Javier Fernandez Follow

Mar 21, 2022 · 11 min read · ✦ · Listen

Learning

This article introduces Voting, Bagging, Boosting, and Stacking, the main ensemble methods used in machine learning

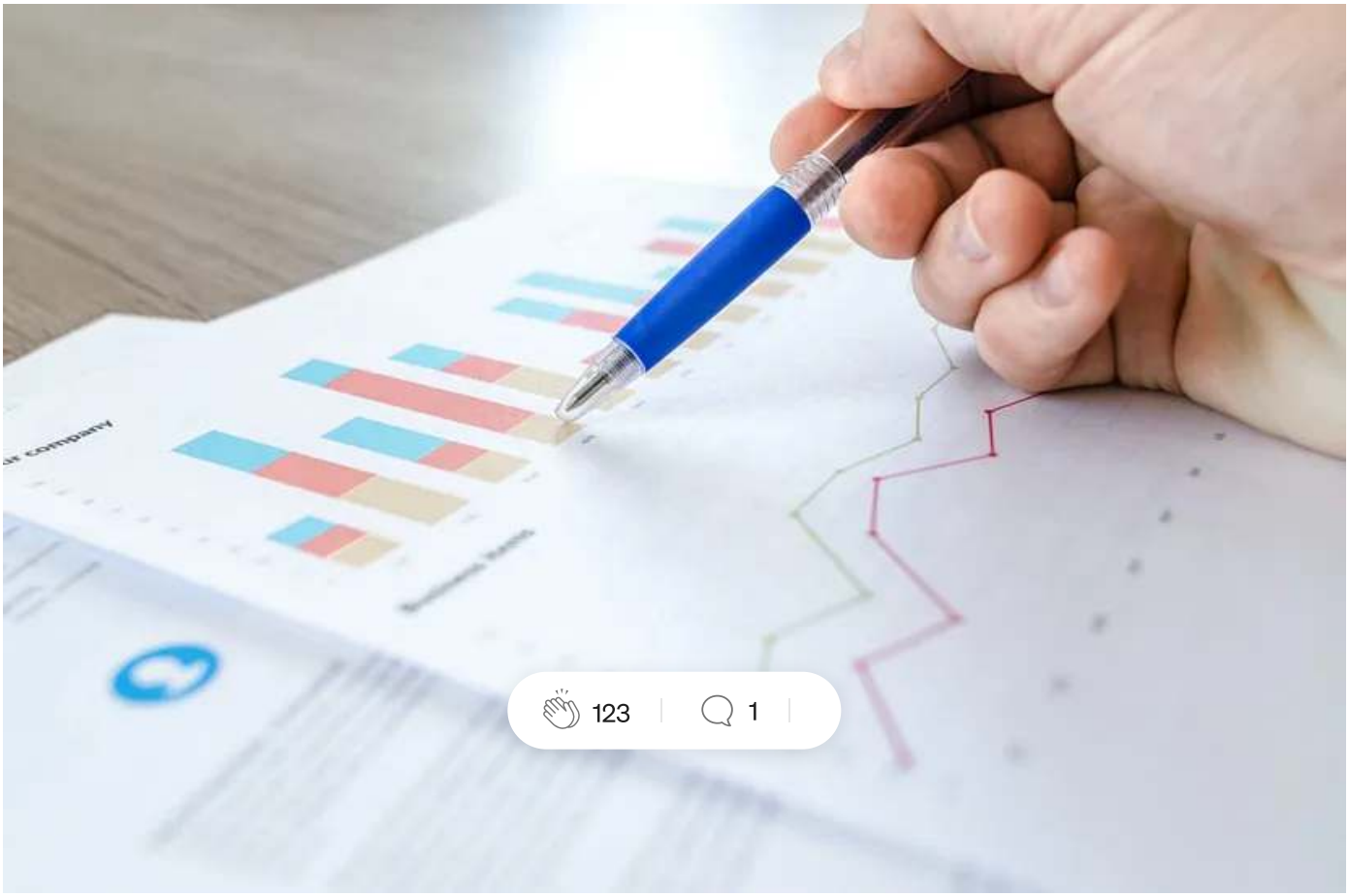

Photo by Lukas on Pexels

In machine learning, the *bias* is the difference between the prediction of the model and the ground truth value, while the *variance* is the error from sensitivity to small fluctuations in the training set.

Having high bias can cause an algorithm to miss the relevant relations between the target outputs and the features(underfitting) while high variance may result from an algorithm modeling the random noise in the training data (overfitting).

The *bias-variance tradeoff* is the property of a model that says that the bias in the estimated parameters can be reduced at the cost of increasing the variance of the parameter estimated across samples.

One way of resolving the bias-variance tradeoff is to use mixture models and ensemble learning. Generally, the main ensemble methods are *voting, stacking, bagging*, and *boosting*.

This article covers these four ensemble methods, introducing each method with its main advantages and disadvantages as well as its implementation in Python.

> *Before moving forward in the article, if you are* ***not*** *familiar with the bias and variance terms, I recommend taking a look at the article Bias-Variance Decomposition for Model Assessment.*

1. Ensemble methods

1.1. Voting

A *voting ensemble method* is a machine learning model that combines the predictions from different models to output a final prediction. For this ensemble method, the models should be different since it uses all the training data to train the models.

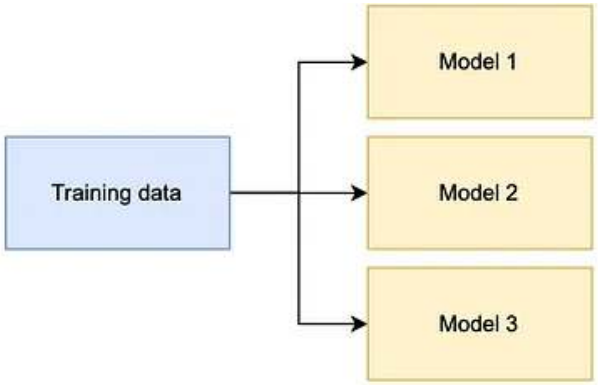

Figure 1. Training models of the voting ensemble method. **Ref**: Image by author.

For regression tasks, the output is the average of the predictions of the models. Instead, for classification tasks, there are two ways of estimating the final output: hard voting and soft voting. The former consists of taking the mode from all the predictions of the model (Figure 2). The latter uses the highest probability after averaging the probabilities for all the models (Figure 3).

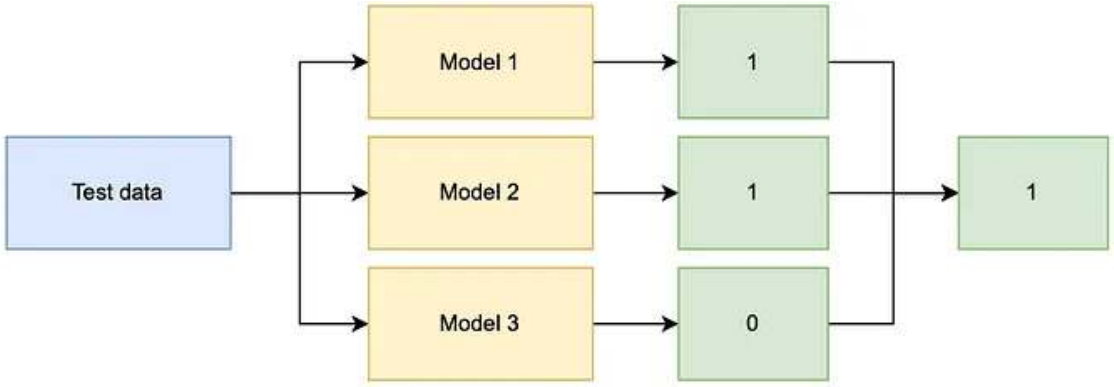

Figure 2. Hard voting. **Ref**: Image by author.

