User Documentation

Decision Tree Classification with Bagging

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator, by introducing randomization into its construction procedure and then making an ensemble out of it. When random samples are drawn with replacement, then the method is known as Bagging.

This catalog item is a bagging (bootstrap) classifier which fits decision tree base classifiers.

Algorithm details

Decision Tree Classification

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules.

In decision analysis a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

Decision trees are commonly used in operations research and operations management. If in practice decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

Bagging (bootstrap)

In statistics and machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Unlike a statistical ensemble in statistical mechanics, which is usually infinite, a machine learning ensemble refers only to a concrete finite set of alternative models, but typically allows for much more flexible structure to exist among those alternatives.

When random subsets of the dataset are drawn as random subsets of the samples, then this

algorithm is known as Pasting. If samples are drawn with replacement, then the method is known as Bagging. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches.

For more details, please check the following links:

- https://en.wikipedia.org/wiki/Decision_tree
- https://en.wikipedia.org/wiki/Ensemble_learning
- https://en.wikipedia.org/wiki/Bootstrap_aggregating

Input Description

The input has two main components stored in JSON format.

```
{ "data": da  , "config": confi  }
```

data

"data" has two main parts. One of them is the "training" data set which is used to build out models. It contains the explanation variables and the labels. The other main part is the "predict" which contains only the explanation variables (input of models).

Example:

```
{
    "predict": {
        "explanation_variables": [
            [5.1, 3.5, 1.4, 0.2],
            [7.0, 3.2, 4.7, 1.4],
            [6.3, 3.3, 6.0, 2.5]
        ]
        },
    "training": {
        "explanation_variables": [
            [4.9, 3.0, 1.4, 0.2],
            [4.7, 3.2, 1.3, 0.2],
            ...
        ],
        "labels": [
            0,
            0,
            ...
        ]
    }
}
```

You can find the whole example dataset in the sample_input.json file.

Config

The config is also in JSON format and contains the parameters of the decision tree and the bagging algorithms.

Decision tree parameters

- **criterion** : The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.
- **splitter** : The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.
- **max_features** : The number of features to consider when looking for the best split.
- **max_depth** : The maximum depth of the tree.
- **min_samples_split** : The minimum number of samples required to split an internal node.
- **min_samples_leaf** : The minimum number of samples required to be at a leaf node.
- **min_weight_fraction_leaf** : The minimum weighted fraction of the input samples required to be at a leaf node.
- **max_leaf_nodes** : Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity.
- **class_weight** : Weights associated with classes in the form {class_label: weight}.
- **random_state** : It is the seed used by the random number generator.
- **presort** : Whether to presort the data to speed up the finding of best splits in fitting.
- **for more details**:http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.decision_path

Bagging parameters

- **n_estimators** : The number of base estimators in the ensemble.
- **max_samples** : The number of samples to draw from X to train each base estimator.
- **max_features** : The number of features to draw from X to train each base estimator.
- **bootstrap**: Whether samples are drawn with replacement.

- **bootstrap_features** : Whether features are drawn with replacement.
- **oob_score** : Whether to use out-of-bag samples to estimate the generalization error.
- **warm_start** : When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new ensemble.
- **n_jobs** : The number of jobs to run in parallel for both fit and predict.
- **random_state** : It is the seed used by the random number generator.
- **verbose**: Controls the verbosity of the building process.
- **for more details**: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Exampl**:**

```
{
    "decision_tree": {
        "presort": false,
        "splitter": "best",
        "max_leaf_nodes": null,
        "min_samples_leaf": 1,
        "min_samples_split": 2,
        "min_weight_fraction_leaf": 0,
        "criterion": "gini",
        "random_state": null,
        "min_impurity_split": 1e-07,
        "max_features": null,
        "max_depth": null,
        "class_weight": "balanced"
    },
    "bagging": {
        "warm_start": false,
        "max_samples": 1.0,
        "oob_score": false,
        "n_jobs": -1,
        "verbose": 0,
        "bootstrap": true,
        "n_estimators": 10,
        "random_state": null,
        "max_features": 1.0,
        "bootstrap_features": false
    }
}
```

You can find the whole example dataset in the sample_input.json file.

Outputs Description

The algorithm returns to the applied model results. In our example the first row (predict/explonation_variable) classified to class 0, the second one to class 1 and the last one to class 2.

Example test output

```
{
        "result": {"predicted_labels": [ 0, 1, 2]}
}
```

You can find the whole example dataset in the sample_output.json file.

Usage samples

Goal

This example will show how bagging (bootstrap) algorithm works what is the idea behind this approach. Moreover, we suppose that the user knows the decision tree algorithm. (If note please check the Wikipedia page: https://en.wikipedia.org/wiki/Decision_tree).

Our iris data set

We use the well-known iris data set which can be found in sklearn package as well. We use the last two columns of the data set (petal length, petal width) because of the easier visualization.

We create 3 subsets from narrowed iris dataset. We choose randomly 100 elements from the 150 items. One item can be chosen multiple times. We ensure the repeatability whit setting the random seed. We used 1111, 1234 and 6784 seeds.

.

