Pattern Recognition - Lecture 9 Ensemble Learning

Dr. Dewan Md. Farid

Associate Professor, Department of Computer Science & Engineering
United International University, Bangladesh

April 12, 2018

Ensemble Classifier

- It is the process of combining different classification techniques to build a powerful composite model from the data.
- It returns a class label prediction for new instances based on the individual classifiers vote.
- It improves the classification accuracy of class-imbalanced data.

Ensemble Classifier (con.)

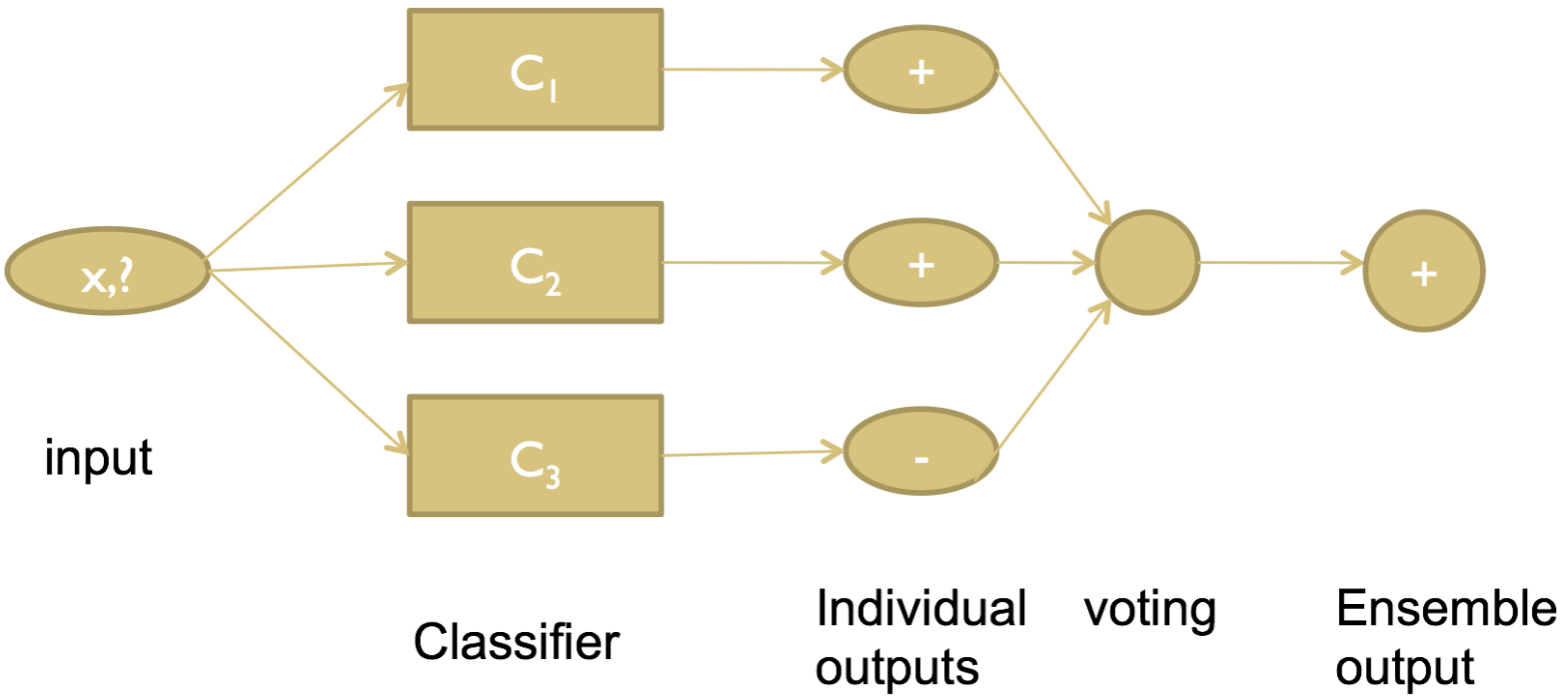

Figure: An example of an ensemble classifier.

Ensemble Classifier (con.)

It is often advantageous to take the training data and derive several sub-data sets from it, learn a classifier from each, and combing them to produce an ensemble model.

