Ensemble Methods in Machine Learning

AcadView

June 12, 2018

---

1 Overview

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model.

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods usually produces more accurate solutions than a single model would. This has been the case in a number of machine learning competitions, where the winning solutions used ensemble methods. In the popular Netflix Competition, the winner used an ensemble method to implement a powerful collaborative filtering algorithm. Another example is KDD 2009 where the winner also used ensemble methods. You can also find winners who used these methods in Kaggle competitions, for example here is the interview with the winner of CrowdFlower competition.

2 What are Ensemble Methods

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking).

Ensemble methods can be divided into two groups:

• **sequential ensemble methods** where the base learners are generated sequentially (e.g. AdaBoost). The basic motivation of sequential methods is to exploit the dependence between the base learners. The overall performance can be boosted by weighing previously mislabeled examples with higher weight.

• **parallel ensemble methods** where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

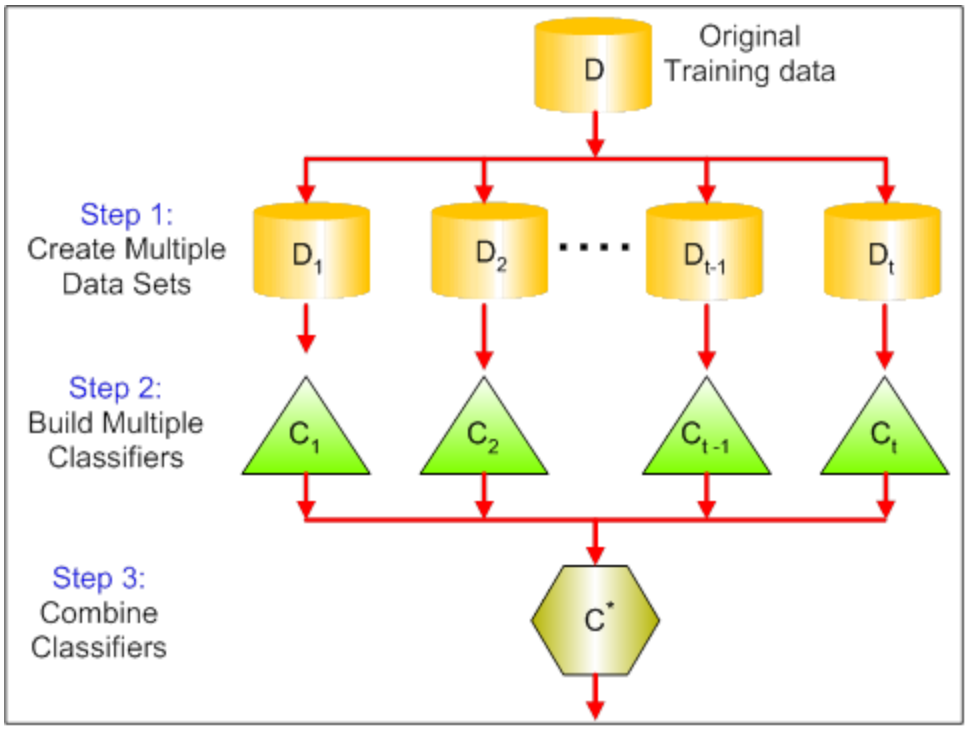
Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type, leading to homogeneous ensembles.

There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members, the base learners have to be as accurate as possible and as diverse as possible.

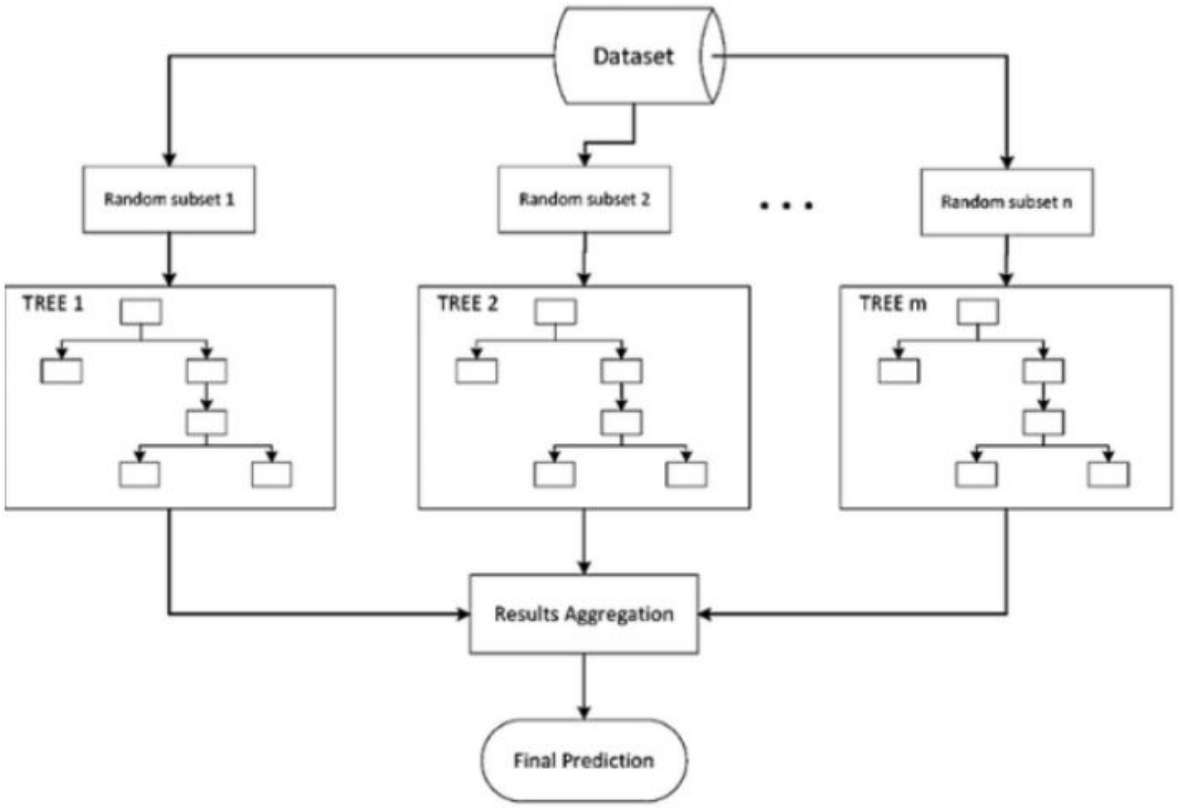
3 Some Commonly used Ensemble learning techniques