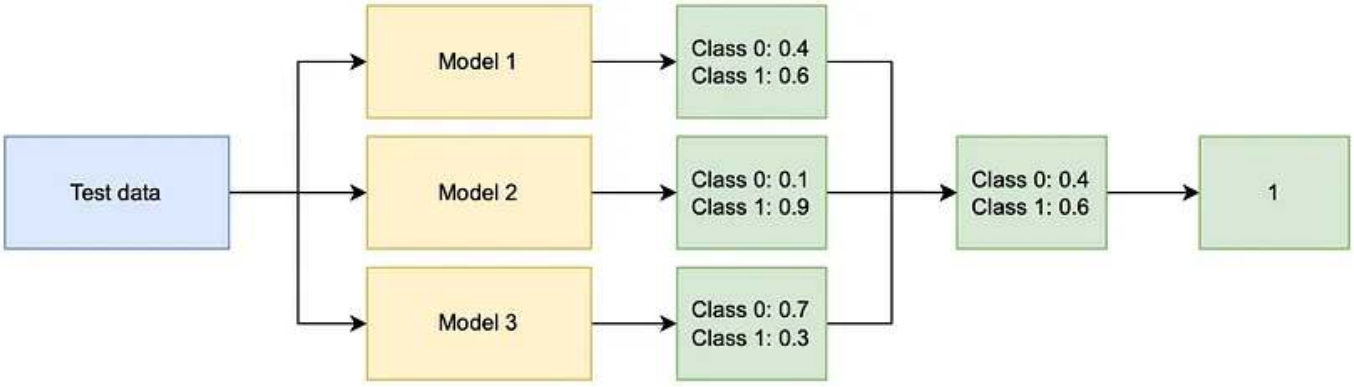

Figure 3. Soft voting. **Ref**: Image by author.

The main idea behind voting is to be able to generalize better by compensating for the errors of each model separately, especially when the models predict well on a predictive modeling task.

This ensemble method is particularly useful for stochastic machine learning models, such as the neural networks, as they result in a different model for each run. It is also useful when combining multiple fits of the same machine learning algorithm if it is observed that the same model performs well when using different hyperparameters.

1.2. Stacking

The stacking ensemble models are an extension of the voting ensemble models, as they use weighted voting of the contributing models to avoid all models contributing equally to the prediction.

The architecture of the stacking models involves *base-models* (models fitted on the training data) and a *meta-model* (model that learns how to combine the predictions of

the base models). It is a common practice to use a linear regression model for regression tasks and a logistic regression model for classification tasks.

The meta-model is trained on the predictions made by base models on out-of-sample data. In other words, (1) data not used to train the base models are fed to the base models, (2) predictions are made from these base models, and (3) these predictions and the ground truth labels are provided to fit the meta-model.

The input of the meta-model depends on the task. For regression tasks, the input is the predicted value. For binary classification tasks, the input is typically the predicted value for the positive class. Lastly, for multiclass classification, it is usually the set of predicted values for all the classes.

There is a type of stacking model called *blending* commonly used in the literature. While stacking models are trained on out-of-fold predictions made during k-fold cross-validation, blending models are trained on predictions made on a holdout dataset.

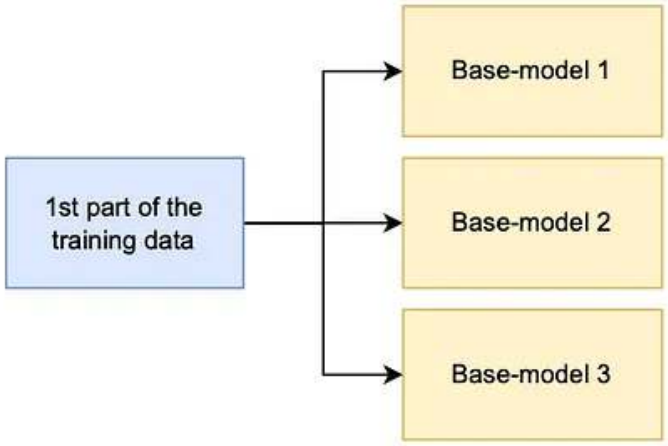

Figure 4. Training base-models of the stacking ensemble method. **Ref**: Image by author.

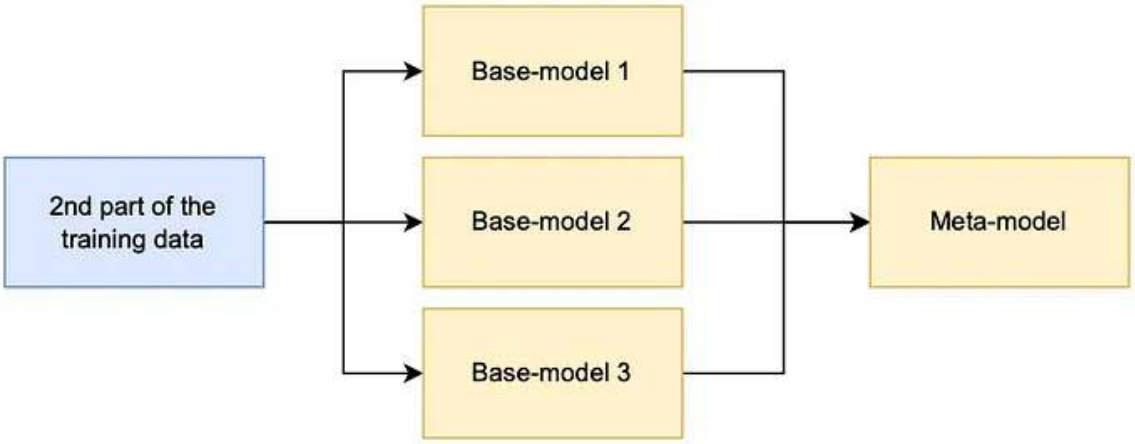

Figure 5. Training meta-models of the stacking ensemble method. **Ref**: Image by author.

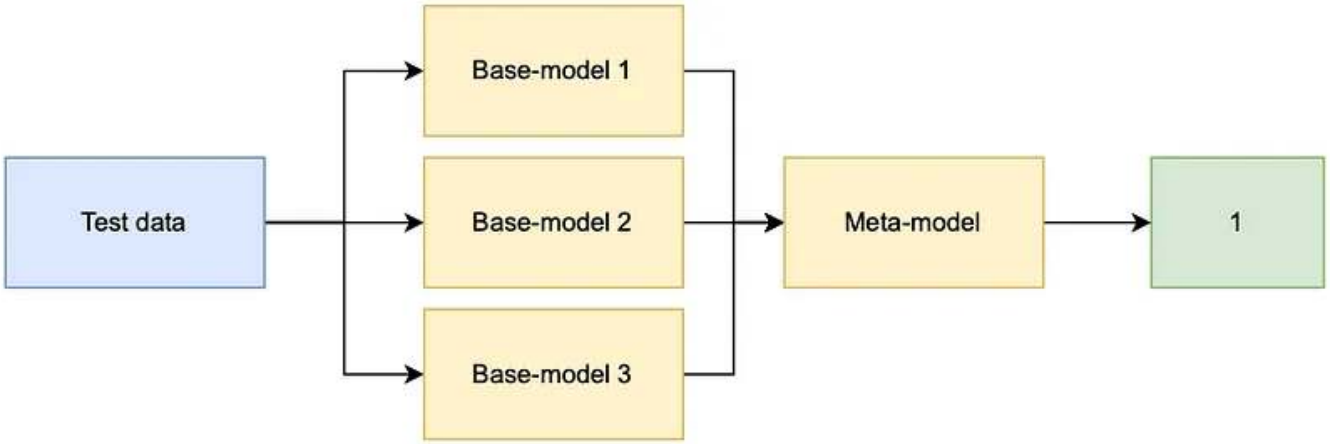

Figure 6. New data on stacking ensemble method. **Ref**: Image by author.

