ENSEMBLING

Constructing a good model from a given data set is one of the major tasks in machine learning (ML).

Strong classifiers are desirable, but are difficult to find.

Training many classifiers at the same time to solve the same problem, and then combining their output to improve accuracy, is known as an **Ensemble Method**.

When an ensemble, also known as a multi-classifier system, is based on learners of the same type, it is called a homogeneous ensemble.

When it is based on learners of different types, it is called a heterogeneous ensemble.

Usually, the ensemble’s generalization ability is better than a single classifier’s, as it can boost weak classifiers to produce better results than can a single strong classifier.

Two results published in the 1990s opened a promising new door for creating strong classifiers using ensemble methods.

The empirical study in Hansen and Salamon (1990) found that a combination of multiple classifiers produces more accurate results than the best single one, and the theoretical study in Schapire (1990) showed that weaker classifiers can be boosted to produce stronger classifiers.

There are two essential elements involved, in the design of systems that integrate multiple classifiers.

First, it is necessary to follow a plan of action to set up an ensemble of classifiers with characteristics that are sufficiently diverse.

Second, there is the need for a policy for combining the decisions, or outputs, of particular classifiers in a manner that strengthens accurate decisions and weakens erroneous classifications.

The most commonly used ensemble algorithms:

Bagging

Breiman's bootstrap aggregating method, or "bagging" for short, was one of the first ensemble-based algorithms, and it is one of the most natural and straightforward ways of achieving a high efficiency (Breiman, 1996).

In bagging, a variety of results is produced, using bootstrapped copies of the training data, i.e. numerous subsets of data are randomly drawn with replacement from the complete training data.

A distinct classifier of the same category is modeled, using a subset of the training data.

Fusing of particular classifiers is achieved by the use of a majority vote on their selections.

Thus, for any example input, the ensemble’s decision is the class selected by the greatest number of classifiers.

The following algorithm contains a pseudocode for the bagging method.

Algorithm 1 Bagging

Input: I (a classifier inducer), T (# of iterations), S (Data set for training), N (subset size).

Output: C_t; $t = 1, 2, .., T$

1: $t \leftarrow 1$
2: **repeat**
3: $S_t \leftarrow$ Subset of N instances taken, with replacement, from S.
4: Create Classifier C_t by using I on S_t.
5: t ++
6: **until** $t > T$.

Boosting

It was shown by Schapire, in 1990, that a weak learner, namely an algorithm that produces classifiers that can slightly outperform random guessing, can be transformed into a strong learner, namely an algorithm that constructs classifiers capable of correctly classifying all of the instances except for an arbitrarily small fraction (Schapire, 1990).

Boosting generates an ensemble of classifiers, as does bagging, by carrying out resampling of the data and combining decisions using a majority vote.

However, that is the extent of the similarities with bagging.

Re-sampling in boosting is carefully devised so as to supply consecutive classifiers with the most informative training data.

Essentially, **boosting generates three classifiers** as follows:

A random subset of the available training data is used for constructing the first classifier.

The most informative subset given for the first classifier is used for training the second classifier, where the most informative subset consists of training data instances, such that half of them were correctly classified by the first classifier and the other half were misclassified.

Finally, training data for the third classifier are made of instances on which the first and second classifiers were in disagreement.

A three-way majority vote is then used, to combine the decisions of the three classifiers.

In 1997, Freund and Schapire presented a generalized version of the original boosting algorithm called “adaptive boosting” or “AdaBoost” for short.

The method received that name from to its ability to adapt to errors related to weak hypotheses, which are obtained from WeakLearn (Freund).

AdaBoost.M1 and AdaBoost.R are two of the most frequently used variations of this category of algorithms, because they are suitable for dealing withmulti-class and regression problems, respectively.

AdaBoost produces a set of hypotheses, and then uses weighted majority voting of the classes determined by the particular hypotheses in order to combine decisions.

A weak classifier is trained to generate the hypotheses, by drawing instances from a successively refreshed distribution of the training data.

The updating of the distribution guarantees that it will be more likely to include in the data set for training the subsequent classifier examples that were wrongly classified by the preceding classifier.

Thus, the training data of successive classifiers tend to advance toward increasingly hard-to-classify instances.

Algorithm 2 AdaBoost

Input: I (a weak classifier inducer), T (# of iterations), S (Data set for training), N (subset size).

Output: C_t, α_t; $t = 1, 2, .., T$

$t \leftarrow 1$

$D_1(i) \leftarrow 1/m$; $i = 1, 2, .., m$

repeat

    Create Classifier C_t by using I and the distribution D_t.

