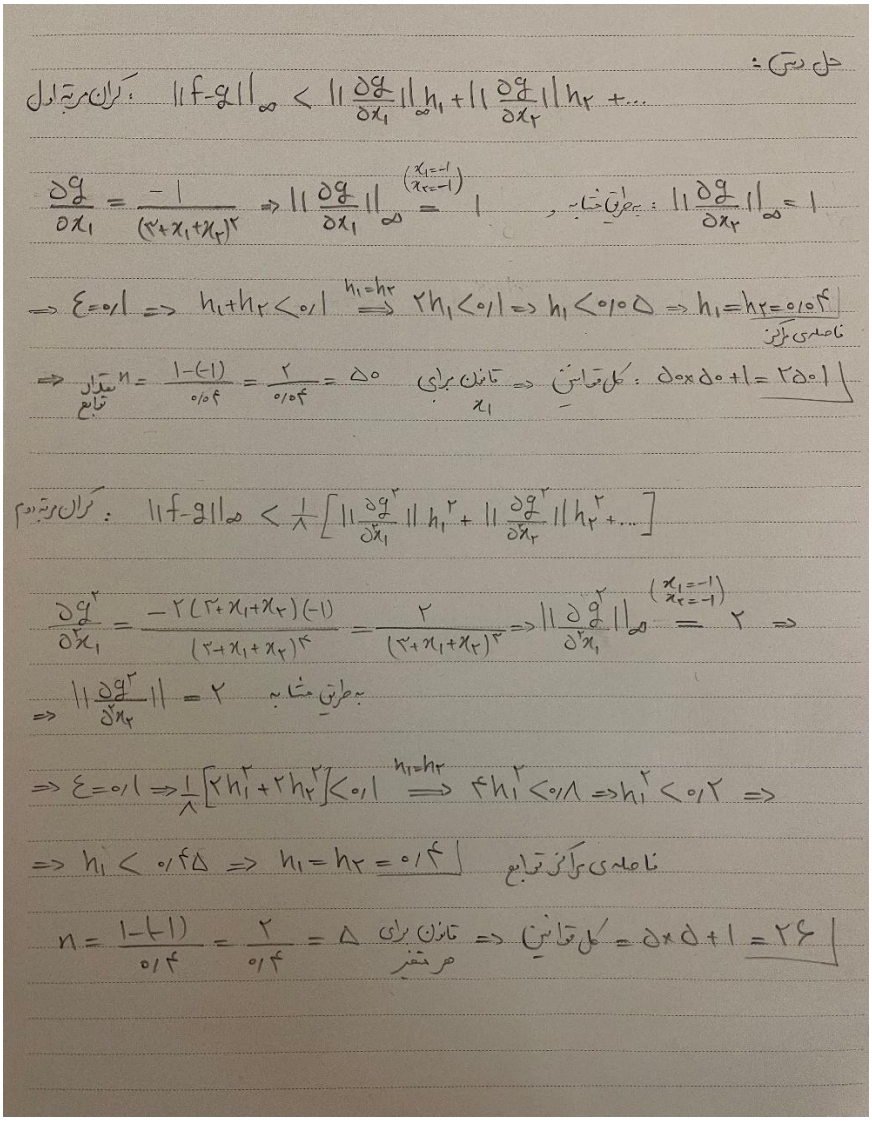
به نام خدا

گزارش مینی پروژه ۳

امیر ارشام بهشتی طهرانی
شماره دانشجویی:۹۸۲۰۲۹۳

مبانی سیستم های هوشمند

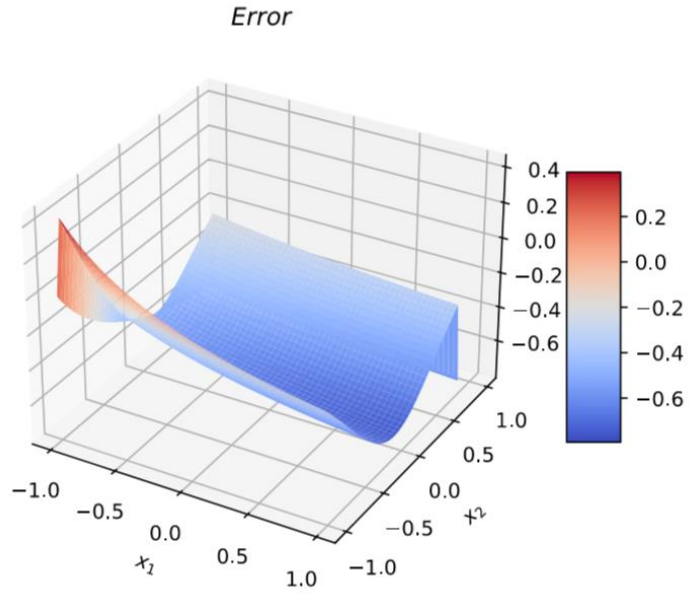
سوال۱) حل دستی:



با توجه به حل دستی به کمک نتایج به دست آمده و محتوای آموزشی تهیه شده، سیستم فازی با تقریب کران مرتبه اول و دوم را تشکیل داده و تابع خطای آن در بازه تعیین شده را رسم می‌کنیم.