Stacking is designed to improve modeling performance, although is not guaranteed to result in an improvement in all cases. If any of the base models perform like the stacking ensemble method, this should be preferable use as it is easier to explain and maintain.

1.3. Bagging

Bagging is the process of sub-sampling training data to improve the generalization performance of a single type of classifier. This technique is effective on models which tend to overfit the dataset.

The method used to sub-sample the data is *bootstrapping* (the name *bagging* comes from *bootstrap* + *aggregating*). This method consists of doing random sampling with replacement over the data, which means that the subsets of the training data will overlap since we are not splitting the data but resampling it.

Once obtained the output of each of the models, the final prediction for each data can be obtained by doing *regression voting*, where predictions are the average of contributing models, or *classification voting,* where predictions are the majority vote of contributing models.

Some important notes to take into account when implementing this method are that (1) the hyperparameters of the classifier don't change from subsample to subsample, (2) the improvement is usually not really significant, (3) it is expensive as it may increase the computational costs by 5 or 10 times, and (4) this is a bias-reduction technique, so it does not help when you are variance-limited.

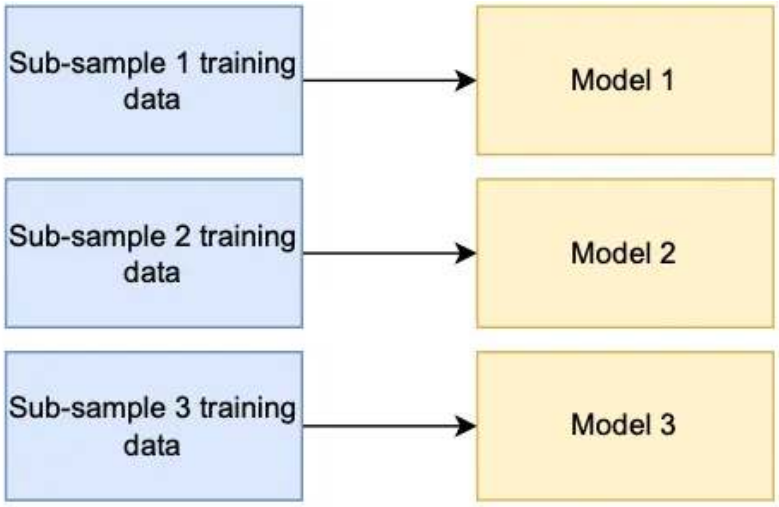

Figure 7. Training models of the bagging ensemble method. **Ref**: Image by author.

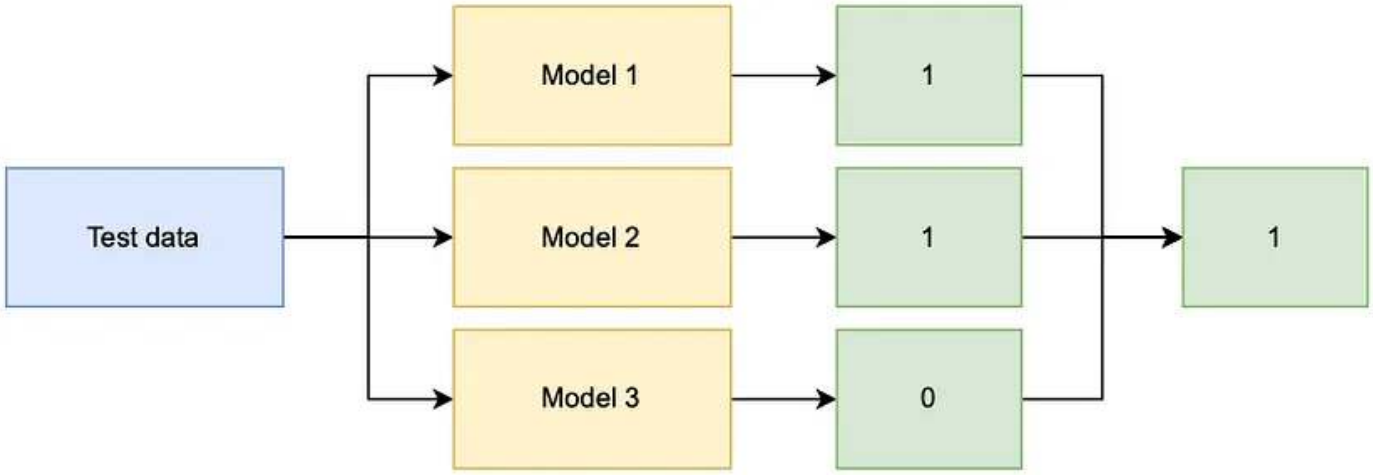

Figure 8. New data on bagging ensemble method. **Ref**: Image by author.

This ensemble method is especially useful when overfitting the data, but not when underfitting it, as it reduces the variance by generalizing better.

1.4. Boosting

Boosting is an ensemble method that builds the model by using weak learners in series. The main idea is that each successive model corrects the errors of its prior model.

Even though there are several types of boosting algorithms such as Gradient Boosting or XGBoosting, the first boosting algorithm developed for the purpose of binary classification was AdaBoost. The way AdaBoosting works is displayed in a simplified way in Figure 9, where the task is to classify the circles the squares based on the x and y features. The first model classifies the data points by generating a vertical separator line. But, as observed, it wrongly classifies some of the circles' data points. Hence, the second model focus on classifying these misclassified data points by increasing the weight of the wrongly classified data points. This process is iteratively done for the number of estimators defined when declaring the object. Since explaining the AdaBoost algorithm in more detail would take a whole article by itself, here I leave this video for anyone interested.