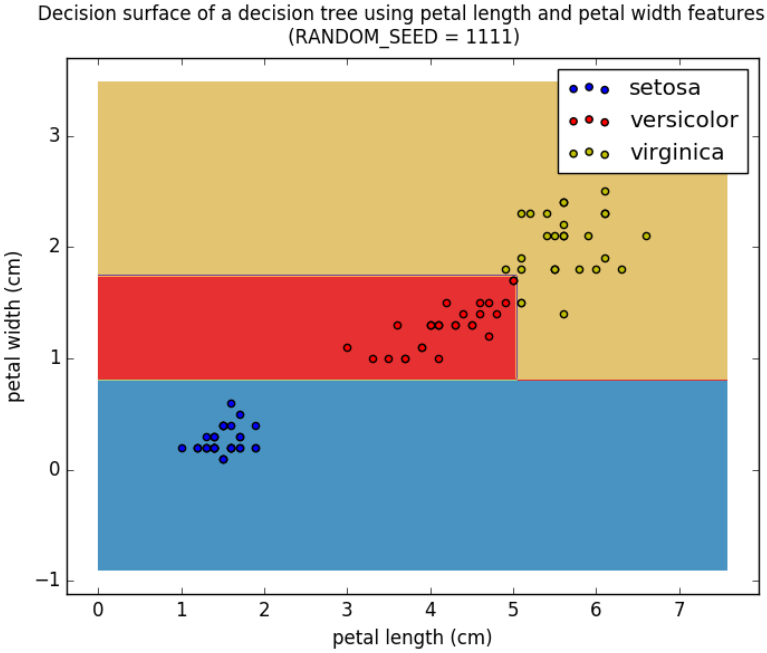
```
## generate the sub datasets
import numpy as np
from sklearn.datasets import load_iris

# Parameters
RANDOM_SEED = 1234

# Load data
iris = load_iris()

np.random.seed(RANDOM_SEED)
selected_rows = np.random.choice(150, 100)

# delete unused attributes
iris.data = np.delete(iris.data, [0,1], 1)
iris.feature_names = np.delete(iris.feature_names, [0,1])
```

```
# select items to built the model
iris.data = iris.data[selected_rows, :]
iris.target = iris.target[selected_rows]
```

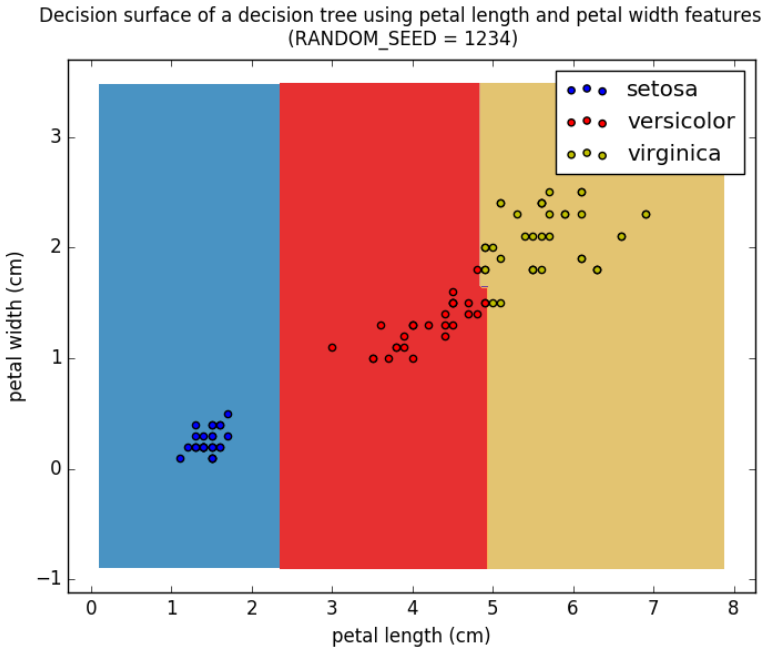
Creation of decision trees

We train different decision trees with these three sub sets. We use the following python code to train them. All the built trees are visualized below. During the training we used the default setting of DecisionTreeClassifier (Python, sklearn package).

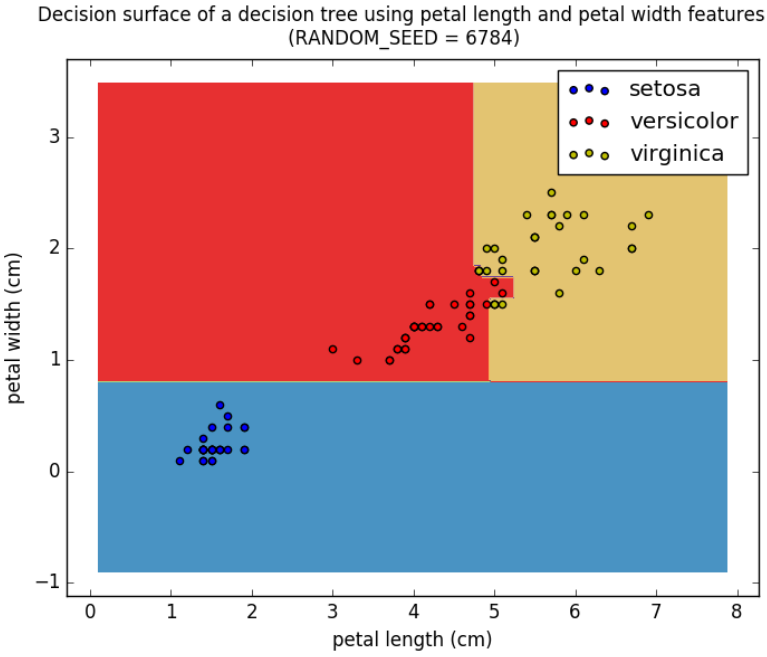
```
## build one decision tree
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier().fit(iris.data, iris.target)
```



1. figure Visualized decision tree when random seed is 1111



2. figure Visualized decision tree when random seed is 1234



3. figure Visualized decision tree when random seed is 6784

Evaluation

We have three different model, which divide the space differently as well, so the models could give different classification results to the same input. We establish the result with voting. Rise of model number can increase the accuracy.