Ensembles: Bagging vs Boosting

This lesson will focus on how to use boosting and bagging machine learning algorithms in Python.

WE'LL COVER THE FOLLOWING

Ensembles

Recall that in the lesson Random Forests, we learned that an **ensemble** is a collection of different predictive models that collectively decide the predicted output. Ensemble methods are divided into two categories:

- Bagging
- Boosting

Bagging

In **bagging**, each individual model randomly samples from the training data with replacement. This means each model is different.

Note that we do not train individual models on random subsets of the data; rather, they are trained on the whole data set, but each training example is randomly sampled with replacement. For instance if our training data has 6 numbers such as [1,2,3,4,5,6] and we sample 6 times with replacement, we might get [1,2,2,4,5,5]. Therefore each individual model is different.

In Bagging, the result is obtained by averaging the responses of the N models

or majority vote.

Boosting

In **boosting**, each individual model samples from the data, but this time, the sampling is not random, rather it is weighted. This means that in contrast to bagging, where each example in the training set has an equal chance of being chosen for training, in boosting, different samples have different chances of being chosen.

Therefore, individual models are trained on different training sets, and hence, different models are obtained.

In Boosting algorithms, each model is trained on data, considering the previous model's success. After each training step, the weights are redistributed. Misclassified data increases its weights to emphasize the most difficult examples. In this way, subsequent models will focus on them during their training.

Once all of the models give their output, the second stage is to decide the final output. the final output is again based on weights. Boosting assigns a second set of weights, this time for the N models, in order to take a weighted average of their predictions. In the Boosting training stage, the algorithm allocates weights to each resulting model. A model with good classification results on the training data will be assigned a higher weight than a model that performs poorly.

Two common boosting algorithms are:

- Gradient Boosting
- AdaBoost

These differ in their loss functions. AdaBoost can be considered a special case of Gradient Boosting with *exponential loss*. Whereas Gradient Boosting is more flexible as it uses the entropy loss we used with logistic regression. We will use both of these later in this lesson.

Bagging classifier in Python

We can easily use bagging methods for our predictions in Python. The model

is available in `sklearn.ensemble` as `BaggingClassifier`. We will be using the Default of Credit Card Clients dataset to make our predictions.

```
import pandas as pd
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report

df = pd.read_csv('credit_card_cleaned.csv')

# MAKE DATA
X = df.drop(columns = ['default.payment.next.month','MARRIAGE','GENDER'])
Y = df[['default.payment.next.month']]

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2, random_state = 3)

# MAKE MODEL
bag_model = BaggingClassifier(n_estimators = 30)
bag_model.fit(X_train,Y_train)

# CALCULATE AND PRINT RESULTS
preds = bag_model.predict(X_test)
acc = accuracy_score(y_true = Y_test,y_pred = preds)
print(acc)
print(classification_report(y_true = Y_test,y_pred = preds))
```

We import `BaggingClassifier` in **line 2**. After we read the data in **line 6**, we separate our target variable as `Y`. To split the data into training and test sets, we use the function `train_test_split`. We provide our inputs `X` and the labels `Y` to the function in **line 12**. We also provide the test set size as `test_size`. 0.2 implies that 20% data will be included in the testing set, while the remaining 80% will form the training set. The function outputs 4 items that we store in `X_train`, `X_test`, `Y_train`, `Y_test`.

In **line 15**, we initialize our Bagging classifier object just like we did in the last lesson. We call `BaggingClassifier` with `n_estimators=30`, which means 30 individual models will be used. Then in the next line, we call the `fit` function of the model. We provide the training examples and labels to the function. After this, we need to evaluate our model. Therefore, we use the `predict` function of the model in **line 19** to store predictions in `preds`. We give the testing inputs `X_test` to `predict` as an argument. We use `accuracy_score` function to measure the accuracy of the predictions. We print the accuracy with the classification report, which we obtained by using the `classification_report` function, in the last two lines.

We see that the model is approximately 81% accurate and the F1-score is 0.79 which is considered decent performance.

AdaBoost classifier in Python

The model is available in `sklearn.ensemble` as `AdaBoostClassifier`. We will be using the Default of Credit Card Clients dataset to make our predictions.

```
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report

df = pd.read_csv('credit_card_cleaned.csv')

# MAKE DATA
X = df.drop(columns = ['default.payment.next.month','MARRIAGE','GENDER'])
Y = df[['default.payment.next.month']]

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state=3)

# MAKE MODEL
model = AdaBoostClassifier(n_estimators = 30)
model.fit(X_train,Y_train)

# CALCULATE AND PRINT RESULTS
preds = model.predict(X_test)
acc = accuracy_score(y_true = Y_test,y_pred = preds)
print(acc)
print(classification_report(y_true = Y_test,y_pred = preds))
```

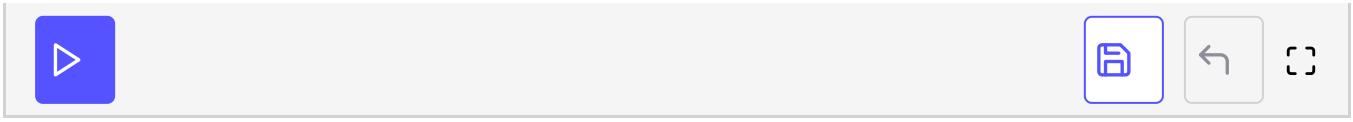
The code is the same as the one we used for bagging classifier. The only differences are in **line 2**, where we import `AdaBoostClassifier` and in **line 15**, where we call `AdaBoostClassifier` with `n_estimators = 30`.

The performance is almost the same as the bagging classifier.

Gradient Boosting classifier in Python

The model is available in `sklearn.ensemble` as `GradientBoostingClassifier`. We will be using the Default of Credit Card Clients dataset to make our predictions.

```
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report

df = pd.read_csv('credit_card_cleaned.csv')

# MAKE DATA
X = df.drop(columns = ['default.payment.next.month','MARRIAGE','GENDER'])
Y = df[['default.payment.next.month']]

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2, random_state = 3)

# MAKE MODEL
model = GradientBoostingClassifier(n_estimators = 30)
model.fit(X_train,Y_train)

# CALCULATE AND PRINT RESULTS
preds = model.predict(X_test)
acc = accuracy_score(y_true = Y_test,y_pred = preds)
print(acc)
print(classification_report(y_true = Y_test,y_pred = preds))
```



The code is the same as the one we used for the bagging classifier. The only differences are in **line 2**, where we import `GradientBoostingClassifier` and in **line 15**, where we call `GradientBoostingClassifier` with `n_estimators = 30`.

The performance is almost the same as the bagging and adaboost classifier.

A lot of other ensemble methods are also available in `sklearn`. Refer to the documentation to look at all of these methods.

So, until now, we have only seen supervised machine learning algorithms. In the next lesson, we will see some examples of unsupervised learning.