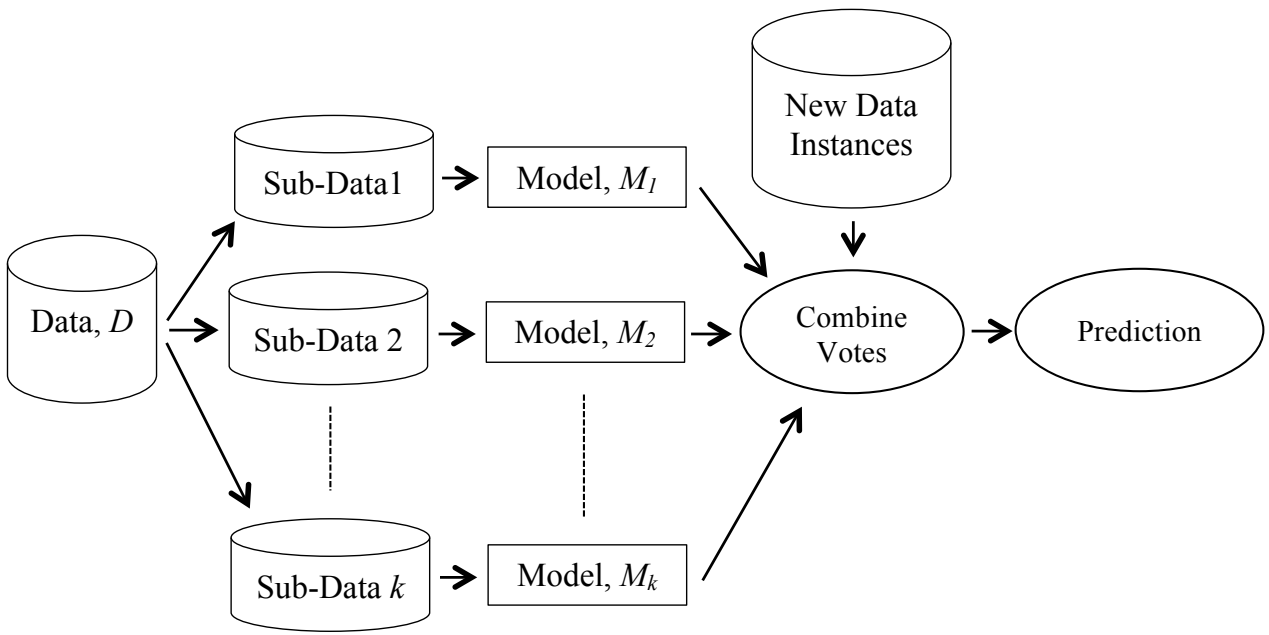

Figure: Ensemble model to improve classification accuracy.

Random Forest

The Random Forest also known as Random Decision Forest is an ensemble learning method for classification and regression that able to classify large amounts of data with accuracy.

- It constructs a number of decision trees based on randomly selected subset of features at training time and outputting the class that is the ensemble of trees vote for the most popular class.
- The selection of a random subset of features is an example of the random subspace method.

Random subspace method (or attribute bagging) is also an ensemble classifier that consists of several classifiers each operating in a subspace of the original feature space, and outputs the class based on the outputs of these individual classifiers. It is an attractive choice for classifying high dimensional data.

Bagging

- Combining the decision of different mining models means the various outputs into a single prediction. The simplest way to do this in the case of classification is to take a vote (perhaps a weighted vote); in the case of numeric prediction it is to calculate the average (perhaps a weighted average).
- **Bagging** and **boosting** both adopt this approach, but they derive the individual models in different ways. In bagging the models receive the equal weight, whereas in boosting weighting is used to give more influence to the more successful ones.
- To introduce bagging, suppose that several training datasets of the same size are chosen at random for the problem domain. Imaging using a particular machine learning technique to build a decision tree for each dataset. We can combine the trees by having them vote on each test instance. If one class receives more votes than any other, it is taken as the correct one. Predictions made by voting become more reliable as more votes are taken into account.

Bagging (con.)

Bagging also known as Bootstrap Aggregation, combines different classifiers into a single prediction model. It uses voting technique (perhaps a weighted vote) for classifying a new instance.

Algorithm 1 Bagging Algorithm

Input: Training data, D, number of iterations, k, and a learning scheme.
Output: Ensemble model, M^*
Method:

1: **for** i = 1 to k **do**
2: create bootstrap sample D_i, by sampling D with replacement;
3: use D_i, and learning scheme to derive a model, M_i;
4: **end for**

To use M^* to classify a new instance, x_{New}:
Each $M_i \in M^*$ classify x_{New} and return the majority vote;

Boosting

AdaBoost (short for Adaptive Boosting) is a popular boosting algorithm, which considers a series of classifiers and combines the votes of each individual classifier for classifying an unknown or known instances.

