^m of the form $\mathbf{s}_i^m = (\hat{\mathbf{y}}_i, y_i) = ((\hat{y}_i^1, \ldots, \hat{y}_i^N), y_i)$. The attributes of the meta learning set are therefore the predictions of the base models ($\hat{y}_i^j$), while the class value is the true class value from the original data set (y_i). In the second step, a meta learning algorithm L^m is applied to this meta learning set. When predicting a value of an unseen example, we first collect the predictions of the base models which are then given to the meta model that combines them into a final prediction. The stacking algorithm is presented in Algorithm 4. The performance of stacking highly depends on the attributes used in the meta learning set (we have only described the simplest option above) and the meta learning algorithm used for learning the meta model (cf. [13,42]).

Random Subspace Method

The random subspace method (RSM) [22] is an ensemble method somewhat similar to bagging. However, while in bagging the diversity of base models is achieved by sampling examples from the initial learning data set, in RSM the diversity is achieved by sampling attributes from the learning set. Let each learning example X_i in the learning set S be a p-dimensional vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. RSM randomly selects p^* attributes from S, where $p^* < p$. By this, we obtain the p^* dimensional random subspace of the original p-dimensional attribute space. Therefore, the modified training set $\tilde{S} = (\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_n)$ consists of p^*-dimensional learning examples $\tilde{\mathbf{x}}_i = (x_{j1}, x_{i2}, \ldots, x_{ip^*})$ $(i = 1, 2, \ldots, n)$. Afterwards, base models are learned from the random subspaces $\tilde{S}^j$ (of the same size), $j = 1, 2, \ldots, B$, and they are combined by voting to obtain a final prediction. Typically, p^* is equal for all base models. The RSM algorithm is presented in Algorithm 5.

The RSM benefits from using random subspaces for learning base models and from their aggregation. When the number of learning examples is relatively small as com-

```
Input: Learning set $S$, Number of folds for meta data generation $K$,
       Base and meta learning algorithms $\{L_1, L_2, \ldots, L_N\}, L^m$
Output: Ensemble $E$
  $E = \emptyset$
  $\{S_1, S_2, \ldots, S_K\} = \text{SplitData}(S, K)$
  $S^m = \emptyset$
  for $k = 1$ to $K$ do
    for $j = 1$ to $N$ do
      $C_j^k = L_j(S - S_k)$
    end for
    $S_k^m = \bigcup_{x_i \in S_k} \{(C_1^k(x_i), C_2^k(x_i), \ldots, C_N^k(x_i), y_i)\}$
  end for
  $S^m = \bigcup_{k=1}^{K} S_k^m$
  $C^m = L^m(S^m)$
  $\{C_1, C_2, \ldots, C_N\} = \{L_1(S), L_2(S), \ldots, L_N(S)\}$
  $E = (\{C_1, C_2, \ldots, C_N\}, C_m)$
  return $E$
```

Machine Learning, Ensemble Methods in, Algorithm 4
Learning ensembles with stacking

```
Input: Training examples $S$, Number of subspaces $B$,
       Dimension of subspaces $p^*$
Output: Ensemble $E$
  $E = \emptyset$
  for $j = 1$ to $B$ do
    $\tilde{S^j} = \text{SelectRandomSubspace}(S, p^*)$
    $C_j = \text{ConstructModel}(\tilde{S^j})$
    $E = E \cup \{C_j\}$
  end for
  return $E$
```

Machine Learning, Ensemble Methods in, Algorithm 5
Learning ensembles with the random subspace method

pared to the dimensionality of the data, learning models in random subspaces alone may solve the small sample problem. In this case the subspace dimensionality is smaller than in the original attribute space, while the number of learning objects remains the same. When the data set has many redundant attributes, one may obtain better models in random subspaces than in the original attribute space. The combined decision of such models may be superior to a single model constructed on the original learning set in the complete attribute space.

The RSM was originally developed to be used with decision trees, but the methodology can also be used to improve the performance of other unstable learning methods (e. g., rule sets, neural networks, etc.). The RSM is expected to perform well when there is a certain redundancy in the data attribute space [22]. It has been noticed that the performance of the RSM is affected by the problem complexity (attribute efficiency, length of class boundary, etc.) [23]. When applied to decision trees, the RSM is superior to a single decision tree and may outperform both bagging and boosting [22].

Other Methods

Mixture of Experts Models The combination of the base learners can be governed by a supervisor learner, that selects the most appropriate element of the ensemble on the basis of the available input data. This idea led to the mixture of experts methods [24], where a gating network performs the division of the input space and small neural networks perform the effective calculation at each assigned region separately. An extension of this approach is the hierarchical mixture of experts method, where the outputs of the different experts are non-linearly combined by different supervisor gating networks hierarchically organized [25]. Cohen and Intrator extended the idea of constructing local simple base learners for different regions of the input space, searching for appropriate architectures that should be locally used and for a criterion to select a proper unit for each region of input space [8,9].

