ML | Bagging classifier

In machine learning, for building solid and reliable models, prediction accuracy is the key factor. Ensemble learning is a supervised machine-learning technique that combines multiple models to build a more powerful and robust model. The idea is that by combining the strengths of multiple models, we can create a model that is more robust and less likely to overfit the data. It can be used for both classification and regression tasks.

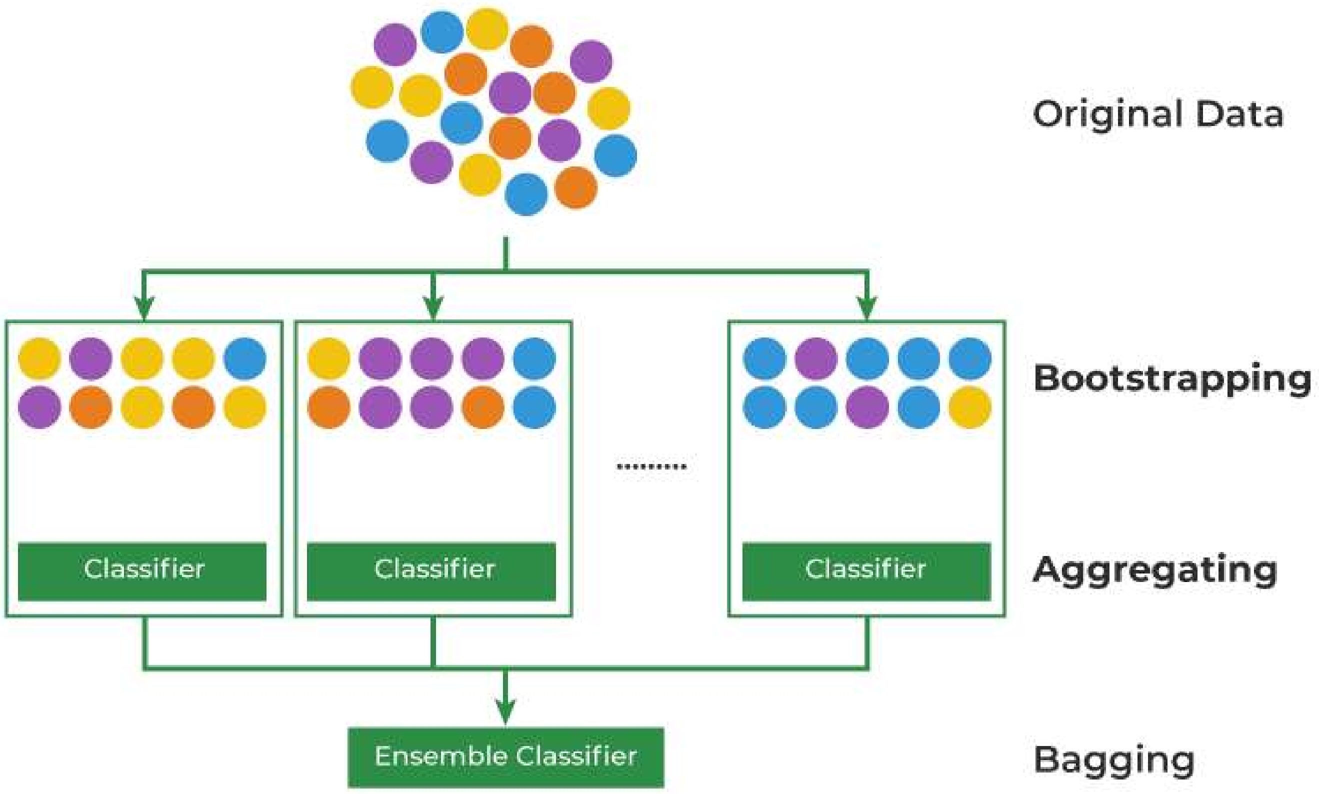
Ensemble learning techniques can be categorized in three ways:

1. Bagging (Bootstrap Aggregating)
2. Boosting
3. Stacking (Stacked Generalization)

Bagging is a supervised machine-learning technique, and it can be used for both regression and classification tasks. In this article, we will discuss the bagging classifier.

Bagging Classifier

Bagging (or Bootstrap aggregating) is a type of ensemble learning in which multiple base models are trained independently and in parallel on different subsets of the training data. Each subset is generated using bootstrap sampling, in which data points are picked at random with replacement. In the case of the bagging classifier, the final prediction is made by aggregating the predictions of the all-base model using majority voting. In the models of regression, the final prediction is made by averaging the predictions of the all-base model, and that is known as bagging regression.