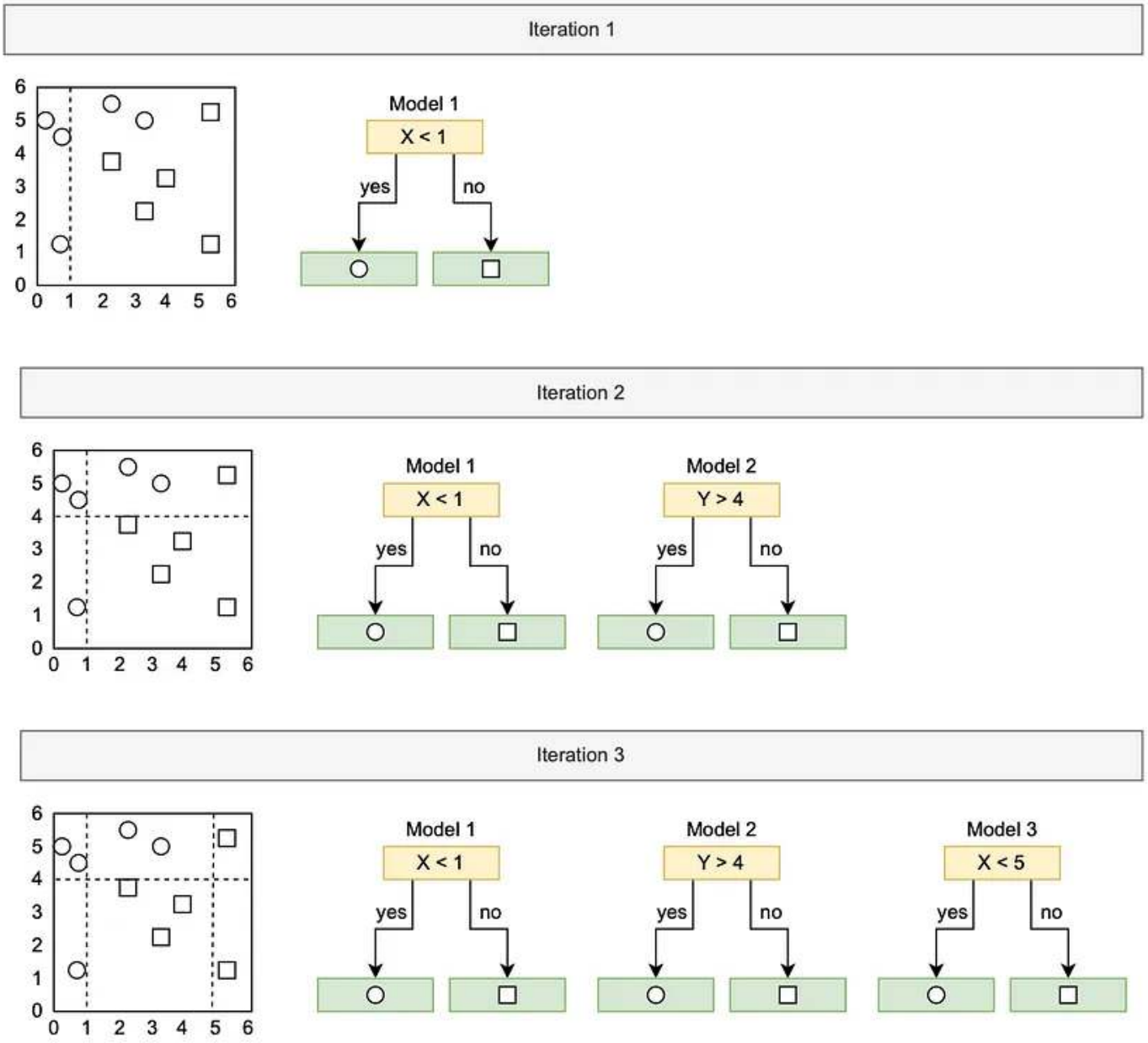

Figure 9. The training methodology for the boosting ensemble method. **Ref**: Image by author.

Boosting has been shown to achieve accurate results in the literature as they are a resilient method that curbs overfitting easily. However, it is sensitive to outliers as every classifier is obliged to fix the errors in the predecessors, and it is also difficult to scale up since each estimator bases its correctness on the previous predictors.

2. Implementation

The implementations for each of the ensemble methods described above are presented throughout this section.

```
import numpy as np
import matplotlib.pyplot as plt
import time

from sklearn.datasets import make_moons
from sklearn.ensemble._bagging import BaggingClassifier
from sklearn.ensemble import VotingClassifier, StackingClassifier, RandomForestClassifier, AdaBoos
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.base import BaseEstimator

import torch
import torch.nn as nn
import torch.nn.functional as F

%matplotlib inline
```

ensemble_libraries hosted with ❤ by GitHub view raw

Here is the dataset randomly created using the *make_moons* function, which creates two interleaving half circles.

```
np.random.seed(16)
tr_x, tr_y = make_moons(100, noise=0.3)
ts_x, ts_y = make_moons(1000, noise=0.3)

xx, yy = np.meshgrid(np.arange(-2, 3, 0.05), np.arange(-1.5, 2.0, 0.05))
zz = np.concatenate([xx.ravel()[:, np.newaxis], yy.ravel()[:, np.newaxis]], axis=1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].scatter(tr_x[:,0], tr_x[:,1], c=tr_y, cmap=plt.cm.Spectral)
axs[0].set_title('Train set')
axs[1].scatter(ts_x[:,0], ts_x[:,1], c=ts_y, cmap=plt.cm.Spectral)
axs[1].set_title('Test set')
plt.show()
```

bagging_dataset hosted with ❤ by GitHub view raw

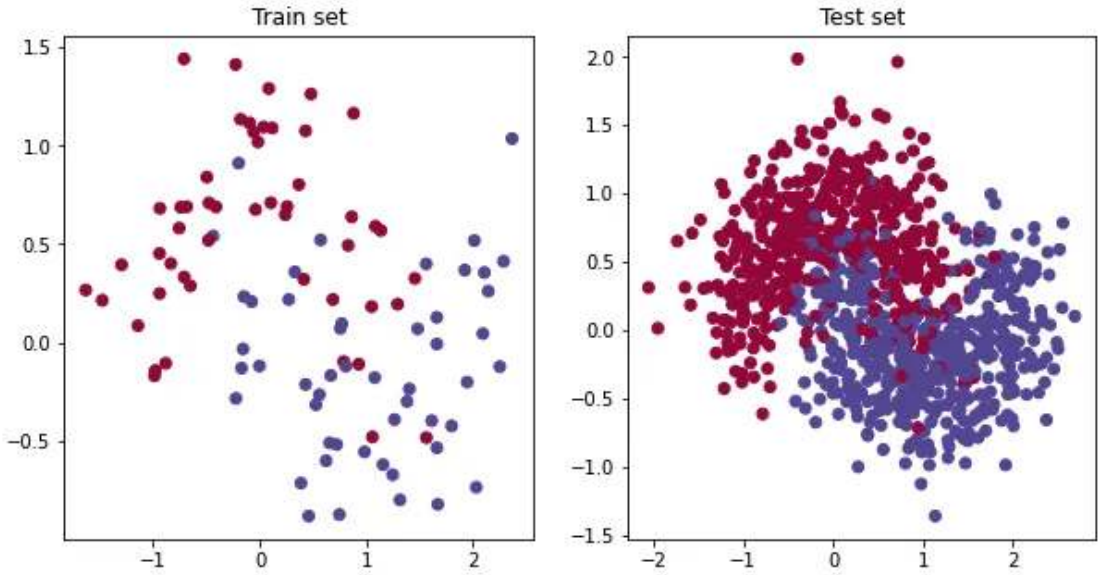

Figure 10. Training and test set. **Ref**: Image by author.

To assess the results for each ensemble method, we have run as benchmarks an SVM classifier, a decision tree classifier, and a logistic regression. The SVM classifier is a machine learning algorithm based on finding a hyperplane in N-dimensional space (being *N* the number of features) that distinctly classifies the data. The decision tree is tree-structured classified where internal nodes represent the features of the dataset, branches indicate the decision rules, and each leaf node is the outcome. The logistic regressor is a statistical analysis method to predict a binary outcome based on prior observations.