Error Correcting Output Codes Error-correcting output codes (ECOC) [12] is an ensemble method for improving the performance of classification algorithms in multiclass learning problems. Let us note that some machine learning algorithms (e. g., standard support vector machines) work only with two class problems. In order to apply such algorithms to a multiclass problem it has to be decomposed into several independent two-class problems; the algorithm is run on each of them and the outputs of the resulting binary models are combined. The error-correcting output codes method enables us to efficiently combine the outputs of such models.

As already mentioned, we have binary base models with possible outputs (-1 or $+1$), and there exists a code matrix W of size $K*B$ whose K rows are the binary codes of classes in terms of B base models C_j. This code matrix allows us to define a multiclass classification problem in terms of two-class classification problems. The problem here is that if there is an error with one of the base models, there will be a misclassification because the class code words are so similar. The ECOC approach sets the B beforehand and then tries to find such a code matrix W that the distances between rows, and at the same time the distances between the columns, are as large as possible in terms of the Hamming distance [19]. The ECOC can be written as a voting scheme where the entries of W, w_{ij} are considered as vote weights $y_i = \sum_{j=}^{B} w_{ij} d_j$. As a final prediction the class with the highest y_i is chosen.

Future Directions

Recent and future research directions in ensemble methods that are likely to have high impact on data mining and other areas of science focus along the following topics: Combinations of different sources of diversity; Understanding and interpretation of ensembles; Understanding and explaining in more basic terms why ensembles perform better that individual models.

Random forests [5], one of the most successful ensemble approaches, combine two sources of diversity of the base models: Variations in the learning data set (achieved through different bootstrap samples, as in bagging) and a randomized base-level learning algorithm. Another recent approach [32] combines the bagging way of sampling with the random subspaces way of randomly selecting subsets of the original set of attributes. This approach has the advantage of being applicable in conjunction with a variety of base-level learning algorithms that do not need to be randomized.

We have provided some intuition of why ensembles work better than individual models in terms of the diversity of the base models. More fundamental explanations are produced in the bias-variance analysis framework: Roughly speaking, the error of a learning algorithm can be divided into a part due to the functional form used by the algorithm (bias) and a part that is due to the instability of the algorithms (variance). Bagging and random forests reduce the variance part. Boosting reduces mainly the bias part, but also the variance part. Finally, boosting can also be viewed as an incremental forward stagewise regression procedure with regularization (Lasso penalty), which maximizes the margin between the two classes, much like the approach of support vector machines [18].

While ensembles typically perform better than a single model, they do have an important disadvantage: They are more complex and difficult (if not impossible) to interpret. Recent research has addressed this issue in several ways. Some approaches produce an estimate of the relative importance of the attributes as an explanation, for example partial dependency plots [16] and the attribute ranking approach based on bagging and random forests. The approach of Caruana [7] is to construct a single model that approximates the behavior of the ensemble: This is done by generating examples, classifying them with the ensemble, and learning a single model from the resulting learning set. Finally, a recent approach [17] builds rule ensembles, where small (and understandable) sets of rules are preferred through regularization.

Bibliography

Primary Literature

1. Allwein EL, Schapire RE, Singer Y (2000) Reducing multiclass to binary: a unifying approach for margin classifiers. J Mach Learn Res 1:113–141
2. Bauer E, Kohavi R (1999) An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Mach Learn 36(1–2):105–139
3. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
4. Breiman L (1998) Arcing classifiers. Ann Stat 26(3):801–849
5. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
6. Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth and Brooks, Monterey
7. Buciluǎ C, Caruana R, Niculescu-Mizil A (2006) Model compression. In: Proc. of the 12th ACM SIGKDD international conference on knowledge discovery and data mining (KDD '06). ACM, New York, pp 535–541
8. Cohen S, Intrator N (2000) A hybrid projection based and radial basis function architecture. In: Proc. of the 1st international workshop on multiple classifier systems (MCS '00). Springer, Berlin, pp 147–156
9. Cohen S, Intrator N (2001) Automatic model selection in a hybrid perceptron/radial network. In: Proc. of the 2nd international workshop on multiple classifier systems (MCS '01). Springer, Berlin, pp 440–454
10. Dietterich TG (1997) Machine-learning research: four current directions. AI Mag 18(4):97–136

