به نام خدا

درخت تصمیم در مسائلی کاربرد دارد که بتوان آنها را به صورتی مطرح نمود که پاسخ واحدی به صورت نام یک دسته یا کلاس ارائه دهند.

داده های انتخاب شده برای این موضوع داده های مربوط به ویژگی ها یک تصویر دیجیتالی از توده سینه محاسبه می شود و ویژگی های هسته های سلولی موجود در تصویر را توصیف می کنند و از سرور https://www.kaggle.com/uciml/breast-cancer-wisconsin-data در دسترس هستند .

پس از دانلود داده ها با استفاده از پایتون و پکیج پاندا داده های موجود را در دیتا فرم می ریزیم :

Feature	Type	ویژگی
ID number	Integer	شماره شناسایی بیمار
Diagnosis	String	M = بدخیم, B = خوشخیم
radius	Float	شعاع (میانگین فاصله از مرکز به نقاط در محیط)
texture	Float	بافت (انحراف استاندارد مقادیر مقیاس خاکستری)
perimeter	Float	محیط
area	Float	مساحت
smoothness	Float	صافی (تنوع محلی در طول شعاع)
compactness	Float	فشردگی (محیط^2 / منطقه - 1.0)
concavity	Float	تقعر (شدت بخش های مقعر کانتور)
concave points	Float	نقاط مقعر (تعداد بخش های مقعر کانتور)
symmetry	Float	تقارن
fractal dimension	Float	ابعاد فراکتال ("خط ساحلی تقریب" - 1)

داده های موجود حاوی اطلاعات 569 بیمار که 357 نفرشان دارای توده خوش خیم و 212 مورد نیز توده بدخیم تشخیص داده شده اند که دارای 32 ستون مربوط به ویژگی هایی که در جدول ذکر شد که از هر کدام سه مقدار (به جز " ID number ", " Diagnosis ") وجود دارند که به ترتیب میانگین، خطای استاندارد و بزرگترین سلول سرطانی را نمایش می دهد ما در این مجموعه داده قصد داریم با استفاده از درخت تصمیم بد خیم یا خوش خیم بودن توده سرطانی را با استفاده از اطلاعات موجود تشخیص داده و اعتبار سنجی کنیم.

```
In [1]: import numpy as np
   ... : import pandas as pd

In [2]: data = pd.read_csv('breast_concer_wisconsin_(diagnostic)_dataset.csv')

In [3]: data.columns
Out[3]:
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
import numpy as np
import pandas as pd
data = pd.read_csv('breast_concer_wisconsin_(diagnostic)_dataset.csv')
data.columns
```

همانطور که از دستورات بالا مشاهده می کنید یکی از ستونها بدون نام است (ستون "Unnamed: 32") با مشاهده مقادیر این ستون متوجه می شویم که ستون مورد نظر خالی می باشد و هیچ اطلاعاتی ندارد پس می توانیم آن را حذف کنیم برای این کار از دستور زیر کمک می گیریم :

```
In [4]: data['Unnamed: 32']
Out[4]:
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
       ..
564   NaN
565   NaN
566   NaN
567   NaN
568   NaN
Name: Unnamed: 32, Length: 569, dtype: float64

In [5]: data.drop('Unnamed: 32', axis = 1, inplace = True)
```

```
data['Unnamed: 32']
data.drop('Unnamed: 32', axis = 1, inplace = True)
```

برای استفاده از الگوریتم درخت تصمیم میبایست ابتدا داده های رشته ای را به صورت صحیح به داده های عددی تبدیل کنیم که در این مجموعه داده فقط یک ستون وجوددارد که دارای مقادیر رشته ای است، برای تبدیل ابتدا یک ستون به نام " malignant " ایجاد کرده و با استفاده از دستور "where" از کتابخانه "numpy" مقادیر ستون جدید را ایجاد می کنیم، در اخر نیز ستون "diagnostis" را حذف می کنیم :

```
In [6]: data['malignant'] = np.where(data['diagnosis'] == 'M', 1, 0)

In [7]: data = data.drop('diagnosis', axis = 1)
```

```
data['malignant'] = np.where(data['diagnosis'] == 'M', 1, 0)
data = data.drop('diagnosis', axis = 1)
```