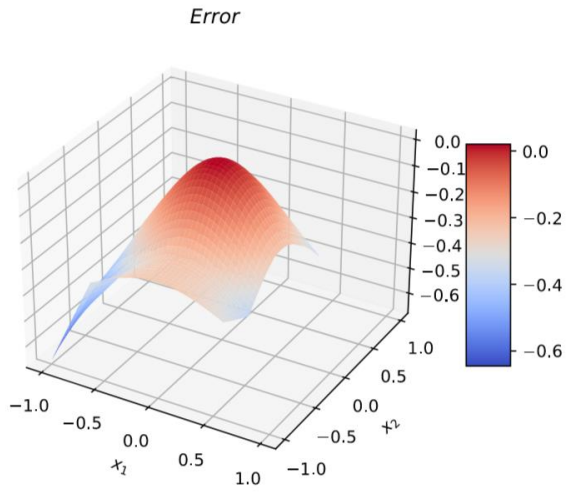
تقریب کران مرتبه اول:

برای این منظور در تقریب کران مرتبه اول، باید h = 0.04 و N = 50 قرار داده شود. در بخش تابع نیز g(x) را برابر با رابطه مورد نظر یعنی $\frac{1}{3+x_1+x_2}$ قرار می دهیم.



تقریب کران مرتبه دوم:

طبق نتایج به دست آمده در حل دستی، $h = 0.4$ و $N = 5$ قرار می دهیم و برنامه را مانند قبل اجرا می کنیم.



مشاهده می‌شود که در حالت دوم طبق محاسبات در حل دستی، با برآورده کردن دقت مورد نظر ۰/۱، تعداد توابع تعلق به وضوح کمتری (۲۶ عدد) در مقایسه با حالت اول (۲۵۰۱ عدد) نیاز است.

سوال ۴)

قسمت ۱:

ابتدا مشابه قبل پس از فراخوانی کتابخانه های مورد نظر، با دستور gdown داده ها را در محیط colab قرار می دهیم. سپس با دستور read_csv داده ها را داخل دیتافریم panda در متغیر df قرار می دهیم.

	Fever	Cough	Breathing issues	Infected
0	No	No	No	No
1	Yes	Yes	Yes	Yes
2	Yes	Yes	No	No
3	Yes	No	Yes	Yes
4	Yes	Yes	Yes	Yes
5	No	Yes	No	No
6	Yes	No	Yes	Yes
7	Yes	No	Yes	Yes
8	No	Yes	Yes	Yes
9	Yes	Yes	No	Yes
10	No	Yes	No	No
11	No	Yes	Yes	Yes
12	No	Yes	Yes	No
13	Yes	Yes	No	No

داده ها مربوط به ویژگی های افراد مشکوک به کرونا هستند که دارای ۳ ویژگی تب (Fever) ، سرفه (Cough) و مشکلات تنفسی (Breathing issues) هستند و در ستون آخر Infected نشان از ابتلا به کرونا است که خروجی و کلاس بندی مدنظر درخت تصمیم است.

برای طراحی درخت تصمیم نیاز داریم تا در هر مرحله و گره با توجه به ویژگی ها داده ها را تقسیم بندی کنیم. برای انتخاب بهترین ویژگی، باید به میزان جداسازی داده ها با توجه به انتخاب آن ویژگی توجه کنیم. برای این کار از آنتروپی و بهره اطلاعات (information gain) کمک می گیریم. آنتروپی داده ها میزان شلختگی و تنوع داده ها را نشان می دهد به این صورت که هر چه بیشتر باشد داده ها، تفکیک پذیری داده ها مشکل تر است در نتیجه به آنتروپی پایین نیاز داریم. اما معیار بهتر جداسازی، بهره اطلاعات است. با توجه به رابطه آن که از آنتروپی کل، آنتروپی ویژگی مورد نظر را کم می کنیم، در حقیقت میزان جداسازی داده ها با توجه به انتخاب یک ویژگی مشخص می شود. در نتیجه هر چه مقدار آن بالاتر باشد، نشان دهنده جداسازی بهتر داده هاست.