    $\epsilon_t \leftarrow \sum_{i:C_t(x_i) \neq y_i} D_t(i)$

    if $\epsilon_t > 0.5$ **then**

        $T \leftarrow t - 1$

        exit Loop

    end if

    $\alpha_t \leftarrow \frac{1}{2} ln \frac{1 - \epsilon_t}{\epsilon_t}$

    $D_{t+1}(i) \leftarrow D_t(i) \cdot e^{-\alpha_t y_i C_t(x_i)}$

    Normalize D_{t+1} so that it becomes a distribution

    t ++

until $t > T$.

Stacking

Some instances are very likely to be misclassified, because it can happen that they are in the close neighborhood of the decision boundary, and, therefore, usually are placed on the wrong side of the boundary determined by the classifier.

On the other hand, there can be instances that are likely to be classified well, as a result of being on the correct side and far away from the corresponding decision boundaries.

This prompts the following question: can it be learned whether specific classifiers consistently perform correct classifications, or whether they consistently classify specific examples incorrectly?

Said another way, if there is an ensemble of classifiers working with a data set taken from an unknown-but-fixed distribution, can we define a correspondence between the decisions of those classifiers and their correct classes?

The idea behind Wolpert’s stacking generalization is that the outputs of an ensemble of classifiers serve as the inputs to another, second-level metaclassifier, which has the purpose of learning the mapping that relates the ensemble outputs with the real true classes

Early research by Dietterich (2000) and Miranda Dos Santos (2008) showed, both theoretically and empirically, that ensembles are superior to single-component classifiers, in terms of classification accuracy.

With the implementation of multiple base classifiers, the overall error rate of an ensemble can be reduced, provided that each base classifier is better than a random guess, namely that the overall accuracy of the base classifier is over 50%.

The advantages of ensemble classifiers are particularly evident in the field of intrusion detection, since there are many different types of intrusions, and different detectors are needed to detect them (Axelsson, 2000).

Moreover, if one classifier fails to detect an attack, then another classifier in the ensemble should detect it (Lee et al., 2000).

Based on an ensemble's structure, two general approaches may be distinguished:
(*i*) **Homogeneous Ensembles**, where all classifiers in the ensemble are generated with the same technique; and
(*ii*) **Heterogeneous Ensembles**, which utilize diverse base classifiers.

Ensemble techniques like bagging and boosting are often used to generate homogeneous ensembles, whereas stacking and voting can be used to produce heterogeneous ensembles.

Active research of ensemble-based systems by Chen et al. (2014) and Kumar and Kumar (2013) raises several open questions:

- How should suitable base components for an ensemble be created?
- How should it be decided upon which base classifiers one should rely?
- How should the decisions of base classifiers be combined into a final decision?

A

AdaBoost

Methods for constructing an ensemble classifier:

The basic idea is to construct multiple classifiers from the original data and then aggregate their predictions when classifying unknown examples.

(1) Manipulating the Training Set

(2) Manipulating the Input Features

(3) Manipulating Class Labels

(4) Manipulating the Learning Algorithm(s)

Algorithm 5.5 General procedure for ensemble method.

1: Let D denote the original training data, k denote the number of base classifiers, and T be the test data.
2: **for** $i = 1$ to k **do**
3: Create training set, D_i from D.
4: Build a base classifier C_i from D_i.
5: **end for**
6: **for** each test record $x \in T$ **do**
7: $C^*(x) = Vote(C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_k(\mathbf{x}))$
8: **end for**

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

A

ADABOOST EXAMPLE

	ROUND #1		ROUND #2	
CLASSIFIER	**MISCLASSIFIED POINTS**	**ERROR RATE**	**MISCLASSIFIED POINTS**	**ERROR RATE**
X < 2	B E	2/5		
X < 4	C B E	3/5		
X < 6	**C**	**1/5**		
X > 2	A C D	3/5		
X > 4	A D	2/5		
X > 6	A B D E	4/5		

WEIGHT	ROUND #1	ROUND #2	ROUND #3
W_A	1/5		
W_B	1/5		
W_C	1/5		
W_D	1/5		
W_E	1/5		

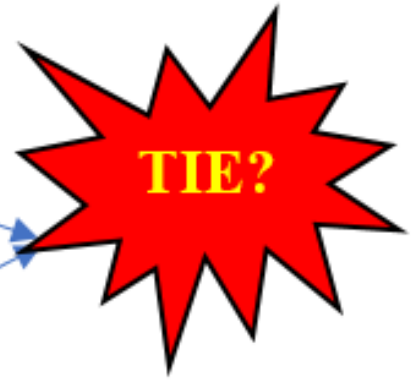
After Round#1: $h = x < 6;\ \alpha = \frac{1}{2} \ln(4);\ \epsilon = 1/5$