3.1 Bagging:

Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions. In generalized bagging, you can use different learners on different population. As you can expect this helps us to reduce the variance error.

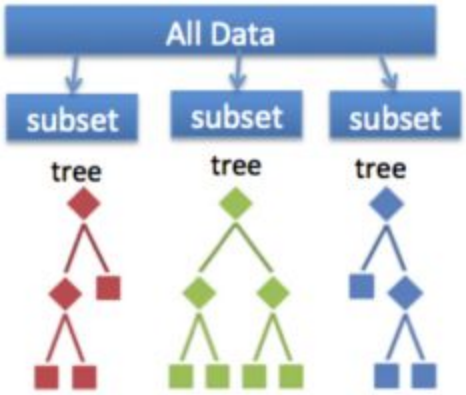


BAGGing gets its name because it combines Bootstrapping and Aggregation to form one ensemble model. Given a sample of data, multiple bootstrapped subsamples are pulled. A Decision Tree is formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. The image below will help explain:



3.1.1 Random Forest Models.

Random Forest Models can be thought of as BAGGing, with a slight tweak. When deciding where to split and how to make decisions, BAGGed Decision Trees have the full disposal of features to choose from. Therefore, although the bootstrapped samples may be slightly different, the data is largely going to break off at the same features throughout each model. In contrary, Random Forest models decide where to split based on a random selection of features. Rather than splitting at similar features at each node throughout, Random Forest models implement a level of differentiation because each tree will split based on different features. This level of differentiation provides a greater ensemble to aggregate over, ergo producing a more accurate predictor. Refer to the image for a better understanding.



A random forest takes a random subset of features from the data, and creates n random trees from each subset. Trees are aggregated together at end.

Similar to BAGGing, bootstrapped subsamples are pulled from a larger dataset. A decision tree is formed on each subsample. HOWEVER, the decision tree is split on different features (in this diagram the features are represented by shapes).

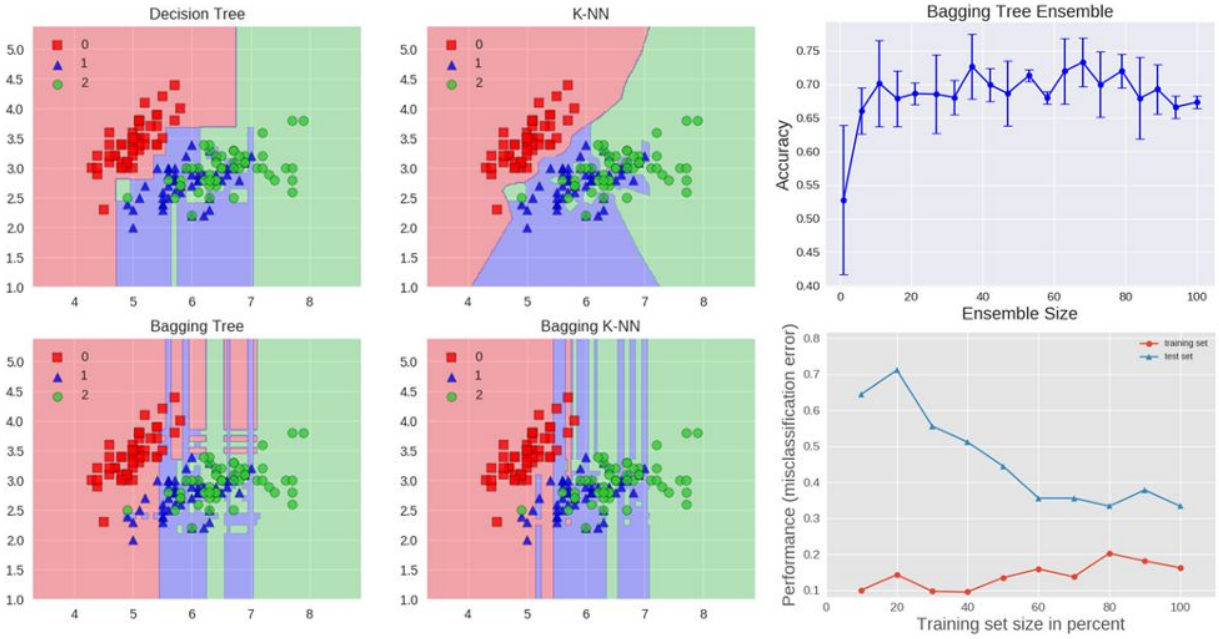
We can study bagging in the context of classification on the Iris dataset. We can choose two base estimators: a decision tree and a k-NN classifier. Figure 1 shows the learned decision boundary of the base estimators as well as their bagging ensembles applied to the Iris dataset.

