Деревья решений

***Деревья решений*** – алгоритм обучения с учителем, который используется в задачах классификации данных. Алгоритм основан на построении древовидной структуры на этапе обучения и использования этой структуры на этапе классификации.

 Для начала используем небольшой датасет, описывающий склонность к диабету в зависимости от возраста и индекса массы тела.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data = [[31,45,'yes'],[29,47,'no'],[27,55,'no'],[35,53,'yes'],[28,51,'no'],[40,50,'yes'],[39,41,'no']]

df = pd.DataFrame(data, columns=['Bmi','Age','Diabetes'])

print(df)



x=df.iloc[:,:2].to\_numpy()

y=df.iloc[:,2].to\_numpy()

plt.figure(figsize=(7,5))

for i in range(x.shape[0]):

 plt.scatter(x[i,0], x[i,1], c=('blue' if y[i]=='no' else 'red'), marker='\*', s=200)

plt.xlabel('Bmi')

plt.ylabel('Age')

plt.show()



from sklearn import tree

from sklearn import inspection

model = tree.DecisionTreeClassifier(criterion="entropy").fit(x, y)

tree.plot\_tree(model)

plt.show()



inspection.DecisionBoundaryDisplay.from\_estimator(model, x, response\_method="predict", cmap=plt.cm.RdYlBu)

for i in range(x.shape[0]):

 plt.scatter(x[i,0], x[i,1], c=('blue' if y[i]=='no' else 'red'), marker='\*', s=200)

plt.xlabel('Bmi')

plt.ylabel('Age')

plt.show()



 ***Критерий разбиения на поддеревья.***

Индекс Джини:

$$Gini\left(S\right)=1-\sum\_{i}^{}P\_{i}^{2}$$

$S$ – множество объектов;

$i$ – индекс класса в множестве;

$P\_{i}$ – вероятность, доля объектов $i$-го класса в множестве $S$;

Энтропия:

$$H\left(S\right)= -\sum\_{∪S\_{i}=S}^{}P\_{i}∙log\_{2}P\_{i}$$

$$H\left(S\right)≈2\*Gini(S)$$

Информационный выигрыш:

$$Gain\left(S,A\right)=H\left(S\right)-\sum\_{v\in A}^{}\frac{\left|S\_{v}\right|}{\left|S\right|}H(S\_{v})$$

S – Множество точек данных

H(S) – Энтропия набора данных

А – Атрибут, признак

$v$ – Возможные значения A

$S\_{v}$ – Подмножество набора данных, ограниченное значением атрибута $v\in A$

Конкретно, для искомого набора данных



получаем:

класс ‘no’ – $P\_{0}= \frac{4}{7}$

класс ‘yes’ – $P\_{1}= \frac{3}{7}$

$$H\left(S\right)=-\frac{4}{7}log\_{2}\left(\frac{4}{7}\right)-\frac{3}{7}log\_{2}\left(\frac{3}{7}\right)=0.985$$

Разобьём на два подмножества по признаку Bmi<=30.

$$S\_{0}= S\left(Bmi\leq 30\right)=\left\{1,2,4\right\}, 3-^{'}no^{'}, 0-'yes'$$

$$S\_{1}= S\left(Bmi>30\right)=\left\{0,3,5,6\right\}, 1-^{'}no^{'}, 3-'yes'$$

$$H\left(S\_{0}\right)=0$$

$$H\left(S\_{1}\right)=-\frac{1}{4}log\_{2}\left(\frac{1}{4}\right)-\frac{3}{4}log\_{2}\left(\frac{3}{4}\right)=0.811$$

$$Gain\left(S,Bmi\right)=0.985-\left(\frac{3}{7}∙0+\frac{4}{7}∙0.811\right)=0.522$$

Разобьём на два подмножества по признаку Age<=50.

$$S\_{0}= S\left(Age\leq 50\right)=\left\{0,1,5,6\right\}, 2-^{'}no^{'}, 2-'yes'$$

$$S\_{1}= S\left(Age>50\right)=\left\{2,3,4\right\}, 2-^{'}no^{'}, 1-'yes'$$

$$H\left(S\_{0}\right)=-\frac{1}{2}log\_{2}\left(\frac{1}{2}\right)-\frac{1}{2}log\_{2}\left(\frac{1}{2}\right)=1.0$$

- плохое разбиение, так как в подмножестве $S\_{0}$ получаем $H\left(S\_{0}\right)=1$ полностью неопределённое состояние.