Bagging Classifier

Bagging helps improve accuracy and reduce overfitting, especially in models that have high variance.

How does Bagging Classifier Work?

The basic steps of how a bagging classifier works are as follows:

- **Bootstrap Sampling:** In Bootstrap Sampling randomly ‘n’ subsets of original training data are sampled with replacement. This step ensures that the base models are trained on diverse subsets of the data, as some samples may appear multiple times in the new subset, while others may be omitted. It reduces the risks of overfitting and improves the accuracy of the model.

```
 Let's break it down step by step:
Original training dataset: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Resampled training set 1: [2, 3, 3, 5, 6, 1, 8, 10, 9, 1]
Resampled training set 2: [1, 1, 5, 6, 3, 8, 9, 10, 2, 7]
Resampled training set 3: [1, 5, 8, 9, 2, 10, 9, 7, 5, 4]
```

- **Base Model Training:** In bagging, multiple base models are used. After the Bootstrap Sampling, each base model is **independently trained** using a specific learning algorithm, such as decision trees, support vector machines, or neural networks on a different bootstrapped subset of data. These models are typically called “Weak learners” because they may not be highly accurate on their own. Since the base model is trained independently of different subsets of data. To make the model computationally efficient and less time-consuming, the base models can be trained in **parallel**.

- **Aggregation:** Once all the base models are trained, it is used to make predictions on the unseen data i.e. the subset of data on which that base model is not trained. In the bagging classifier, the predicted class label for the given instance is chosen based on the majority voting. The class which has the majority voting is the prediction of the model.
- **Out-of-Bag (OOB) Evaluation:** Some samples are excluded from the training subset of particular base models during the bootstrapping method. These “out-of-bag” samples can be used to estimate the model’s performance without the need for cross-validation.
- **Final Prediction:** After aggregating the predictions from all the base models, Bagging produces a final prediction for each instance.

Algorithm for the Bagging classifier:

```
Classifier generation:
Let N be the size of the training set.
for each of t iterations:
    sample N instances with replacement from the original training set.
    apply the learning algorithm to the sample.
    store the resulting classifier.
Classification:
for each of the t classifiers:
    predict class of instance using classifier.
return class that was predicted most often.
```

Python implementation of the Bagging classifier algorithm:

BaggingClassifier

Define the BaggingClassifier class with the base_classifier and n_estimators as input parameters for the constructor.

Step 1: Initialize the class attributes base_classifier, n_estimators, and an empty list classifiers to store the trained classifiers.

Step 2: Define the **fit method** to train the bagging classifiers:

For each iteration from 0 to n_estimators – 1:

- Perform bootstrap sampling with replacement by randomly selecting len(X) indices from the range of len(X) with replacement.

    - Create new subsets X_sampled and y_sampled by using the selected indices.
- Create a new instance of the base_classifier to create a new classifier model for this iteration.
- Train the classifier on the sampled data X_sampled and y_sampled.
- Append the trained classifier to the list classifiers.
    - Return the list of trained classifiers.

Step 3: Define the predict method to make predictions using the ensemble of classifiers:

- For each classifier in the classifiers list:
    - Use the trained classifier to predict the classes of the input data X.
        - Aggregate the predictions using majority voting to get the final predictions.
- Return the final predictions.

Python3

```
   `
`class` `BaggingClassifier:`

    `def` `__init__(``self``, base_classifier, n_estimators):`

        `self``.base_classifier` `=` `base_classifier`

        `self``.n_estimators` `=` `n_estimators`

        `self``.classifiers` `=` `[]`

    `def` `fit(``self``, X, y):`

          `for` `_` `in` `range``(``self``.n_estimators):`

              `indices` `=` `np.random.choice(``len``(X),` `len``(X), replace``=``True``)`

              `X_sampled` `=` `X[indices]`

              `y_sampled` `=` `y[indices]`

              `classifier` `=` `self``.base_classifier.__class__()`

              `classifier.fit(X_sampled, y_sampled)`

              `self``.classifiers.append(classifier)`

          `return` `self``.classifiers`

    `def` `predict(``self``, X):`
```

```
        `predictions` `=` `[classifier.predict(X)` `for` `classifier` `in` `self``.classifiers
        `majority_votes` `=` `np.apply_along_axis(``lambda` `x: np.bincount(x).argmax(), axis`
        `return` `majority_votes`


`
```

Importing Necessary Libraries

Python3

```
from sklearn.datasets import load_digits

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.tree import DecisionTreeClassifier
```