پس برای کد نویسی از ابتدا درخت تصمیم ابتدا باید معیار انتخاب ویژگی ها را تعریف کنیم که این کار با آنتروپی و بهره اطلاعات انجام می‌شود.

برای آنتروپی باید تعداد هر نوع برچسب (label) خروجی را بر تعداد کل تقسیم کنیم و سپس مجموع حاصل ضرب آن را در log مبنای دو آن طبق رابطه آنتروپی انجام دهیم. این کار را در تابع entropy انجام داده که ورودی آن برچسب ها و خروجی آن مقدار تابع آنتروپی است. برای محاسبه تعداد برپسب ها نیز از value_counts(). استفاده کرده‌ایم.

برای محاسبه بهره اطلاعات، نیز طبق رابطه آن عمل می کنیم. ابتدا انتروپی کل برچسب ها را محاسبه کرده و آنتروپی ویژگی مورد نظر را که با نام entropy_child مشخص شده، از آن طبق رابطه بهره اطلاعات (با در نطر گرفتن تعداد نمونه ها به عنوان وزن ها) کم می کنیم.

در ادامه نیاز داریم تا زیرگروه تشکیل شده، شامل ویژگی استفاده شده نباشد که این کار را با تعریف subset انجام داده‌ایم.

```
IG_Fever: 0.12808527889139443
IG_Cough: 0.0391486719030707
IG_Breathing_Issues: 0.39603884492804464
```

با توجه به بهره اطلاعات های محاسبه شده انتظار داریم تا Breathing_Issues در اولین گره درخت برای تصمیم گیری قرار گیرد.

در ادامه طبق محتوای آموزشی، درخت را از ابتدا کدنویسی می کنیم. برای گره های تصمیم گیری درخت یک کلاس به نام Node تعریف کرده و مشخصه های آن شامل feature و label را تعریف می کنیم. یک متد children هم برای قرار دادن گره های بعدی داخل همین کره تعریف می کنیم.

با تابع repr هم نحوه نمایش ازلاعات خروجی درخت را به صورت مورد نظر تبدیل می کنیم. به این صورت که درخت هر گره را با Feature استفاده شده از آن و سایر گره های زیرمجموعه اش نشان دهد.

در نهایت با تابع بازگشتی make_tree سازوکار تشکیل درخت را تکمیل می کنیم.

ابتدای هر مرحله اجرای درخت باید بررسی کنیم که آیا درخت شامل گره برگ (Leaf node) شده است یا خیر زیرا در این صورت به انتهای ان شاخه رسیده ایم و درخت توانایی تصمیم گیری دارد. برای این کار ابتدا به صورت کد زیر بررسی می کنیم که آیا تعداد برپسب ها در زیرمجموعه آن گره درخت به صفر رسیده یا اینکه تعداد ویژگه های استفاده شده در درخت به پایان رسیده است یا خیر که در صورت رخداد یکی از این دو شرط به گره برگ یا پیش بینی رسیده ایم.

```
if (len(data[target].unique()) == 1 or len(data.columns) == 1):
  return Node(label = data[target].iloc[0])
```

سپس اگر شرط رخ نداده بود باید گره های تصمیم گیری را تشکیل دهیم. برای این کار نیز از تابع بهره اطلاعات و آنتروپی کمک می گیریم تا با توجه به آن ها گره ها ایجاد شوند.

پس از محاسبه IG، با روش Greedy Search، index ویژگی با بیشترین IG را انتخاب کرده و آن ویژگی را برای گره انتخاب می کنیم. سپس داخل این گره همان طور که گفته شد یک

subset تشکیل داده و تصمیم گیری را به همین صورت با ویژگی های باقی مانده انجام می دهیم تا به گره پیش بینی برسیم.