حال داده های ما برای استفاده از درخت تصمیم آماده است، فقط یک نکته دیگر مانده که باید داده های ما بالانس باشند تا نتیجه گیری در داده های آموزش و آزمایش یادگیری دقیق نمی باشد برای این کار با دو دستور زیر درصد خوشخیم و بدخیم بودن را متوجه می شویم :

```
In [8]: len(data[data['malignant'] == 1])/ len(data)
Out[8]: 0.37258347978910367

In [9]: len(data[data['malignant'] == 0])/ len(data)
Out[9]: 0.6274165202108963

In [10]: len(data[data['malignant'] == 1])
Out[10]: 212

In [11]: len(data[data['malignant'] == 0])
Out[11]: 357
```

```
len(data[data['malignant'] == 1])/ len(data)
len(data[data['malignant'] == 0])/ len(data)
len(data[data['malignant'] == 1])
len(data[data['malignant'] == 0])
```

برای بالانس کردن داده ها فقط کافی است که از بین داده های خوش خیم 146 مورد را حذف کنیم ما در دستور زیر این کار را به صورت تصادفی انجام داده ایم :

```
In [12]: data.drop(data[data['malignant'] == 0].sample(frac = 1).index[:146], inplace = True)

In [13]: len(data[data['malignant'] == 1])/ len(data)
Out[13]: 0.5011820330969267

In [14]: len(data[data['malignant'] == 0])/ len(data)
Out[14]: 0.4988179669030733
```

```
data.drop(data[data['malignant'] == 0].sample(frac = 1).index[:146], inplace = True)
len(data[data['malignant'] == 1])/ len(data)
len(data[data['malignant'] == 0])/ len(data)
```

حال داده ها بالانس شده و اماده استفاده از درخت تصمیم هستند .

ابتدا دو کتابخانه مورد نیاز را فرا خوانی می کنیم :

```
In [15]: from sklearn import tree
    ... : from sklearn.model_selection import cross_val_score
```

```
from sklearn import tree
from sklearn.model_selection import cross_val_score
```

حال با استفاده از دستور زیر داده ها را به صورت تصادفی کنار هم میچینیم تا داده های آموزش و تست را از بین انها انتخاب کنیم :

```
In [16]: data = data.sample(frac = 1)

In [17]: data_train = data[:400]

In [18]: data_test = data[400:]
```

```
data = data.sample(frac = 1)
data_train = data[:400]
data_test = data[400:]
```

در گام بعدی ستون " malignant " را هم از داده های آموزش و هم از داده های تست جدا می کنیم :

```
In [19]: data_train_M = data_train['malignant']

In [20]: data_test_M = data_test['malignant']

In [21]: data_train_a = data_train.drop('malignant', axis = 1)

In [22]: data_test_a = data_test.drop('malignant', axis = 1)
```

```
data_train_M = data_train['malignant']
data_test_M = data_test['malignant']
data_train_a = data_train.drop('malignant', axis = 1)
data_test_a = data_test.drop('malignant', axis = 1)
```

حال مدل خود را با عمق 5 ایجاد کرده و برای داده های آموزش خود برازش می کنیم :

```
In [23]: model = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth= 5)

In [24]: model = model.fit(data_train_a, data_train_M)
```

```
model = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth= 5)
model = model.fit(data_train_a, data_train_M)
```

حال به اعتبار سنجی می پردازیم برای این کار از دستور "cross_val_score" استفاده می کنیم :

```
In [36]: data_a = data.drop('malignant', axis = 1)
    ...: data_M = data['malignant']
    ...: scoresall = cross_val_score(model, data_a, data_M, cv = 7)
    ...: print('Total Accuracy : %.2f (+/- %.2f)' % (scoresall.mean(), scoresall.std() * 2))
Total Accuracy : 0.92 (+/- 0.06)
```

