Дерево решений и случайный лес

Дерево решений — логический алгоритм классификации, решающий задачи классификации и регрессии. Представляет собой объединение логических условий в структуру дерева.

Содержание

Дерево решений

Определение:

Дерево решений (англ. *decision tree, DT*) — алгоритм классификации $a(x) = (V_{\text{внутр}}, v_0, V_{\text{лист}}, S_v, \beta_v)$, задающийся деревом (связным ациклическим графом), где:

- $V = V_{\text{внутр}} \cup V_{\text{лист}}$ — множество вершин , $v_0 \in V$ — корень дерева;
- $S_v : D_v \to V_v$ — функция перехода по значению предиката в множество детей вершины v;
- $\beta_v : X \to D_v$ — предикат ветвления, $v \in V_{\text{внутр}}$ и $|D_v| < \infty$;
- Для листьев $v \in V_{\text{лист}}$ определена метка класса $y_v \in Y$.

Определение:

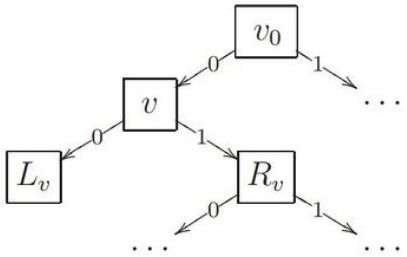
Бинарное дерево решений — частный случай дерева решений, для которого $D_v = \{0, 1\}$.

```
function classify(x):
  v = v_0
  if β_v(x) = 1
     v := R_v
  else
```

```
    v := L_v
return y_v
```

Информативность ветвления

Для того, чтобы оценивать качество разбиения объектов по предикату β, введем понятие *информационного выигрыша* разбиения.
Сначала оценим распределение значений классов объектов внутри каждого множества из разбиения, введя понятие *меры неопределенности распределения*.



Классификация объекта $x \in X$ бинарным решающим деревом

Определение:

Частотная оценка вероятности класса y в вершине $v \in V_{\text{внутр}}$:
$p_y = P(y|x \in U) = \frac{1}{|U|} \sum_{x_i \in U} [y_i = y]$

Определение:

Мера неопределенности (англ. *impurity*) распределения p_y:

- минимальна, когда $p_y \in \{0, 1\}$;
- максимальна, когда $p_y = \frac{1}{|Y|}$ для всех $y \in Y$;
- не зависит от перенумерации классов

$\Phi(U) = \sum_{y \in Y} p_y L(p_y) = \frac{1}{|U|} \sum_{x_i \in U} L(P(y_i|x_i \in U)) \to min,$
где $L(p)$ убывает и $L(1) = 0$, например: $-log_2(p), 1 - p, 1 - p^2$

Примерами мер неопределенности распределения являются:

- Энтропия: $\Phi(U) = -\sum\limits_{i}^{N} p_i log_2 p_i$, определяется для каждого множества из разбиения, N — количество возможных классов, и p_i — вероятность объекта принадлежать i-ому классу.
- Критерий Джини: $\Phi(U) = \sum_{i!=j} p_i p_j = \sum_i p_i * (1 - p_i)$, максимизацию этого критерия можно интерпретировать как максимизацию числа пар объектов одного класса, оказавшихся после разбиения в одном множестве.

Теперь определим суммарную *неопределенность распределения* в разбиении.

Определение:

Неопределенность распределения $P(y_i|x_i \in U_{\beta(x_i)})$ после ветвления вершины v по предикату β и разбиения $U = \bigcup_{k \in D_v} U_k$:
$\Phi(U_0, \ldots, U_{D_v}) = \frac{1}{|U|} \sum_{k \in D_v} \sum_{x_i \in U_k} L(P(y_i|x_i \in U_k)) = \sum_{k \in D_v} \frac{|U_k|}{|U|} \Phi(U_k)$

Информационный выигрыш от разбиения определяется как изменение неопределенности в системе.