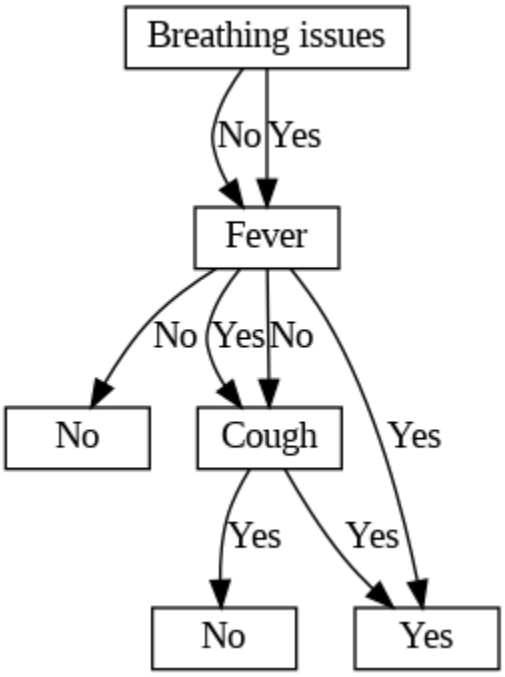
نکته مهم در این تابع این است که ابتدا هر شاخه درخت را ان قدر ادامه می دهیم تا به گره پیش‌بینی برسیم و سپس به مرحله قبلی برمی‌گردیم و همین عمل را انجام می دهیم تا در هایت درخت کامل شود.

در انتها درخت را با ورودی df به عنوان دیتافریم شامل داده ها و ’Infected‘ به عنوان خروجی و برچسب های داده ها تشکیل می دهیم.

نمای توضیحی درخت با توجه به تابع repr استفاده شده به صورت زیر است.

```
DecisionNode(feature="Breathing issues", children={'No': DecisionNode(feature="Fever", children={'No': LeafNode(label="No"), 'Yes': DecisionNode(feature="Cough", children={'Yes': LeafNode(label="No")})}), 'Yes': DecisionNode(feature="Fever", children={'Yes': LeafNode(label="Yes"), 'No': DecisionNode(feature="Cough", children={'Yes': LeafNode(label="Yes")})})})
```

برای نمایش درخت از کد آماده تهیه شده در محتوای آموزشی استفاده می کنیم. اما به نظر کد نیاز به اصلاح با توجه به درخت تصمیم این مسئله دارد.



تحلیل منطقی درخت تصمیم:

با توجه به تابع repr و نمایش زیر مجموعه دیتافریم در هر گره تصمیم که در خروجی کد در قسمت ساختن درخت نمایش داده می شود می توان درخت را تحلیل کرد.

ابتدا طبق انتظار ما ‘Breathing issues’ به عنوان گره تصمیم ابتدایی به توجه به بیشتر بودن IG آن انتخاب شده است. طبق دیرمجموعه در صورت خروجی ‘No’ ، subset به صورت زیر می‌شود.

	Fever	Cough	Infected
0	No	No	No
2	Yes	Yes	No
5	No	Yes	No
9	Yes	Yes	Yes
10	No	Yes	No
13	Yes	Yes	No

در این صورت در گره بعد ‘Fever’ انتخاب شده است که در صورت خروجی ‘No’ ، تمام برپسب های باقی مانده ‘No’ هستند و به یک گره پیش بینی رسیده‌ایم و نیازی نیست که ادامه دهیم.

	Cough	Infected
0	No	No
5	Yes	No
10	Yes	No

در ادامه نیز تمام حالت های ویژگی ‘Cough’ ، ‘yes’ هستند و نیازی به ادامه درخت نیست چون حالتی برای ‘No’ با توجه به داده ها وجود ندارد. ولی با توجه به کد نوشته شده درخت یک گره تصمیم با ‘Cough’ می سازد که فقط حالت ‘yes’ آن تکمیل می‌شود و خروجی به صورت زیر می شود.

	Infected
2	No
9	Yes
13	No

برای حالت ’Yes‘ در گره ابتدایی زیرمجموعه به صورت زیر می شود.

	Fever	Cough	Infected
1	Yes	Yes	Yes
3	Yes	No	Yes
4	Yes	Yes	Yes
6	Yes	No	Yes
7	Yes	No	Yes
8	No	Yes	Yes
11	No	Yes	Yes
12	No	Yes	No

مانند قبل ویژگی بعدی ’Fever‘ بوده که برای ’Yes‘ آن، با توجه به برپسب ها به گره پیش‌بینی می رسیم که به صورت زیر است.

	Cough	Infected
1	Yes	Yes
3	No	Yes
4	Yes	Yes
6	No	Yes
7	No	Yes

در حالت دیگر هم با اینکه برچسب ها یکی هستند، ولی گره برای ‘Cough’ هم ادامه پیدا می‌کند و خروجی به صورت زیر تنها در حالت ‘Yes’ برای ویژگی ‘Cough’ نشان داده می شود.