Разобьём признаку Age<=45.

$$S\_{0}= S\left(Age\leq 45\right)=\left\{0,6\right\}, 1-^{'}no^{'}, 1-'yes'$$

$$S\_{1}= S\left(Age>45\right)=\left\{1,2,3,4,5\right\}, 3-^{'}no^{'}, 2-'yes'$$

$$H\left(S\_{0}\right)=-\frac{1}{2}log\_{2}\left(\frac{1}{2}\right)-\frac{1}{2}log\_{2}\left(\frac{1}{2}\right)=1.0$$

$$H\left(S\_{1}\right)=-\frac{3}{5}log\_{2}\left(\frac{3}{5}\right)-\frac{2}{5}log\_{2}\left(\frac{2}{5}\right)=0.972$$

$$Gain\left(S,Age\right)=0.985-\left(\frac{2}{7}∙1+\frac{5}{7}∙0.972\right)=0.005$$

Выбираем разбиение $Bmi\leq 30$, имеющее высокий информационный выигрыш.

Продолжаем аналогичную процедуру разбиения для полученных подмножеств до достижения необходимой точности или глубины дерева. См. результат в приведённом примере программы:



 Алгоритм построения дерева решений:

1. На основе множества $S$ создаём узел дерева.
2. Завершаем процесс при достижении заданных глубины дерева или распределения элементов по классам.
3. Разбиваем множество $S$ на подмножества по критерию, связанному с атрибутами элементов множества. Выбираем лучшее по информационному выигрышу разбиение.
4. Создаём поддеревья и для каждого подмножества повторяем процесс с 1-го пункта.

Пример на реальных данных.

***Ирисы Фишера:***

from sklearn.datasets import load\_iris

iris = load\_iris()

import matplotlib.pyplot as plt

import numpy as np

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

from sklearn import tree

#from sklearn import inspection

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.tree import DecisionTreeClassifier

X = PCA(n\_components=2).fit\_transform(iris.data)

y = iris.target

#model = DecisionTreeClassifier().fit(X, y)

model = DecisionTreeClassifier(max\_depth=4, min\_samples\_leaf=5).fit(X, y)

plt.figure(figsize=(9,7))

tree.plot\_tree(model)

plt.show()



DecisionBoundaryDisplay.from\_estimator(model, X, response\_method="predict", cmap=plt.cm.RdYlBu)

colors = ['blue', 'green', 'purple', 'brown', 'black']

for i in range(X.shape[0]):

 plt.scatter(X[i,0], X[i,1], c=colors[y[i]], marker='o', s=40)

plt.show()



Дерево «без ограничений»

model = DecisionTreeClassifier().fit(X, y)





 ***Датасет Mushroms***:

# Decision Trees - Mushroom

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read\_csv('../mushrooms.csv')

X = dataset.drop(['class'],axis=1)

y = dataset['class']

X = pd.get\_dummies(X)

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

y = encoder.fit\_transform(y)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=1)

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

model = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=0).fit(X\_train, y\_train)

plt.figure(figsize=(12,8))

tree.plot\_tree(model)

plt.show()



y\_pred = model.predict(X\_test)

from sklearn.metrics import accuracy\_score

print('Model accuracy score: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))



 Для наглядности результатов используем PCA:

# Decision Trees - Mushroom

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read\_csv('../mushrooms.csv')

X = dataset.drop(['class'],axis=1)

y = dataset['class']

X = pd.get\_dummies(X)

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

y = encoder.fit\_transform(y)

# ========== PCA ==================

from sklearn.decomposition import PCA

X = PCA(n\_components=2).fit\_transform(X)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.inspection import DecisionBoundaryDisplay

model = DecisionTreeClassifier(criterion='gini', max\_depth=3, random\_state=0).fit(X\_train, y\_train)

plt.figure(figsize=(12,8))

tree.plot\_tree(model)

plt.show()



y\_pred = model.predict(X\_test)

DecisionBoundaryDisplay.from\_estimator(model, X, response\_method="predict", cmap=plt.cm.RdYlBu)

colors = ['blue', 'green', 'purple', 'brown', 'black']

for i in range(X.shape[0]):

 plt.scatter(X[i,0], X[i,1], c=colors[y[i]], marker='o')

plt.show()



from sklearn.metrics import accuracy\_score

print('Model accuracy score: {0:0.4f}'. format(accuracy\_score(y\_test, y\_pred)))



***Дерево решений и случайный лес.***

Случайный лес – совокупность деревьев решений на случайных выборках из исходного набора данных и атрибутов.

Позволяет избежать преобучения и сильной зависимости от «шума» в данных.