Определение:

Информационный выигрыш от разбиения по предикату β

$$Gain(\beta, U) = \Phi(U) - \Phi(U_1, \ldots, U_{|D_v|}) = \Phi(U) - \sum_{k \in D_v} \frac{|U_k|}{|U|} \Phi(U_k) \to max_{\beta \in B}$$

Рекурсивный алгоритм построения бинарного дерева решений ID3

Покажем идею построения дерева решения на частном случае бинарного дерева. Алгоритм $ID3$ (англ. *Induction of Decision Tree*) заключается в последовательном дроблении выборки на две части до тех пор, пока в каждой части не окажутся объекты только одного класса. Разделение производится по предикату β, который выбирается из множества элементарных предикатов. На практике в качестве элементарных предикатов чаще всего берут простые пороговые условия вида $\beta(x) = [f_j(x) >= d_j]$.
Проще всего записать этот алгоритм в виде рекурсивной процедуры $ID3$, которая строит дерево по заданной подвыборке U и возвращает его корневую вершину.

```
1:function ID3(U):
2:    if forall u ∈ U: y_u = y, y ∈ Y
         // создать листовую вершину v с меткой класса y_v
3:       v = createLeafVertex(y_v)
4:       return v
      // найти предикат с максимальным информационным выигрышом
      β = arg max_{β∈B} Gain(β, U)
      // разбить выборку на две части U = U_0 ∪ U_1 по предикату β
5:    U_0 := {x ∈ U : β(x) = 0}
6:    U_1 := {x ∈ U : β(x) = 1}
7:    if U_0 = ∅ || U_1 = ∅
         // найти класс, в котором находится большинство объектов из U
8:       y_v = majorClass(U)
9:       v = createLeafVertex(y_v)
      else
         // создать внутреннюю вершину v
10:      v = createVertex()
11:      β_v = β
12:      S_0 = ID3(U_0)
13:      S_1 = ID3(U_1)
14:   return v
```

Редукция решающих деревьев

Суть редукции (англ. *pruning*) состоит в удалении поддеревьев, имеющих недостаточную статистическую надёжность. При этом дерево перестаёт безошибочно классифицировать обучающую выборку, зато качество классификации новых объектов, как правило, улучшается. Рассмотрим наиболее простые варианты редукции.

Предредукция

Предредукция (англ. *pre-pruning*) или критерий раннего останова досрочно прекращает дальнейшее ветвление в вершине дерева, если информативность $I(\beta, U)$ для всех возможных предикатов β не дотягивает до заданного порогового значения I_0.
Для этого на шаге 8 алгоритма $ID3$ условие $U_0 = \emptyset$ или $U_1 = \emptyset$ заменяется условием $I(\beta, U) <= I_0$. Порог I_0 является управляющим параметром метода.
Предредукция считается не самым эффективным способом избежать переобучения, так как жадное ветвление по-прежнему остаётся глобально неоптимальным. Более эффективной считается стратегия постредукции.

Постредукция

Постредукция (англ. *post-pruning*) просматривает все внутренние вершины дерева и заменяет отдельные вершины либо одной из дочерних вершин (при этом вторая дочерняя удаляется), либо терминальной вершиной. Процесс замен продолжается до тех пор, пока в дереве остаются вершины, удовлетворяющие критерию замены.

Критерием замены является сокращение числа ошибок на контрольной выборке, отобранной заранее, и не участвовавшей в обучении дерева. Стандартная рекомендация — оставлять в контроле около 30% объектов.

Для реализации постредукции контрольная выборка X^k пропускается через построенное дерево. При этом в каждой внутренней вершине v запоминается подмножество $S_v \subseteq X_k$ попавших в неё контрольных объектов. Если $S_v = \emptyset$, то вершина v считается ненадёжной и заменяется терминальной по *мажоритарному правилу*:
в качестве y_v берётся тот класс, объектов которого больше всего в обучающей подвыборке U, пришедшей в вершину v.
Затем для каждой внутренней вершины v вычисляется число ошибок, полученных при классификации выборки S_v следующими способами:

- $r(v)$ — классификация поддеревом, растущим из вершины v;
- $r_L(v)$ — классификация поддеревом левой дочерней вершины L_v;
- $r_R(v)$ — классификация поддеревом правой дочерней вершины R_v;
- $r_c(v)$ — отнесение всех объектов выборки S_v к классу $y \in Y$.

Эти величины сравниваются, и в зависимости от того, какая из них оказалась минимальной, принимается, соответственно, одно из четырёх решений:

- сохранить поддерево вершины v;
- заменить поддерево вершины v поддеревом левой дочерней вершины L_v;
- заменить поддерево вершины v поддеревом правой дочерней вершины R_v;
- заменить поддерево v терминальной вершиной класса $y_v = \arg\min_{y \in Y} r_c(v)$.

Алгоритмы построения деревьев решения

Недостатки рассмотренного алгоритма ID3:

- Применим только для дискретных значений признаков;
- Переобучение;
- На каждом шаге решение принимается по одному атрибуту.