> ***Note 1**: I have just selected these three models to compare different machine learning algorithms, not because they are most optimal ones for this task.*
>
> ***Note 2**: For the SVM classfier, the C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly [1]. Hence, I have selected a regularization term C of 10000 so that the model overfits the training data.*

- SVC: accuracy of 84.2% and runtime of 0.07 seconds.
- Decision tree: accuracy of 86.6% and runtime of 0.02 seconds.
- Logistic regressor: accuracy of 83.5% and runtime of 0.02 seconds.

```
clf_svc = SVC(C=10000, gamma='scale', probability=True)  # Support vector classifier
clf_dt = DecisionTreeClassifier()                        # Decision tree
clf_lr = LogisticRegression()                            # Logistic regressor
```

ensemble_benchmarks hosted with ❤ by GitHub view raw

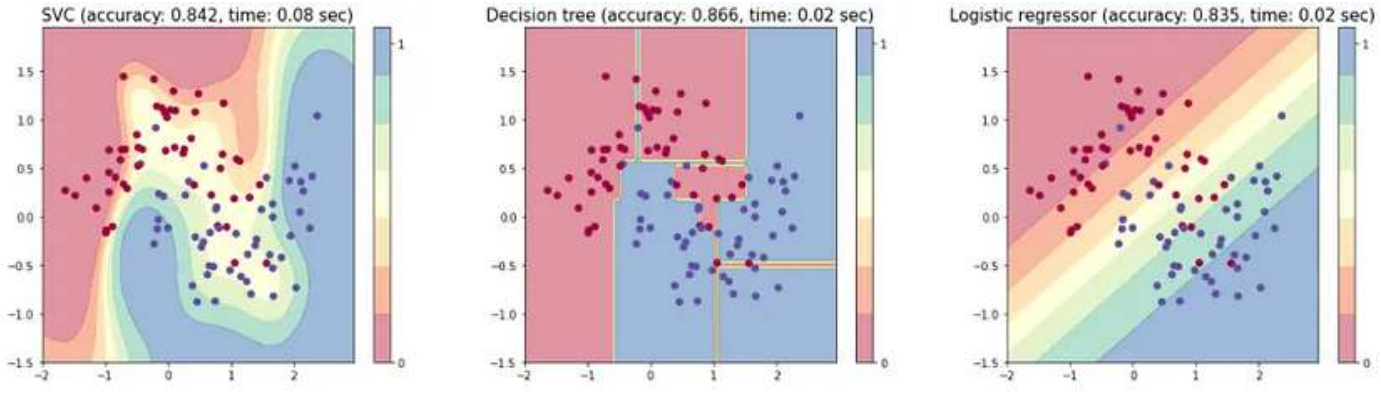

Figure 11. Accuracy and computational time for the SVC, decision tree, and logistic regressor models. **Ref**: Image by author.

For each of these plots and the ones displayed below, I have used the following code to visualize the classification results for each of the models. While the plot displayed is for the training data, both the accuracy and time are based on the test set.

```
time_now = time.time()
clf.fit(tr_x, tr_y)
p = clf.predict_proba(zz)
p = p.reshape((xx.shape[0], xx.shape[1], 2))
cax = axs[0].contourf(xx, yy, p[:,:,1], alpha=0.5, cmap=plt.cm.Spectral)
cbar = plt.colorbar(cax, ticks=[-1, 0, 1], ax=axs[0])
axs[0].scatter(tr_x[:,0], tr_x[:,1], c=tr_y,cmap=plt.cm.Spectral)
axs[0].set_title(f'Accuracy: {clf.score(ts_x, ts_y)}, time: {(time.time() - time_now):.2f} sec')
```

ensemble_plotting_code hosted with ❤ by GitHub view raw

2.1. Voting

Here are the results when implementing the voting ensemble with hard voting and soft voting:

- Hard voting: accuracy of 88.2% and runtime of 0.12 seconds.
- Decision tree: accuracy of 87.3% and runtime of 0.09 seconds.

```
clf_hv = VotingClassifier(estimators=[('lr', clf_svc), ('rf', clf_dt), ('gnb', clf_lr)], voting='ha
clf_sv = VotingClassifier(estimators=[('lr', clf_svc), ('rf', clf_dt), ('gnb', clf_lr)], voting='so
```

ensemble_voting hosted with ❤ by GitHub view raw

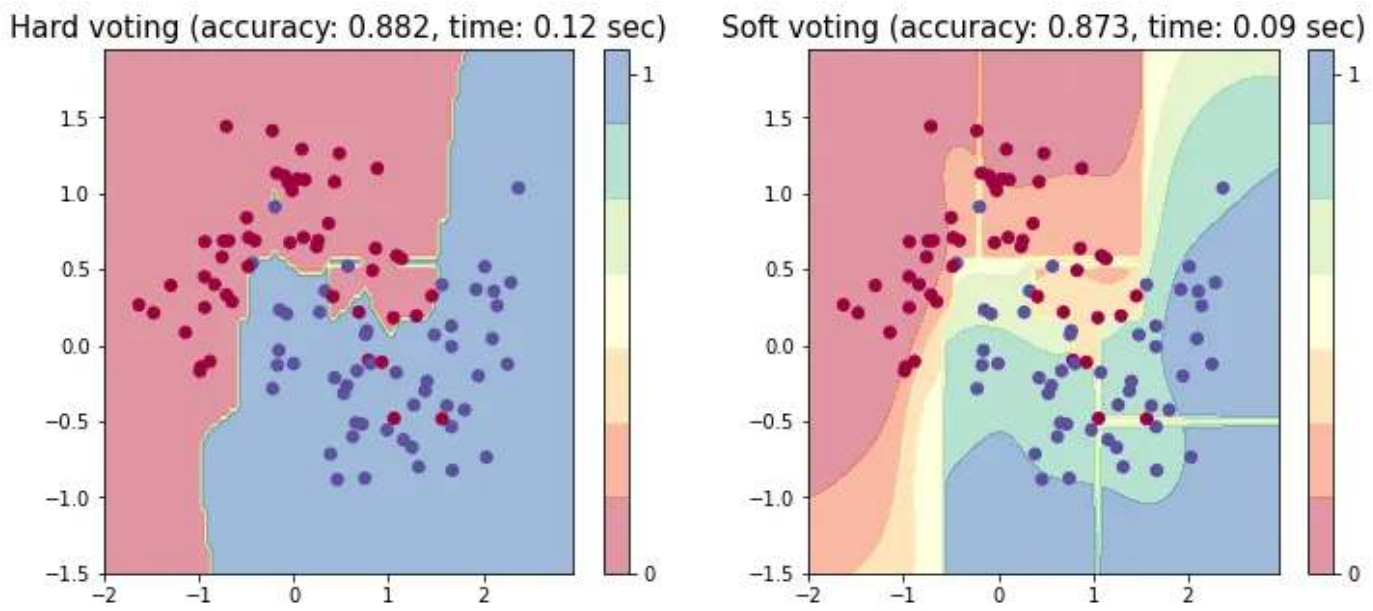

Figure 12. Accuracy and computational time for the hard voting and soft voting models. **Ref**: Image by author.

It is observed how both outputs look like the plot for the decision tree in Figure 11, but the models generalize better for the training set as they misclassify the red data points contained within the blue data points and vice versa. The main difference between hard voting and soft voting is observed for the region located at the center of the image, where the region considered as one class or the other has been influenced by the weight given to the decision tree classifier.