Accuracy: 0.63 (+/- 0.02) [Decision Tree]

Accuracy: 0.70 (+/- 0.02) [K-NN]

Accuracy: 0.64 (+/- 0.01) [Bagging Tree]

Accuracy: 0.59 (+/- 0.07) [Bagging K-NN]



In the example below see an example of using the BaggingClassifier with the Classification and Regression Trees algorithm (DecisionTreeClassifier). A total of 100 trees are created.

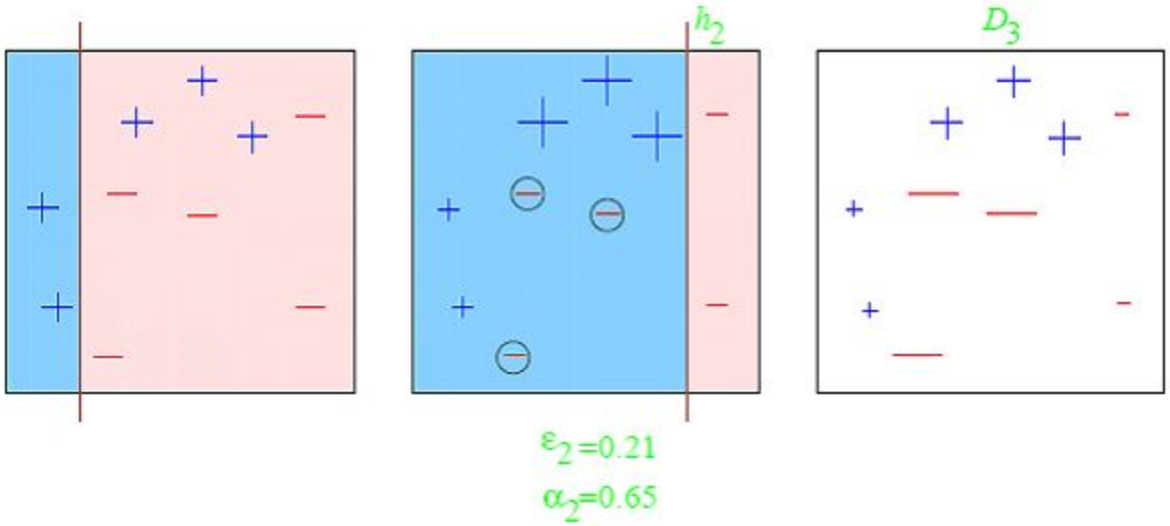
```
# Bagged Decision Trees for Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-di
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_stat
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example, we get a robust estimate of model accuracy.

```
0.770745044429
```

3.2 Boosting

Boosting is an iterative technique which adjust the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Boosting in general decreases the bias error and builds strong predictive models. However, they may sometimes over fit on the training data.



Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners models that are only slightly better than random guessing, such as small decision trees to weighted versions of the data. More weight is given to examples that were miss classified by earlier rounds.

The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods, such as bagging, is that base learners are trained in sequence on a weighted version of the data. The two most common boosting ensemble machine learning algorithms are:

• AdaBoost

• Stochastic Gradient Boosting

3.2.1 AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or or less attention to them in the construction of subsequent models.

You can construct an AdaBoost model for classification using the AdaBoostClassifier class.

The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

```
# AdaBoost Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import AdaBoostClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-di
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 30
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example provides a mean estimate of classification accuracy.

```
0.76045796309
```

3.2.2 Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps of the the best techniques available for improving performance via ensembles.

You can construct a Gradient Boosting model for classification using the GradientBoostingClassifier class.

The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

```
# Stochastic Gradient Boosting Classification
import pandas
from sklearn import model_selection
from sklearn.ensemble import GradientBoostingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-di
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
num_trees = 100
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Running the example provides a mean estimate of classification accuracy.

```
0.764285714286
```

3.3 Voting Ensemble

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms. It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data.

The predictions of the sub-models can be weighed, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from submodels, but this is called stacking (stacked aggregation) and is currently not provided in scikit-learn.

You can create a voting ensemble model for classification using the VotingClassifier class.

The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification

problem.

```
# Voting Ensemble for Classification
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-di
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```

Running the example provides a mean estimate of classification accuracy.

```
0.712166780588
```