Алгоритм CART (https://en.wikipedia.org/wiki/Predictive_analytics#Classification_and_regression_trees_.28CART.29) (англ. *Classification And Regression Trees*)

- В отличие от ID3 работает и с непрерывными значениями признаков: на каждом шаге построения дерева последовательно сравнивает все возможные разбиения для всех атрибутов и выбирает наилучший атрибут и наилучшее разбиение для него. Разбивает объекты на две части;
- Использует редукцию для избежания переобучения;
- Обрабатывает пропущенные или аномальные значения признаков.

Алгоритм C4.5 (https://en.wikipedia.org/wiki/C4.5_algorithm)

- Также работает и с непрерывными значениями признаков: на каждом шаге построения дерева выбирает правило разбиения по одному из признаков. Разбивает объекты на несколько частей по этому правилу, рекурсивно запускается из полученных подмножеств;
- Использует редукцию для избежания переобучения;
- Обрабатывает пропущенные или аномальные значения признаков.

Случайный лес

Случайный лес — один из примеров объединения классификаторов в ансамбль.
Алгоритм построения случайного леса, состоящего из N деревьев на основе обучающей выборки X такой:

```
for (n: 1,...,N):
   // сгенерировать выборку X_n с помощью бутстрэпа
   X_n = bootstrap(X)
   // построить решающее дерево t_n по выборке X_n
   t_n = ID3(X_n)
```

Итоговый классификатор — $a(x) = \frac{1}{N}\sum\limits_{i=1}^{N} t_i(x)$. Для задачи классификации мы выбираем решение по большинству результатов, выданных классификаторами, а в задаче регрессии — по их среднему значению.

Таким образом, случайный лес — бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков.

Примеры кода

Примеры на языке Python

- Для решения задач классификации и регрессии используют DecisionTreeClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier), DecisionTreeRegressor (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.DecisionTreeRegressor);
- В **sklearn.ensemble** также представлены методы классификации, основанные на ансамблях, в том числе: бэггинг (https://scikit-learn.org/stable/modules/ensemble.html#bagging) и случайный лес (https://scikit-learn.org/stable/modules/ensemble.html#forest), которые были описаны выше.

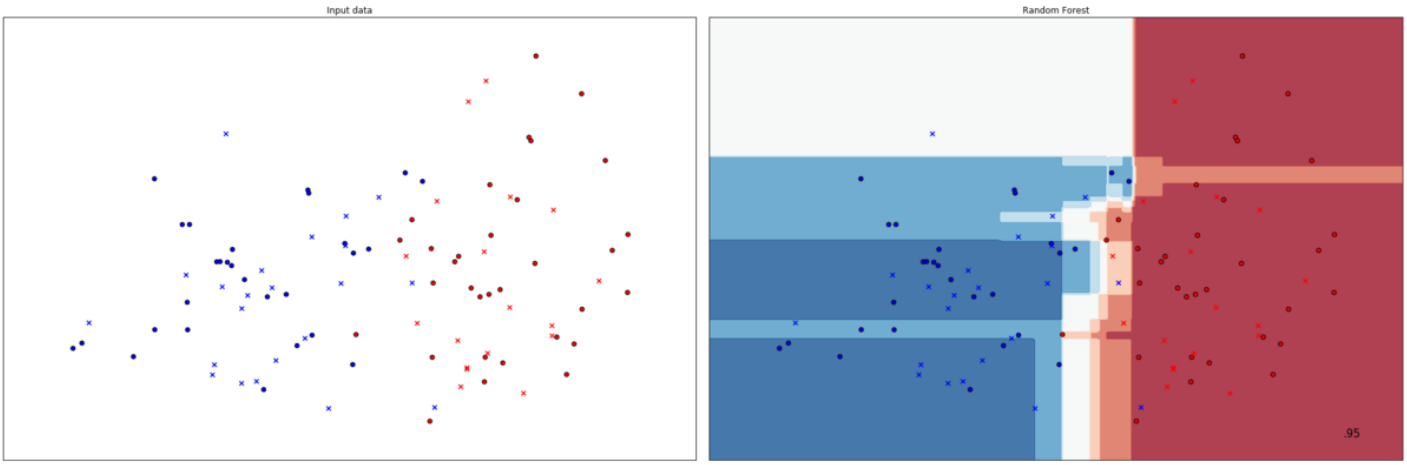
Так, в этом примере создается бэггинг ансамбль из классификаторов **KNeighborsClassifier**, каждый из которых обучен на случайных подмножествах из 50% объектов из обучающей выборки, и 50% случайно выбранных признаков.

