ensemble_introduction

May 29, 2021

1 Introductory example to ensemble models

This first notebook aims at emphasizing the benefit of ensemble methods over simple models (e.g. decision tree, linear model, etc.). Combining simple models result in more powerful and robust models with less hassle.

We will start by loading the california housing dataset. We recall that the goal in this dataset is to predict the median house value in some district in California based on demographic and geographic data.

Note

If you want a deeper overview regarding this dataset, you can refer to the Appendix - Datasets description section at the end of this MOOC.

[ ]:
```
from sklearn.datasets import fetch_california_housing

data, target = fetch_california_housing(as_frame=True, return_X_y=True)
target *= 100  # rescale the target in k$
```

We will check the statistical performance of decision tree regressor with default parameters.

[ ]:
```
from sklearn.model_selection import cross_validate
from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor(random_state=0)
cv_results = cross_validate(tree, data, target, n_jobs=-1)
scores = cv_results["test_score"]

print(f"R2 score obtained by cross-validation: "
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

We obtain fair results. However, as we previously presented in the “tree in depth” notebook, this model needs to be tuned to overcome over- or under-fitting. Indeed, the default parameters will not necessarily lead to an optimal decision tree. Instead of using the default value, we should search via cross-validation the optimal value of the important parameters such as `max_depth`, `min_samples_split`, or `min_samples_leaf`.

We recall that we need to tune these parameters, as decision trees tend to overfit the training data if we grow deep trees, but there are no rules on what each parameter should be set to. Thus, not making a search could lead us to have an underfitted or overfitted model.

Now, we make a grid-search to tune the hyperparameters that we mentioned earlier.

[ ]:
```
%%time
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

param_grid = {
    "max_depth": [5, 8, None],
    "min_samples_split": [2, 10, 30, 50],
    "min_samples_leaf": [0.01, 0.05, 0.1, 1]}
cv = 3

tree = GridSearchCV(DecisionTreeRegressor(random_state=0),
                    param_grid=param_grid, cv=cv, n_jobs=-1)
cv_results = cross_validate(tree, data, target, n_jobs=-1,
                            return_estimator=True)
scores = cv_results["test_score"]

print(f"R2 score obtained by cross-validation: "
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

We see that optimizing the hyperparameters will have a positive effect on the statistical performance. However, it comes with a higher computational cost.

We can create a dataframe storing the important information collected during the tuning of the parameters and investigate the results.

Now we will use an ensemble method called bagging. More details about this method will be discussed in the next section. In short, this method will use a base regressor (i.e. decision tree regressors) and will train several of them on a slightly modified version of the training set. Then, the predictions of all these base regressors will be combined by averaging.

Here, we will use 50 decision trees and check the fitting time as well as the statistical performance on the left-out testing data. It is important to note that we are not going to tune any parameter of the decision tree.

[ ]:
```
%%time
from sklearn.ensemble import BaggingRegressor

base_estimator = DecisionTreeRegressor(random_state=0)
bagging_regressor = BaggingRegressor(
    base_estimator=base_estimator, n_estimators=20, random_state=0)

cv_results = cross_validate(bagging_regressor, data, target, n_jobs=-1)
scores = cv_results["test_score"]

print(f"R2 score obtained by cross-validation: "
      f"{scores.mean():.3f} +/- {scores.std():.3f}")
```

Without searching for optimal hyperparameters, the overall statistical performance of the bagging

regressor is better than a single decision tree. In addition, the computational cost is reduced in comparison of seeking for the optimal hyperparameters.

This shows the motivation behind the use of an ensemble learner: it gives a relatively good baseline with decent statistical performance without any parameter tuning.

Now, we will discuss in detail two ensemble families: bagging and boosting:

- ensemble using bootstrap (e.g. bagging and random-forest);
- ensemble using boosting (e.g. adaptive boosting and gradient-boosting decision tree).