		Round #1	Round #2
CLASSIFIER	**MISCLASSIFIED POINTS**	**ERROR RATE**	**ERROR RATE**
X < 2	B E	2/5	2/8
X < 4	C B E	3/5	6/8
X < 6	**C**	**1/5**	4/8
X > 2	A C D	3/5	6/8
X > 4	A D	2/5	2/8
X > 6	A B D E	4/5	4/8

WEIGHT	ROUND #1	ROUND #2	ROUND #3
W_A	1/5	1/8	
W_B	1/5	1/8	
W_C	1/5	½ = 4/8	
W_D	1/5	1/8	
W_E	1/5	1/8	

In this round which one is the next weak classifier?

CLASSIFIER	MISCLASSIFIED POINTS	Round #1 ERROR RATE (ϵ)	Round #2 ERROR RATE (ϵ)
X < 2	B E	2/5	2/8
X < 4	C B E	3/5	6/8
X < 6	C	1/5	4/8
X > 2	A C D	3/5	6/8
X > 4	A D	2/5	2/8
X > 6	A B D E	4/5	4/8



WEIGHT	ROUND #1	ROUND #2	ROUND #3
W_A	1/5	1/8	
W_B	1/5	1/8	
W_C	1/5	½ = 4/8	
W_D	1/5	1/8	
W_E	1/5	1/8	

After Round#2: $h = x < 2;\ \alpha = \frac{1}{2}\ln(3);\ \epsilon = 1/4$

CLASSIFIER	MISCLASSIFIED POINTS	Round #1 ERROR RATE (ϵ)	Round #2 ERROR RATE (ϵ)	Round #3 ERROR RATE (ϵ)
X < 2	B E	2/5	2/8	½
X < 4	C B E	3/5	6/8	10/12
X < 6	**C**	**1/5**	4/8	4/12
X > 2	A C D	3/5	6/8	½
X > 4	A D	2/5	2/8	2/12
X > 6	A B D E	4/5	4/8	8/12

WEIGHT	ROUND #1	ROUND #2	ROUND #3
W_A	1/5	1/8	1/12
W_B	1/5	1/8	¼ = 3/12
W_C	1/5	½ = 4/8	4/12
W_D	1/5	1/8	1/12
W_E	1/5	1/8	¼ = 3/12

After Round#3: $h = x > 4;\ \alpha = 1/2\ln(5);\ \epsilon = 1/6$

KERNELS IN ML

Kernels are measures of similarity. Broadly speaking, machine learning algorithms which rely only on the dot product between instances can be "kernelized" by replacing all instances of $\langle x, x' \rangle$ by a kernel function $k(x, x')$.

The Basics

Let $\mathcal{X}$ denote the space of inputs and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a function which satisfies

$$k(x, x') = \langle \Phi(x), \Phi(x) \rangle$$

where Φ is a feature map which maps $\mathcal{X}$ into some dot product space $\mathcal{H}$.

kernels correspond to dot products in some dot product space.

The main advantage of using a kernel as a similarity measure are threefold:
First, if the feature space is rich enough, then simple estimators such as hyperplanes and half-spaces may be sufficient. For instance, to classify the points in Figure BUGBUG, we need a nonlinear decision boundary, but once we map the points to a 3 dimensional space a hyperplane suffices.

Second, kernels allow us to construct machine learning algorithms in the dot product space H without explicitly computing $\Phi(x)$.

Third, we need not make any assumptions about the input space X other than for it to be a set. This allows us to compute similarity between discrete objects such as strings, trees, and graphs.

A

Linear kernels are perhaps the simplest of all kernels. We assume that $x \in \mathbb{R}^n$ and define

$$k(x, x') = \langle x, x' \rangle = \sum_i x_i x'_i.$$

If x and x' are dense then computing the kernel takes $O(n)$ time.

A

for sparse vectors this can be reduced to $O(|nnz(x) \cap nnz(x')|)$, where $nnz(\cdot)$ denotes the set of non-zero indices of a vector and $|\cdot|$ denotes the size of a set.

Linear kernels are a natural representation to use for vectorial data.

They are also widely used in text mining where documents are represented by a vector containing the frequency of occurrence of words.

Instead of a simple bag of words, one can also map a text to the set of pairs of words that co-occur in a sentence for a richer representation.