Decision Tree

Algorithms

Create a decision tree by repeating grouping by finding the rule(decision) that minimizes the given metric(default gini). Pretty simple and easy.

Support **random forest**. This is an ensemble method in which a plurality of decision trees are randomly learned. Divide learning data randomly, learn each tree, make several weak classifiers, and give the final result through voting. *max_features* can limit the number of features that each tree will use when creating random forest.

Support various **metrics** such as gini, entropy, and classification error.

Various options to **prevent overfitting** are supported such as *depth*, *min_size*, *min_gain* and *max_features*. You can limit the *depth* of the tree to avoid overfitting the tree. Prevent the creation of small leaf nodes by entering the *min_size* at which the decision will be created. Also, by setting *min_gain*, you can avoid branching if you do not exceed the minimum information gain. This helps each tree to learn a different characteristic so it can get a better value.

Implementations

Python is a simple language that is easy enough to understand directly. So it's not difficult to see and understand the code right away. But here are tips for Python beginners.

dt.main

It consists of three major parts. **Load data**, **Process data** and **Write data**.

Load data using *pandas*, it's simple and effective. And change categorical data to numeric data(just encoding) using `pandas.factorize`. After that, split train, test data.

In Process data, just generate an instance of DecisionTree from `lib.tree.DecisionTree` and fit train data and predict with test data.

Write predicted results with decoded data from categorical data.

lib.tree.DecisionTree

There are two important functions such as `fit` and `predict`. `fit` receives training data and creates a decision tree. `predict` returns prediction of a created tree.

fit

Use `_build` function internally. The `_build` function creates a node and recursively creates left and right nodes. Each node has an index that determines which data to select and its value besides the left, right nodes. It is important how to set the index. The value that minimizes the calculated value of the metric from given training data is the index. This means that conditions can divide the group well. In the code(#L34-L36), `l`, `r` is divided group and `i` is an index(condition) to divide. After that, create left, right node by the divided group. If it does not meet the condition, create a terminal node that leaf node of the tree. Terminal nodes have the most detected class values. So If encountered a terminal node while prediction, the value is the predicted value.

predict

Apply `_predict` function to the value want to predict. Inside of `_predict`, just traversal nodes in the decision tree and return prediction.

lib.metric

Implement several metrics, `gini`, `information gain` and `gain ratio`. Type`--metric` to select metric.

Gini is the implementation of gini index. **Entropy** and **Classification Error** also implemented as below in `metric.py`.

$$Entropy(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$

$$I_E(t) = 1 - max\{p_i\}$$

$$G.I(A) = \sum_{i=1}^{d} \left(R_i \left(1 - \sum_{k=1}^{m} p_{ik}^2 \right) \right)$$

Requirements

- NumPy: is the fundamental package for scientific computing with Python.
- Pandas: is providing high-performance, easy-to-use data structures and data analysis tools for the Python.

install packages using pip

```
pip3 install -r requirements.txt
```

Tested @ python3.5 in Ubuntu 16.04 LTS, macOS High Sierra and Windows 10

Run as below

```
python3 dt.py (train_file_path) (test_file_path) (result_file_path) [--metric] [--depth] [--minsize] [--mingain] [--feature] [--forest]
```