	Cough	Infected
8	Yes	Yes
11	Yes	Yes
12	Yes	No

	Infected
8	Yes
11	Yes
12	No

به علت کم بودن تعداد داده ها امکان تقسیم بهدی و استفاده از داده های ارزیابی وجود نداشت

قسمت ۲:

برای این سوال دیتاست Drugs را انتخاب می کنیم. مشابه قبل آن را با دستور gdown در colab بارگذاری کرده و با read_csv آن را داخل دیتافریم df قرار می دهیم.

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY
...	...	...	...	...	...	...
195	56	F	LOW	HIGH	11.567	drugC
196	16	M	LOW	HIGH	12.006	drugC
197	52	M	NORMAL	HIGH	9.894	drugX
198	23	M	NORMAL	NORMAL	14.020	drugX
199	40	F	LOW	NORMAL	11.349	drugX

200 rows × 6 columns

دیتاست شامل ۲۰۰ نمونه در سطر هاست و دارای ۶ ستون که ۵ ستون اول آن ویژگی ها (Features) هستند و ستون آخر مربوط به داروی مورد نظر و خروجی است. این دیتاست برای

کلاس بندی دارو ها به کار می رود. دقیق تر اینکه با توجه به ویژگی ها کدام دارو مد نظر ما برای رفع مشکل است.

برای این مسئله از دستورات اماده sci-kit learn مربوط به درخت تصمیم استفاده می کنیم. با توجه به این دستور داده همگی باید از نوع عدد باشند در نتیجه باید با استفاده از one-hot encoding، داده ها با نام را به صورت عددی درآوریم.

برای خروجی ’Drug‘ باید به هر دارو یک عدد نسبت دهیم چون این ستون تنها شامل خروجی است و با روش قبل تعداد ستون ها به ازای هر حالت دارو ها افزایش می یابد که مدنظر ما نیست. برای این کار از متد replace(). استفاده می کنیم و به ترتیب از ۰ تا ۴ به ’DrugA‘ تا ’DrugY‘ نسبت می دهیم.

سپس با متد pop(). ابتدا ستون ’Drug‘ را حذف کرده و سپس با insert(). آن را به عنوان آخرین ستون به دیتافریم df2 اضافه می کنیم و در ادامه با آن کار می کنیم.

```
df1 = pd.get_dummies(df, columns=['Sex', 'BP', 'Cholesterol'], drop_first = True)
df2 = df1.replace({'Drug' : {'drugA': 0, 'drugB': 1, 'drugC': 2, 'drugX': 3, 'drugY':4 }})

col = df2.pop('Drug')
df2.insert(len(df2.columns), 'Drug', col)
df2
```

	Age	Na_to_K	Sex_M	BP_LOW	BP_NORMAL	Cholesterol_NORMAL	Drug
0	23	25.355	0	0	0	0	4
1	47	13.093	1	1	0	0	2
2	47	10.114	1	1	0	0	2
3	28	7.798	0	0	1	0	3
4	61	18.043	0	1	0	0	4
...	...	...	...	...	...	...	...
195	56	11.567	0	1	0	0	2
196	16	12.006	1	1	0	0	2
197	52	9.894	1	0	1	0	3
198	23	14.020	1	0	1	1	3
199	40	11.349	0	1	0	1	3

200 rows × 7 columns

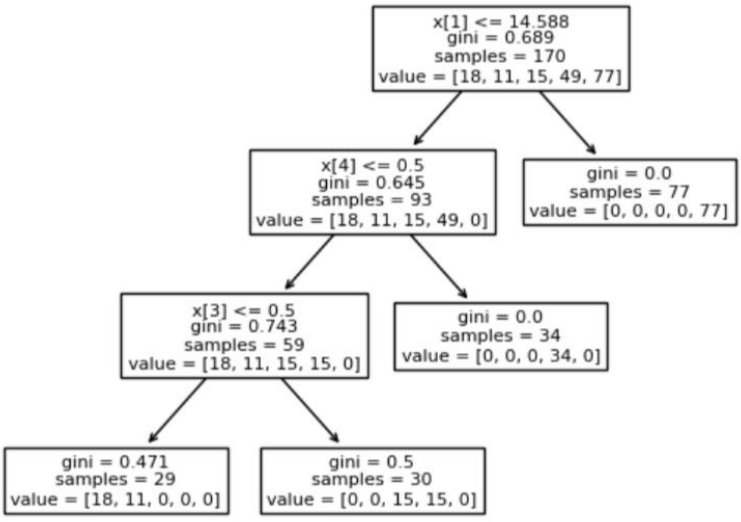
سپس داده ها را با متد iloc(). به جز ستون آخر داخل X قرار داده و داده های ستون آخر را به عنوان خروجی در Y قرار می دهیم. به کمک train_test_split و با نسبت ۸۵ به ۱۵ درصد داده ها را داخل داده های آموزشی و ارزیابی قرار می دهیم.