- In boosting, weights are assigned to each training instance.
- 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 instances that were misclassified by M_i.
- The final boosted classifier, M^*, combines the votes of each individual classifier.

Boosting (con.)

In **boosting**, weights are assigned to each training instance. 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 instances 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.

Boosting (con.)

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 instances, $(x_1, y_1), (x_2, y_2), \cdots, (x_d, y_d)$, where y_i is the class label of instance x_i. Initially, AdaBoost assigns each training instance an equal weight of $\frac{1}{d}$. Generating k classifiers for the ensemble requires k rounds through the rest of the algorithm. We can sample to form any sized training set, not necessarily of size d. Sampling with replacement is used - the same instance may be selected more than once. Each instance?s chance of being selected is based on its weight. A classifier model, M, is derived form the training instances of D_i. Its error is then calculated using D_i as a test set. The weights of the training instances are the adjusted according to how they were classified.

Boosting (con.)

If an instance was incorrectly classified, it weight is increased. If an instance was correctly classified, its weight is decreased. An instance'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 instances of the previous round. Some classifiers may be better at classifying some "difficult" instances than others. In this way, we build a series of classifiers that complement each other.

Error Rate

To compute the error rate of model M_i, we the sum the weights of each of the instances in D_i, that M_i misclassified. That is,

$$error(M_i) = \sum_{j=1}^{d} w_j * err(x_j) \tag{1}$$

Where $err(x_j)$ is the misclassification error of instance x_i. If the instance x_i was misclassified, then $err(x_i)$ is 1. Otherwise, it is 0. If the performance of classifier M_i is so poor that is 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.

Normalising Weight

If a instance in round i was correctly classified, its weight is multiplied by error $(\frac{error(M_i)}{1-error(M_i)})$. Once the weights of all of the correctly classified instances are updated, the weights for all instances (including the misclassified instances) are normalised so that their sum remains the same as it was before. To normalise 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 instances are increased and the weights of correctly classified instances are decreased.

AdaBoost Algorithm

Algorithm 2 AdaBoost Algorithm

Input: Training data, D, number of iterations, k, and a learning scheme.
Output: Ensemble model, M^*
Method:

1: initialise weight, $x_i \in D$ to $\frac{1}{d}$;
2: **for** i = 1 to k **do**
3: sample D with replacement according to instance weight to obtain D_i;
4: use D_i, and learning scheme to derive a model, M_i;
5: compute $error(M_i)$;
6: **if** $error(M_i) \geq 0.5$ **then**
7: go back to step 3 and try again;
8: **end if**
9: **for** each correctly classified $x_i \in D$ **do**
10: multiply weight of x_i by $(\frac{error(M_i)}{1-error(M_i)})$;
11: **end for**
12: normalise weight of instances;
13: **end for**

To use M^* to classify a new instance, x_{New}:

1: initialise weight of each class to zero;
2: **for** i = 1 to n **do**
3: $w_i = log\frac{1-error(M_i)}{error(M_i)}$; // weight of the classifier's vote
4: $c = M_i(x_{New})$; // class prediction by M_i
5: add w_i to weight for class c;
6: **end for**
7: return class with largest weight;

Ensemble of Trees

- It combines a number of decision trees in order to reduce the risk of overfitting.
- It creates several sub-datasets, $D_1, \cdots, D_i, \cdots, D_k$, from the original training data, D.
- It groups the features of each sub-dataset, D_i, and build a tree, DT_j on each group.
- It calculates the *error rate* of DT_j on sub-datasets, D_i. If *error rate* of DT_j is less than or equal to *threshold value* then the tree is consider for ensemble.
- To make a prediction on a new instance, each tree's prediction is counted as a vote for one class. The label is predicted to be the class that receives the most votes (majority voting).

Algorithm 3 Ensemble of Trees

Input: Training data, D, and C4.5 learning algorithm.
Output: A set of trees, DT^*
Method:

1: $DT^* = \emptyset$;
2: create sun-datasets, $D_1, \cdots, D_i, \cdots, D_k$, from the training data, D;
3: **for** i $= 1$ to k **do**
4: group features in D_i into m groups;
5: **for** j $= 1$ to m **do**
6: build a DT_j with jth feature group;
7: compute $error(DT_j)$ on D_i;
8: **if** $error(DT_j) \leq threshold\_value$ **then**
9: $DT^* = DT^* \cup DT_j$;
10: **end if**
11: **end for**
12: **end for**

To use DT^* to classify a new instance, x_{New}:
Each $DT_i \in DT^*$ classify x_{New} and return majority voting;

*** THANK YOU ***