Load the dataset

Python3

```
digit = load_digits()

X, y = digit.data, digit.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size``=``0.2``,
random_state``=``42``)

dc = DecisionTreeClassifier()

model = BaggingClassifier(base_classifier``=``dc, n_estimators``=``10``)

classifiers = model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print``(``"Accuracy:"``, accuracy)
```

Output:

```
Accuracy: 0.9472222222222222
```

Let's check the result of each classifier individually,

Python3

```
for  i, clf  in  enumerate``(classifiers):

  y_pred  =  clf.predict(X_test)

  accuracy  =  accuracy_score(y_test, y_pred)

  print``(``"Accuracy:"``+``str``(i``+``1``),``':'``, accuracy)
```

Output:

```
Accuracy:1 : 0.8833333333333333
Accuracy:2 : 0.8361111111111111
Accuracy:3 : 0.85
Accuracy:4 : 0.85
Accuracy:5 : 0.8388888888888889
Accuracy:6 : 0.8388888888888889
Accuracy:7 : 0.8472222222222222
Accuracy:8 : 0.8222222222222222
Accuracy:9 : 0.8527777777777777
Accuracy:10 : 0.8111111111111111
```

Bagging classifier using Sklearn Library

Python3

```
from  sklearn.ensemble  import  BaggingClassifier

digit  =  load_digits()

X, y  =  digit.data, digit.target

X_train, X_test, y_train, y_test  =  train_test_split(X, y, test_size``=``0.2``,
random_state``=``42``)

base_classifier  =  DecisionTreeClassifier()
```

```
 n_estimators  =  10

 bagging_classifier  =  BaggingClassifier(base_estimator``=``base_classifier,
n_estimators``=``n_estimators)

 bagging_classifier.fit(X_train, y_train)

 y_pred  =  bagging_classifier.predict(X_test)

 accuracy  =  accuracy_score(y_test, y_pred)

 print``(``"Accuracy:"``, accuracy)
```

Output:

```
  Accuracy: 0.9361111111111111
```

Advantages of Bagging Classifier

The advantages of using a Bagging Classifier are as follows:

1. **Improved Predictive Performance:** Bagging Classifier often outperforms single classifiers by reducing overfitting and increasing predictive accuracy. By combining multiple base models, it can better generalize to unseen data.
2. **Robustness**: Bagging reduces the impact of outliers and noise in the data by aggregating predictions from multiple models. This enhances the overall stability and robustness of the model.
3. **Reduced Variance**: Since each base model is trained on different subsets of the data, the aggregated model's variance is significantly reduced compared to an individual model.
4. **Parallelization**: Bagging allows for parallel processing, as each base model can be trained independently. This makes it computationally efficient, especially for large datasets.
5. **Flexibility**: Bagging Classifier is a versatile technique that can be applied to a wide range of machine learning algorithms, including decision trees, random forests, and support vector machines.

Disadvantages of Bagging :

1. **Loss of Interpretability:**
   - Bagging involves aggregating predictions from multiple models, making it harder to interpret the individual contributions of each base model. This can limit the ability to derive precise business insights from the ensemble.
2. **Computationally Expensive:**

- As the number of iterations (bootstrap samples) increases, the computational cost of bagging also grows. This makes it less suitable for real-time applications where efficiency and speed are crucial. Efficient parallel processing or distributed systems are often required for handling large datasets and numerous iterations.

3. **Less Flexible:**
    - Bagging is most effective when applied to algorithms that are less stable or more prone to overfitting. Algorithms that are already stable or exhibit low bias might not benefit significantly from bagging, as there is less variance to be reduced within the ensemble.

Applications of Bagging Classifier

Bagging Classifier can be applied in various real-world tasks:

1. **Fraud Detection**: Bagging Classifier can be used to detect fraudulent transactions by aggregating predictions from multiple fraud detection models.
2. **Spam filtering**: Bagging classifier can be used to filter spam emails by aggregating predictions from multiple spam filters trained on different subsets of the spam emails.
3. **Credit scoring**: Bagging classifier can be used to improve the accuracy of credit scoring models by combining the predictions of multiple models trained on different subsets of the credit data.
4. **Image Classification**: Bagging classifier can be used to improve the accuracy of image classification tasks by combining the predictions of multiple classifiers trained on different subsets of the training images.
5. **Natural language processing:** In NLP tasks, the bagging classifier can combine predictions from multiple language models to achieve better text classification results.

Conclusion

Bagging Classifier, as an ensemble learning technique, offers a powerful solution for improving predictive performance and model robustness. Bagging Classifier avoids overfitting, improves generalisation, and gives solid predictions for a wide range of applications by using the collective wisdom of numerous base models.