در ادامه مدل را با دستور آماده tree.DecisionTreeClassifier تعریف می کنیم. برای اولین مدل max_depth را برابر ۳ قرار داده و پارامتر مخصوص به هرس کردن (ccp_alpha) را ابتدا صفر قرار می دهیم. این پارامتر با حذف گره هایی که تا دقت خوبی داده ها را جداسازی کرده‌اند، از overfit شدن مدل بر داده های آموزش جلوگیری کرده و مقدار محاسبات را هم کاهش دهیم. برای ایجاد درخت ثابت هم در هر مرحله اجرا، random_state = 93 قرار می‌دهیم.

سپس مدل را بر داده های آموزش fit می کنیم و با plot_tree درخت را نمایش می دهیم.

برای داده های ارزیابی می توان با متد predict. خروجی را مشاهده کرد. با متد score. هم دقت داده ها بر داده های آموزشی و ارزیابی را می‌سنجیم تا عملکرد مدل را بررسی کنیم.

با توجه به دقت های به دست آمده، نیاز داریم تا هایپرپارامتر های مدل را تنظیم کنیم تا به دقت بالاتری برسیم.

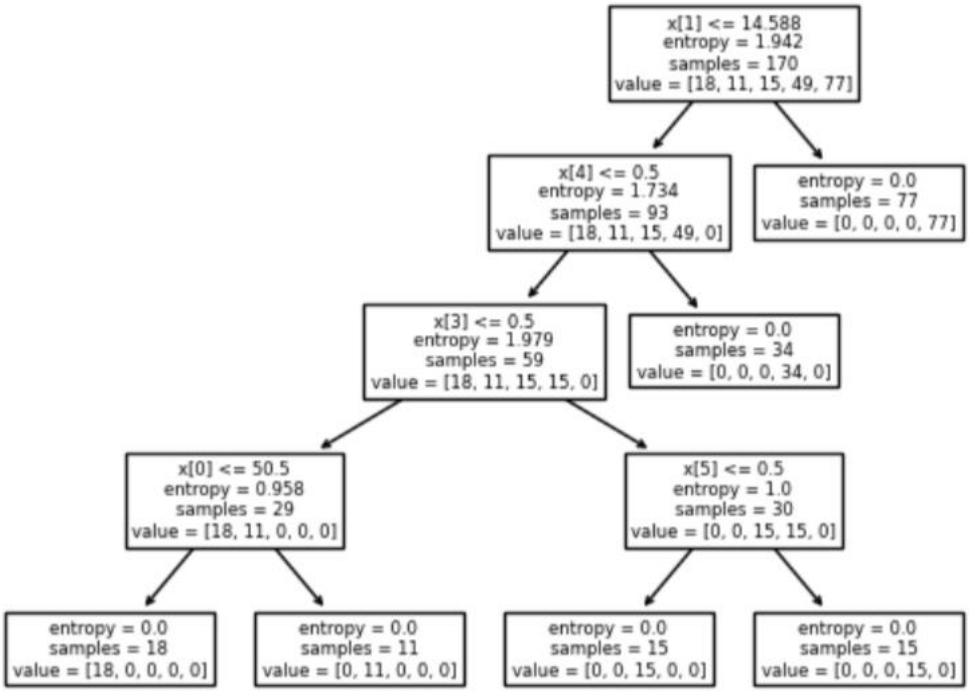



```
Train Accuracy: 0.8470588235294118
Test Accuracy: 0.7333333333333333
```

برای مدل بهتر، معیار سنجش را به جای ‘gini’ ، ‘entropy’ قرار داده، عمق درخت را برابر ۴ قرار می دهیم و پارامتر هرس کردن را برابر ۰/۰۰۱ قرار می دهیم.

```
DecisionTreeClassifier
DecisionTreeClassifier(ccp_alpha=0.001, criterion='entropy', max_depth=4,
                       random_state=93)
```