Regarding hard voting, it does not output probabilities so it is needed to change the line code `clf.predict_proba(zz)` by `clf.predict(zz)` .

```
time_now = time.time()
clf.fit(tr_x, tr_y)
p = clf.predict(zz)
p = p.reshape((xx.shape[0], xx.shape[1]))
cax = axs[axs_no].contourf(xx, yy, p[:,:], alpha=0.5, cmap=plt.cm.Spectral)
cbar = plt.colorbar(cax, ticks=[-1, 0, 1], ax=axs[axs_no])
axs[axs_no].scatter(tr_x[:,0], tr_x[:,1], c=tr_y,cmap=plt.cm.Spectral)
axs[axs_no].set_title(f'Hard voting (accuracy: {clf.score(ts_x, ts_y)}, time: {(time.time() - time_
```

ensemble_hard_voting hosted with ❤ by GitHub view raw

2.2. Stacking

The selected meta-model for the stacking ensemble method is a logistic regressor as we are in a binary classification task, trained using a 5-fold cross-validation.

- Stacking: accuracy of 88.4% and runtime of 0.24 seconds.

```
clf = StackingClassifier(estimators=[('lr', clf_svc), ('rf', clf_dt), ('gnb', clf_lr)],
                         final_estimator=LogisticRegression(),
                         cv=5)
```

ensemble_stacking hosted with ❤ by GitHub view raw

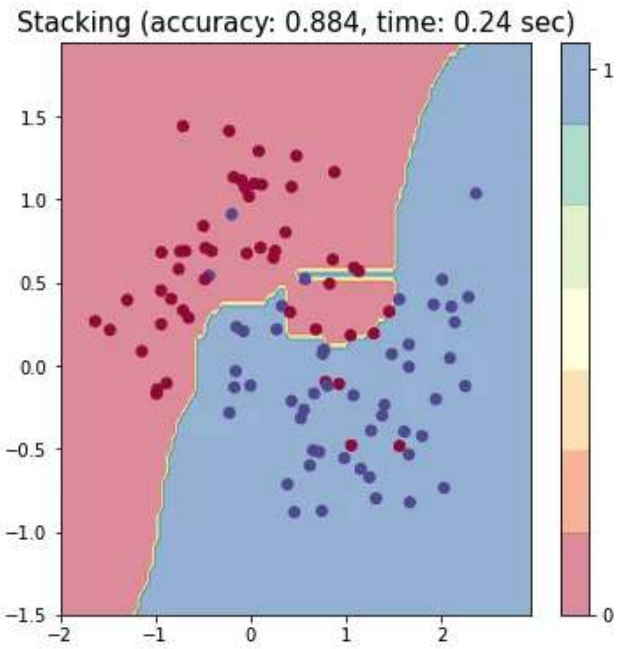

Figure 13. Accuracy and computational time for the stacking models. **Ref**: Image by author.

This ensemble model outperforms both hard voting and soft voting at the cost of increasing the total computational time.

2.3. Bagging

The accuracy and time for the bagging ensemble methods when using 100 estimators are:

- SVC: accuracy of 87.7% and runtime of 0.19 seconds
- Decision tree: accuracy of 89.3% and runtime of 0.16 seconds
- Logistic regression: accuracy of 84.1% and runtime of 0.32 seconds

```
1  clf_svc_bagging = BaggingClassifier(SVC(C=100000, gamma='scale', probability=True), n_estimators=10
2  clf_dt_bagging = BaggingClassifier(DecisionTreeClassifier(), n_estimators=100, max_samples=1.0)
3  clf_lr_bagging = BaggingClassifier(LogisticRegression(C=10000, solver='lbfgs'), n_estimators=100, m
```

ensemble_bagging hosted with ❤ by GitHub view raw

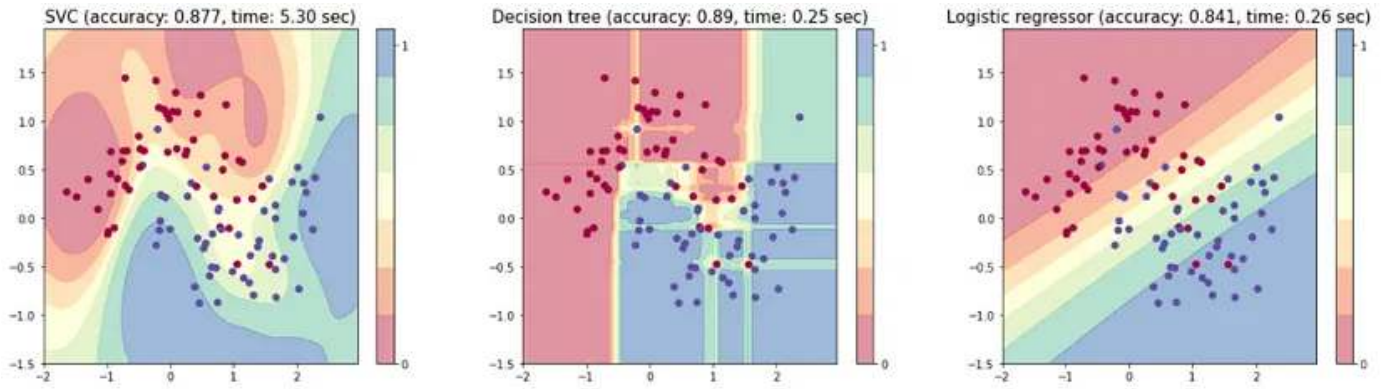

Figure 14. Accuracy and computational time for the SVC, decision tree, and logistic regressor bagging models. **Ref**: Image by author.

For logistic regression, the bagging method allows obtaining a slight non-linear model from combining multiple linear models. When compared with the benchmarks, the bagging methods outperform their respective models as they increase the accuracy. Specifically, a significant improvement can be observed for the decision tree, where bagging helps to reduce overfitting by voting over several trees that have different training sets.

One interesting topic to bring up here is the difference between Bagging and Random Forest, as both methods seem to be quite similar. As mentioned in [2, 3], the main difference relies on the number of features (*Important note*: Here we are speaking about the number of features, not the number of data) since only a subset of features are selected for the random forest. Watch the following video for further information about this topic.

```
clf_1 = RandomForestClassifier()
clf_2 = BaggingClassifier(DecisionTreeClassifier(), n_estimators=100, max_samples=1.0)
plot_results(clf_1, clf_2)
```