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
bagging = BaggingClassifier(KNeighborsClassifier(), max_samples=0.5, max_features=0.5)
```

Пример использования классификатора на случайном лесе: Полную версию кода можно найти здесь (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-download-auto-examples-classification-plot-classifier-comparison-py%7C)

```
from sklearn import RandomForestClassifier
from sklearn.datasets import make_classification
// сгенерируем случайную обучающую выборку с классификацией по n_classes классам
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1, n_classes=2)
// разбиваем выборку на обучающую и тестовую
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, random_state=42)
// создадим классификатор на случайном лесе, состоящим из n_estimators деревьев
RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1)
clf.fit(X_train, y_train)
score = clf.score(X_test, y_test)
```

Результат классификации показан на рисунке.



Классификация RandomForestClassifier. Кружочками изображены объекты обучающей выборки, крестиками тестовой выборки. Справа цветом выделены границы принятия решений, в правом нижнем углу — значение accuracy.

Пример на языке Scala

SBT зависимость:

```
libraryDependencies += "com.github.haifengl" %% "smile-scala" % "1.5.2"
```

Пример классификации датасета и вычисления F1 меры[1] используя smile.classification.cart[2]:

```
import smile.classification._
import smile.data._
import smile.plot._
import smile.read
import smile.validation.FMeasure
```

```
val iris: AttributeDataset = read.table("iris.csv", delimiter = ",", response = Some((new NumericAttribute("class"), 2)))
val x: Array[Array[Double]] = iris.x()
val y: Array[Int] = iris.y().map(_.toInt)
val dt: DecisionTree = cart(x, y, 1000)
val predictions: Array[Int] = x.map(dt.predict)
val f1Score = new FMeasure().measure(predictions, y)
plot(x, y, dt)
```

Пример на языке Java

Пример классификации с применением `weka.classifiers.trees.RandomForest`[3]

`Maven` зависимость:

```
<dependency>
  <groupId>nz.ac.waikato.cms.weka</groupId>
  <artifactId>weka-stable</artifactId>
  <version>3.8.0</version>
</dependency>
```

```
import weka.classifiers.evaluation.Evaluation;
import weka.classifiers.trees.RandomForest;
```

```
// read dataset
var trainingDataSet = getDataSet(...);
var testingDataSet = getDataSet(...);
// create random forest classifier
var forest = new RandomForest();
forest.setMaxDepth(15);
forest.setNumFeatures(2);
forest.buildClassifier(trainingDataSet);
// evaluate the model on test dataset and print summary
var eval = new Evaluation(trainingDataSet);
```

```
eval.evaluateModel(forest, testingDataSet);
System.out.println(eval.toSummaryString());
```

Пример на языке R

Деревья решений

Для создания деревьев решений используется функция `ctree()` из пакета `party`.

```
# importing package
install.packages("party")

# reading data
rdata <- read.csv("input.csv", sep = ',', header = FALSE)

# evaluating model
output.tree <- ctree(target ~ x + y + z, data = rdata)

# plotting results
plot(output.tree)
```

Случайный лес

Для создания случайного леса необходимо импортировать пакет `randomForest`

```
# importing packages
install.packages("party")
install.packages("randomForest")

# reading data
rdata <- read.csv("input.csv", sep = ',', header = FALSE)

# creating the forest
output.forest <- randomForest(target ~ x + y + z, data = rdata)

# getting results
print(output.forest)
```

См. также

- Виды ансамблей

Источники информации

1. Логические алгоритмы классификации (http://www.machinelearning.ru/wiki/images/3/3e/Voron-ML-Logic.pdf) — Лекция К. В. Воронцова
2. Случайный лес (https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-5-ensembles-of-algorithms-and-random-forest-8e05246cbba7) — статья на Medium, Yury Kashnitskiy
3. Деревья решений (https://scikit-learn.org/stable/modules/tree.html) — scikit-learn.org
4. Ансамбли классификаторов (https://scikit-learn.org/stable/modules/ensemble.html) — scikit-learn.org.
5. F1 мера (https://en.wikipedia.org/wiki/F1_score)
6. Smile, Decision Trees (https://haifengl.github.io/smile/classification.html#cart)
7. Weka, Random Forest (http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html)

- Эта страница последний раз была отредактирована 4 сентября 2022 в 19:22.