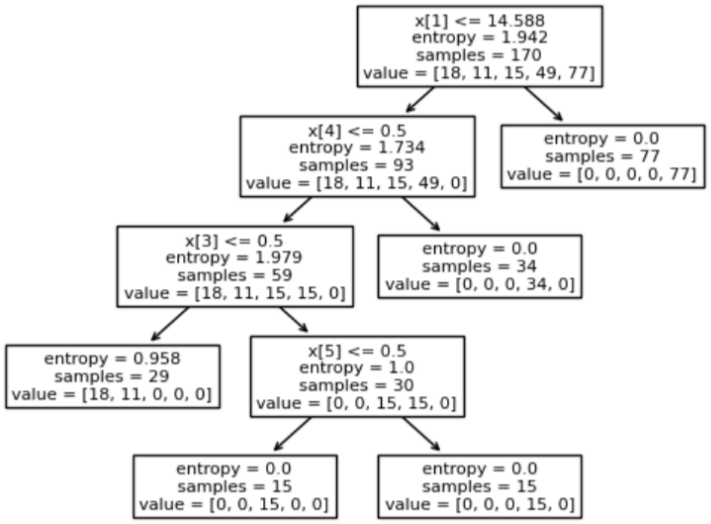
نتایج درخت در این حالت به صورت زیر می شود.




```
Train Accuracy: 1.0
Test Accuracy: 0.9666666666666667
```

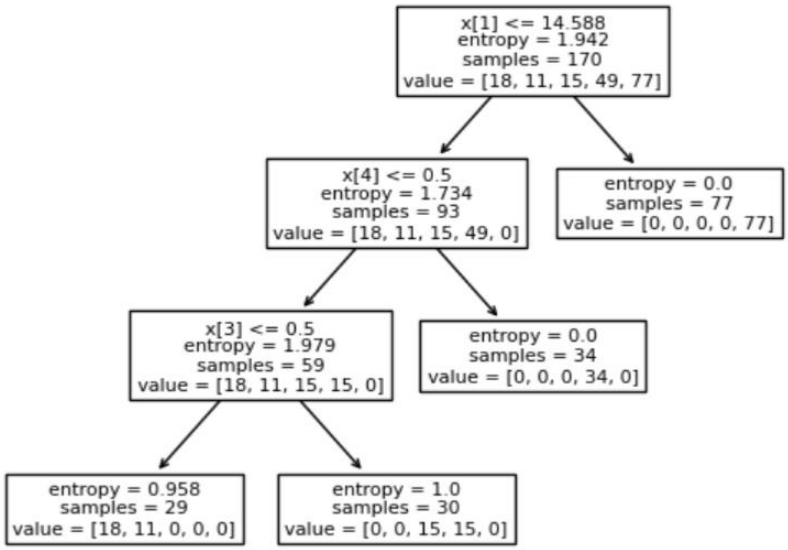
پارامتر هرس کردن:

با تغییر پارامتر هرس کردن مشاهده می کنیم در حقیقت مقدار 0.001 در حقیقت هیچ اثری ندارد. با تغییر آن مقدار کمینه این پارامتر برابر 0.17 است که درخت با دقت نسبتا خوبی به شکل زیر حاصل می شود. مقادیر کمتر از 0.17 اثری بر درخت نمی‌گذارد.




```
Train Accuracy: 0.9352941176470588
Test Accuracy: 0.8
```

مقدار بیشینه نیز حدود 0.18 است که دقت حاصل پایین آمده و مورد نظر نیست.

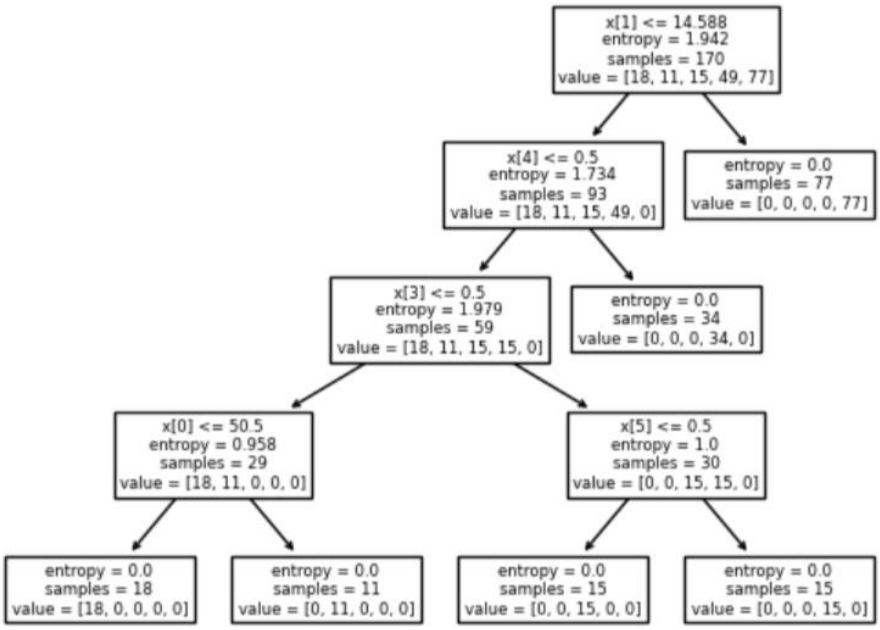



```
Train Accuracy: 0.8470588235294118
Test Accuracy: 0.7333333333333333
```

تحلیل درخت:

برای تحلیل درخت، model2 را تحلیل می‌کنیم که دقت بالا و موردنظر ما را برآورده کرده است.