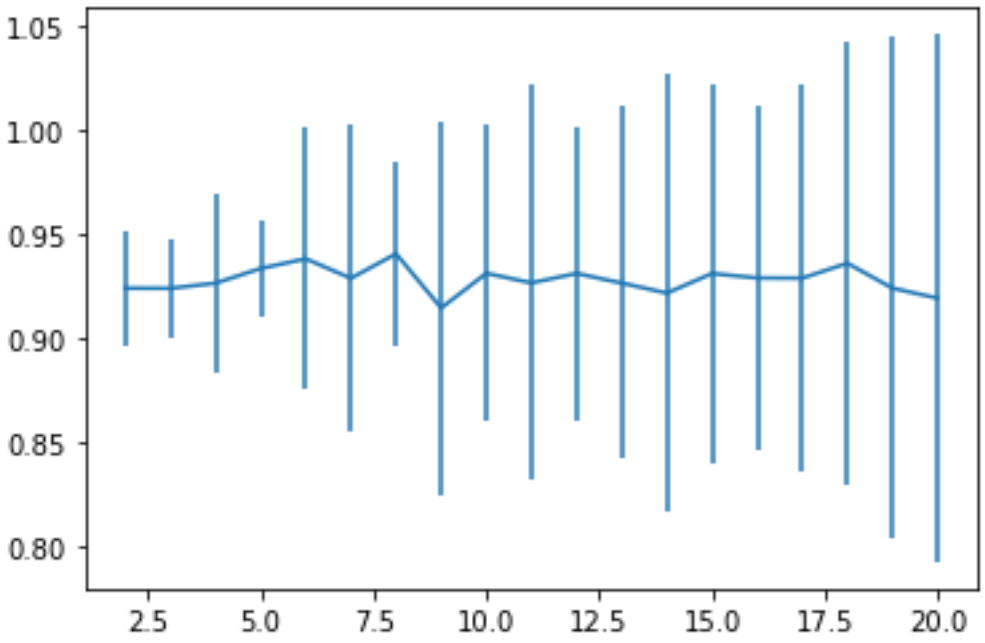
```
data_a = data.drop('malignant', axis = 1)
data_M = data['malignant']
scoresall = cross_val_score(model, data_a, data_M, cv = 7)
print('Total Accuracy : %.2f (+/- %.2f)' % (scoresall.mean(), scoresall.std() * 2))
```

عدد 7 نشان دهنده این است که هر بار داده ها را به 7 قسمت مساوی تقسیم کرده و هر بار یک قسمت را بعنوان داده تست و مابقی را بعنوان داده آموزش در نظر می گیرد .

در آخر با استفاده از دستور "for" تعداد مختلف عمق درخت را برای داده ها امتحان کرده و نتایج را در یک نمودار "errorbar" نمایش می دهیم :

```
In [34]: import matplotlib.pyplot as plt

In [35]: data_acc = np.empty((19,3), float)
    ...: i = 0
    ...: for max_depth in range(2, 21) :
    ...:     model = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth= max_depth)
    ...:     score = cross_val_score(model, data_a, data_M, cv = max_depth)
    ...:     data_acc[i, 0] = max_depth
    ...:     data_acc[i, 1] = score.mean()
    ...:     data_acc[i, 2] = score.std() * 2
    ...:     i += 1
    ...: data_acc
    ...: plt.errorbar(data_acc[:, 0], data_acc[:, 1], yerr = data_acc[:, 2])
Out[35]: <ErrorbarContainer object of 3 artists>
```




```
import matplotlib.pyplot as plt
data_acc = np.empty((19,3), float)
i = 0
for max_depth in range(2, 21) :
    model = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth= max_depth)
    score = cross_val_score(model, data_a, data_M, cv = max_depth)
    data_acc[i, 0] = max_depth
    data_acc[i, 1] = score.mean()
    data_acc[i, 2] = score.std() * 2
    i += 1
data_acc
plt.errorbar(data_acc[:, 0], data_acc[:, 1], yerr = data_acc[:, 2])
```