11. Dietterich TG (2000) Ensemble methods in machine learning. In: Proc. of the 1st international workshop on multiple classifier systems (MCS '00). Springer, Berlin, pp 1–15
12. Dietterich TG, Bakiri G (1995) Solving multiclass learning problems via error-correcting output codes. J Artif Intell Res 2:263–286
13. Džeroski S, Ženko B (2004) Is combining classifiers with stacking better than selecting the best one? Mach Learn 54(3):255–273
14. Efron B (1979) Bootstrap methods: Another look at the jackknife. Ann Stat 7(1):1–26
15. Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: Saitta L (ed) Machine learning: Proc. of the 13th international conference (ICML '96). Morgan Kaufmann, San Francisco, pp 148–156
16. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29(5):11189–1232
17. Friedman JH, Popescu BE (2005) Predictive learning via rule ensembles. Technical report, Stanford University, Department of Statistics
18. Friedman JH, Hastie T, Tibshirani RJ (1998) Additive logistic regression: a statistical view of boosting. Technical report, Stanford University, Department of Statistics
19. Hamming RW (1950) Error detecting and error correcting codes. Bell Syst Tech J 26(2):147–160
20. Hansen LK, Salamon P (1990) Neural network ensembles. IEEE Trans Pattern Anal Mach Intell 12(10):993–1001
21. Hastie T, Tibshirani RJ, Friedman JH (2001) The elements of statistical learning. Springer Series in Statistics. Springer, Berlin
22. Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844
23. Ho TK (2000) Complexity of classification problems and comparative advantages of combined classifiers. In: Kittler J, Roli F (eds) Proc. of the 1st international workshop on multiple classifier systems (MCS '00), vol 1857. Springer, Berlin, pp 97–106
24. Jacobs RA (1995) Methods for combining experts' probability assessments. Neural Comput 7(5):867–888
25. Jordan MI, Jacobs RA (1992) Hierarchies of adaptive experts. In: Moody JE, Hanson S, Lippmann RP (eds) Advances in Neural Information Processing System (NIPS). Morgan Kaufmann, San Mateo, pp 985–992
26. Kearns MJ, Vazirani UV (1994) An introduction to computational learning theory. MIT Press, Cambridge
27. Kittler J, Hatef M, Duin RPW, Matas J (1998) On combining classifiers. IEEE Trans Pattern Anal Mach Intell 20(3):226–239
28. Kononenko I, Kukar M (2007) Machine learning and data mining: introduction to principles and algorithms. Horwood, Chichester
29. Kuncheva LI (2004) Combining pattern classifiers: methods and algorithms. Wiley-Interscience, Hoboken
30. Kuncheva LI, Whitaker CJ (2003) Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Mach Learn 51(2):181–207
31. Mitchell T (1997) Machine Learning. McGraw-Hill, New York
32. Panov P, Džeroski S (2007) Combining bagging and random subspaces to create better ensembles. In: Proc. of 7th international symposium on intelligent data analysis (IDA '07), vol 4723. Lecture notes in computer science. Springer, Berlin, pp 118–129
33. Polikar R (2006) Ensemble based systems in decision making. IEEE Circuits Syst Mag 6(3):21–45
34. Rätsch G, Demiriz A, Bennett KP (2002) Sparse regression ensembles in infinite and finite hypothesis spaces. Mach Learn 48(1–3):189–218
35. Ridgeway G, Madigan D, Richardson T (1999) Boosting methodology for regression problems. In: Heckerman D, Whittaker J (eds) Proc. of the 7th international workshop on artificial intelligence and statistics. Morgan Kaufmann, San Francisco, pp 152–161
36. Rosenblatt F (1962) Principles of neurodynamics: perceptron and the theory of brain mechanisms. Spartan Books, Washington
37. Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
38. Schapire RE (1999) A brief introduction to boosting. In: Proc. of the 6th international joint conference on artificial intelligence. Morgan Kaufmann, San Francisco, pp 1401–1406
39. Schapire RE (2001) The boosting approach to machine learning: an overview. In: MSRI workshop on nonlinear estimation and classification, Berkeley, CA, 2001
40. Schapire RE, Singer Y (1999) Improved boosting using confidence-rated predictions. Mach Learn 37(3):297–336
41. Schapire RE, Freund Y, Bartlett P, Lee WS (1998) Boosting the margin: a new explanation for the effectiveness of voting methods. Ann Stat 26(5):1651–1686
42. Ting KM, Witten IH (1999) Issues in stacked generalization. J Artif Intell Res 10:271–289
43. Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco
44. Wolpert DH (1992) Stacked generalization. Neural Netw 5(2):241–259

Books and Reviews

Brown G Ensemble learning bibliography. http://www.cs.man.ac.uk/~gbrown/ensemblebib/index.php. Accessed 26 March 2008

Weka 3: Data mining software in Java. http://www.cs.waikato.ac.nz/ml/weka/. Accessed 26 March 2008

Macroeconomics, Non-linear Time Series in

James Morley
Washington University, St. Louis, USA

Article Outline