bagging_random_forest hosted with ❤ by GitHub view raw

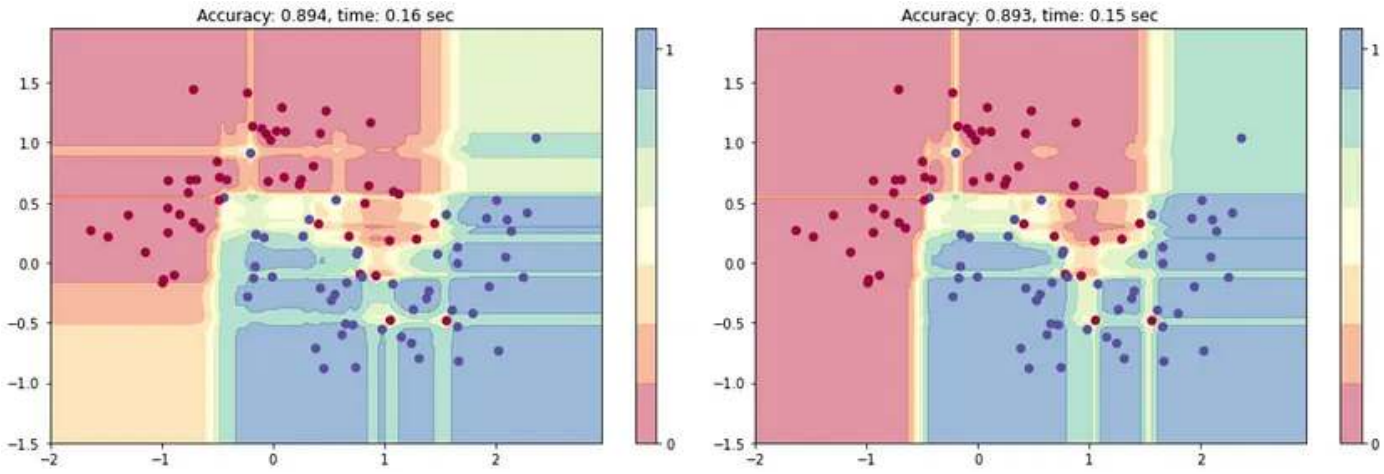

Figure 15. Accuracy and computational time for the random forest and decision tree bagging models. **Ref**: Image by author.

Lastly, here is an example of how to implement bagging on a neural network using PyTorch and Scikit-Learn's bagging wrapper. To do this, it is necessary to wrap the network in a sklearn.base.BaseEstimator object.

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(2, 64)
        self.l2 = nn.Linear(64, 64)
        self.l3 = nn.Linear(64, 64)
        self.l4 = nn.Linear(64, 2)
        self.optim = torch.optim.Adam(self.parameters(), lr=1e-2)

    def forward(self, x):
        z = F.leaky_relu(self.l1(x))
        z = F.leaky_relu(self.l2(z))
        z = F.leaky_relu(self.l3(z))
        z = F.log_softmax(self.l4(z),1)
        return z

class NetworkClassifier(BaseEstimator):
    def __init__(self):
        super().__init__()
        self.classes_ = np.array([0, 1])
        self.net = Net()

    def get_params(self, deep=True):
        return {}

    def set_params(self, **parameters):
        return self

    def fit(self, x, y, sample_weight = None):
        self.net = Net()
        idx = torch.arange(x.shape[0]).long()
        yy = torch.LongTensor(y)

        if sample_weight is not None:  # Useful for the AdaBoost
            w = torch.FloatTensor(sample_weight)
        else:
            w = torch.ones(x.shape[0])

        for i in range(150):
            self.net.optim.zero_grad()
            p = self.net.forward(torch.FloatTensor(x))
            loss = -torch.mean(w * p[idx, yy[idx]])
            loss.backward()
            self.net.optim.step()

```

```
46      def predict_proba(self, x):
47          p = np.exp(self.net.forward(torch.FloatTensor(x)).cpu().detach().numpy())
48          return p
```

```
1  clf_1 = NetworkClassifier()
2  clf_2 = BaggingClassifier(NetworkClassifier(), n_estimators=100, max_samples=1.0)
3  plot_results(clf_1, clf_2)
```

bagging_nn hosted with ❤ by GitHub view raw

```
54          return np.mean(p==y)
```

bagging_nn_network hosted with ❤ by GitHub view raw

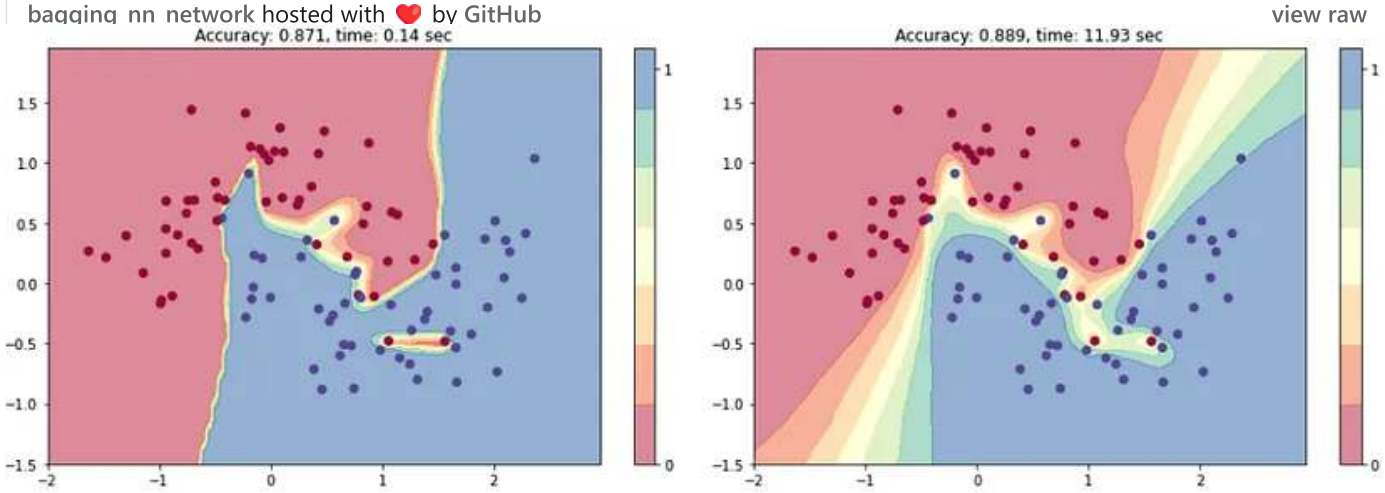

Figure 15. Accuracy and computational time for the neural network and neural network bagging models. **Ref**: Image by author.

2.4. Boosting

Lastly, the performance for the boosting models are:

- SVC: accuracy of 88.4% and runtime of 2.40 seconds
- Decision tree: accuracy of 87.2% and runtime of 0.02 seconds
- Logistic regression: accuracy of 83.7% and runtime of 0.15 seconds

For boosting, there are two main parameters that may severely affect the performance of the model. Firstly, the learning rate defines how aggressively the weights/coefficients change. Then, the number of estimators is the number of trees used in the model. The lower the learning rate, more trees are needed to train our model. Also, the lower the learning rate, the slower the model learns but the more robust and efficient it becomes. The learning rate and the number of estimators have been set to 0.1 and 50, respectively.

```
clf_svc_boosting = AdaBoostClassifier(SVC(C=100000, gamma='scale', probability=True), n_estimators=
clf_dt_boosting = AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=50, learning_rate = 0.1
clf_lr_boosting = AdaBoostClassifier(LogisticRegression(C=10000, solver='lbfgs'), n_estimators=50,
```

ensemble_boosting hosted with ❤ by GitHub view raw

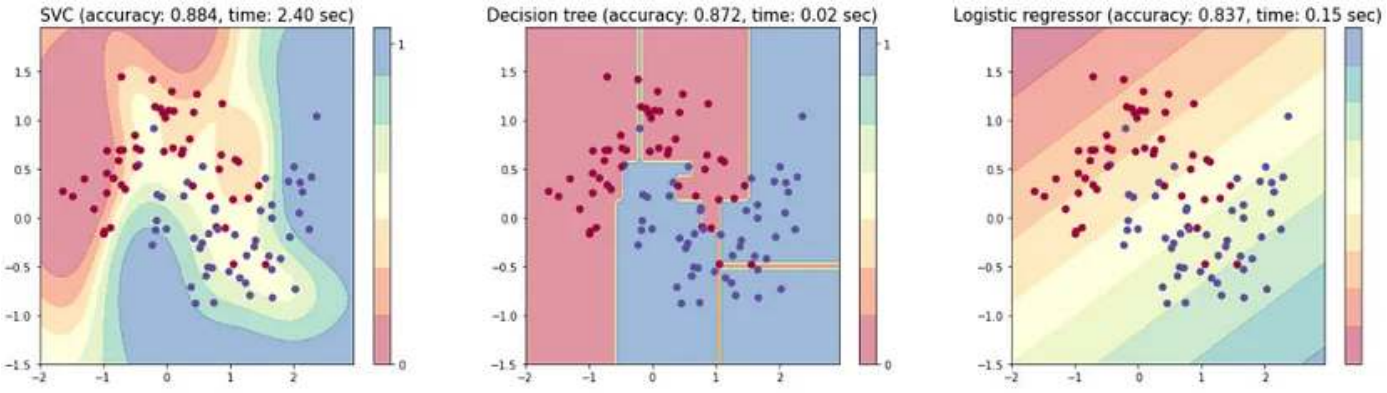

Figure 16. Accuracy and computational time for the SVC, decision tree, and logistic regressor boosting models. **Ref**: Image by author.

While results improved for the SVM classifier, they did not improve significantly for the decision tree, as it is still overfitting the training data. Nevertheless, this result changes when modifying the maximum depth of the decision tree as it solves the underfitting problem of the decision tree models.

```
clf_dt_1 = DecisionTreeClassifier(max_depth=1)
clf_dt_2 = DecisionTreeClassifier(max_depth=2)
clf_dt_3 = DecisionTreeClassifier()
```

ensemble_decision_trees_max_depth hosted with ❤ by GitHub view raw

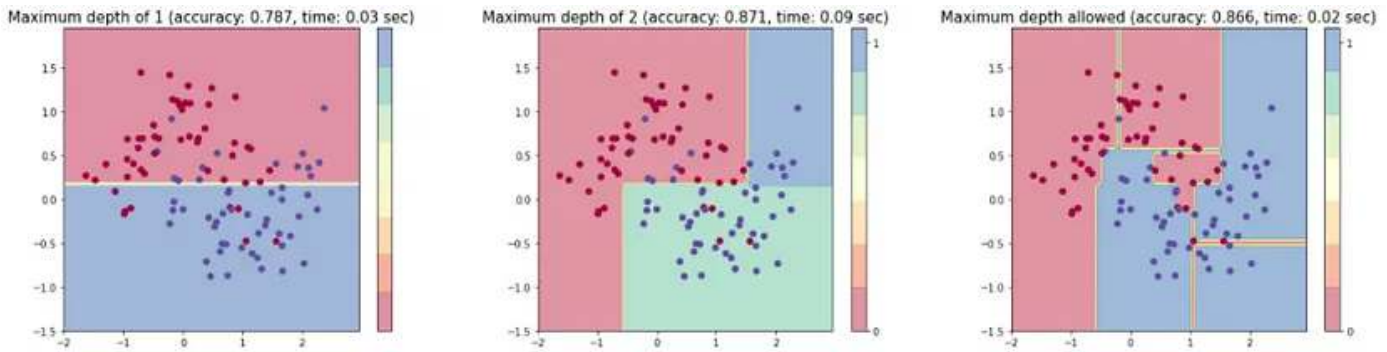

Figure 17. Accuracy and computational time for the decision tree models. **Ref**: Image by author.

```
clf_dt_boosting_1 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), n_estimators=50, learni
clf_dt_boosting_2 = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2), n_estimators=50, learni
clf_dt_boosting_3 = AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=50, learning_rate = 0
```

ensemble_boosting_decision_trees hosted with ❤ by GitHub view raw

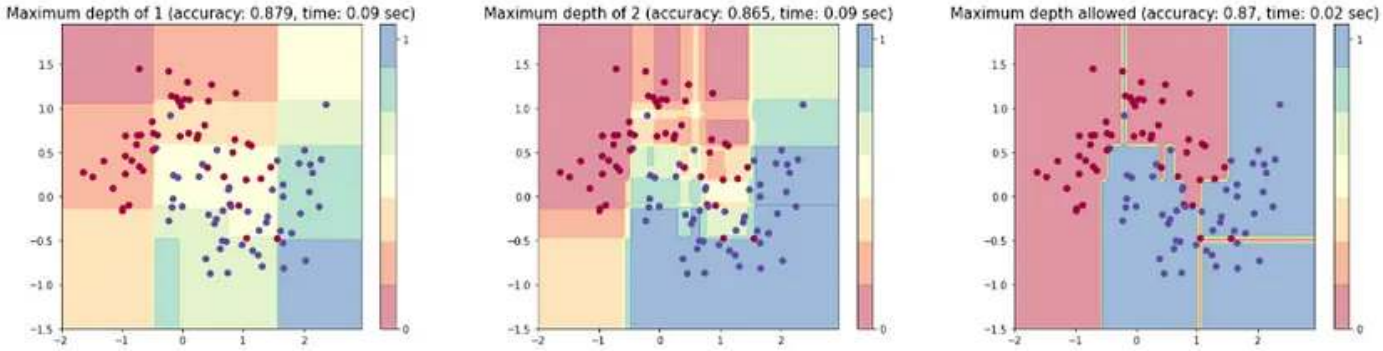

Figure 17. Accuracy and computational time for the decision tree boosting models. **Ref**: Image by author.

3. Summary

This article has introduced the four main ensemble methods (Voting, Stacking, Bagging, and Boosting), presenting their main advantages, disadvantages, and implementation. Here is a summary of the results so you can easily compare the results in terms of accuracy and computational cost for each of the methods.

Benchmark:

- SVM: 84.2% and 0.08 seconds
- Decision tree: 86.6% and 0.02 seconds
- Logistic regression: 83.5% and 0.02 seconds

Voting:

- Hard voting: 86.2% and 0.12 seconds
- Soft voting: 87.3% and 0.09 seconds

Stacking:

- 88.2% and 0.24 seconds

Bagging:

- SVC: 87.7% and 0.19 seconds
- Decision tree: 89.3% and 0.16 seconds
- Logistic regression: 84.1% and 0.32 seconds

Boosting:

- SVC: 88.4% and 2.4 seconds
- Decision tree: 87.2% and 0.02 seconds
- Logistic regression: 83.7% and 0.15 seconds

If you enjoyed this post, please consider subscribing. You'll get access to all of my content + every other article on Medium from awesome creators!

References

[1] Stack Exchange, What is the influence of C in SVMs with linear kernel?

[2] CSIAS, What is the difference between Bagging and Random forest?

[3] Stack Exchange, What is the difference between bagging and random forest if only one explanatory variable is used?

[4] YouTube, Bootstrapping main ideas

[5] Collins Ayuya, Engineering education.

[6] Medium, Ensemble methods: bagging, boosting, and stacking.

[7] CSIAS, What is the difference between Bagging and Random forest?

[8] Stack Exchange, What is the difference between bagging and random forest if only one explanatory variable is used?

[9] Medium, Stacking and blending intuitive explanation of advanced ensemble methods

Data Science Machine Learning Artificial Intelligence Ensemble Learning Model