علاوه بر شکل درخت و اطلاعات داخل آن از متد export_text() استفاده می کنیم که مسیر های درخت را با توجه به ویژگی ها مشخص می کند.




```
'|--- feature_1 <= 14.59\n|   |--- feature_4 <= 0.50\n|   |   |--- feature_3 <= 0.50\n|   |   |   |--- feature_0 <= 50.50\n|   |   |   |   |--- class: 0\n|   |   |
|--- feature_0 >  50.50\n|   |   |   |--- class: 1\n|   |   |--- feature_3 >  0.50\n|   |   |   |--- feature_5 <= 0.50\n|   |   |   |   |--- class: 2\n|   |   |
|--- feature_5 >  0.50\n|   |   |   |--- class: 3\n|   |--- feature_4 >  0.50\n|   |   |--- class: 3\n|--- feature_1 >  14.59\n|   |--- class: 4\n'
```


طبق این درخت، ابتدا با feature 1 تصمیم گیری انجام شده و مقدار عددی با توجه به محاسبات داخل درخت صورت گرفته است. در داخل text مسیر یک شاخه کامل طی شده است و سپس به شاخه دیگر پرداخته شده. ابتدا شرط کوچکتر feature 1 بررسی شده در این حالت به سراغ feature 3 رفتیم و با شرط کوچکتر آن از feature 0 استفاده می کنیم تا دو در نهایت دو کلاس ۰ و ۱ با آن پیش بینی شوند. سپس در text به گره قبلی بازگشتیم و شرط بزرگتر feature 3 را با feature5 بررسی کرده و کلاس های ۲ و ۳ پیش بینی شده‌اند. شرط بزرگتر feature 4، ادامه کلاس ۳ و شرط بزرگتر گره ابتدایی، کلاس ۱ را مشخص می‌کند.

مسیر دو نمونه از داده ها:

برای مشخص کردن مسیر داده ها از متد decision_path() استفاده می‌کنیم. این متد گره‌هایی که برای تصمیم گیری داده مورد نظر استفاده شده‌اند را نشان می دهد و به کمک آن می‌توان مسیر داده را مشخص کرد.

برای این منظور ابتدا کلاس حقیقی و سپس کلاس پیش بینی شده داده تست را نمایش می‌دهیم و در ادامه با استفاده از متد گفته شده، مسیر گره ها را مشخص می کنیم.

```
#First sample
i = 5
print(f"x_example: {x_test.iloc[i]}")
print(f"\ny_example: {y_test.iloc[i]}")

p1 = model2.predict(x_test.iloc[[i]])
print(f"\nPrediction: {p1}")

path = model2.decision_path(x_test.iloc[[i]])
print(f"Path: {path.toarray()}")
```

```
x_example: Age                  20.000
Na_to_K               11.686
Sex_M                  0.000
BP_LOW                 1.000
BP_NORMAL              0.000
Cholesterol_NORMAL     1.000
Name: 182, dtype: float64

y_example: 3

Prediction: [3]
Path: [[1 1 1 0 0 0 1 0 1 0 0]]
```

در نمونه اول کلاس پیش بینی و حقیقی یکسان و هر دو برابر ۳ هستند.

طبق آرایه خروجی مسیر (‘Path’)، گره ابتدایی طبیعتاً فعال شده، سپس طبق شماره گذاری از چپ به راست این متد، گره چپ فعال شده (Feature 4) در ادامه در داخل این گره باز گره سمت چپ (Feature 3) فعال شده. سپس صفر ها نشان دهنده غیر فعال بودن گره داخلی سمت چپ (Feature 0) و زیرمجموعه هایشان است. و ادامه آرایه مشخص می کند که گره سمت راست (Feature 5) فعال شده و داخل آن نیز گره راست مربوط به کلاس ۳ (با ۱۵ نمونه) فعال شده است که طبق انتظار ما می‌باشد.

```
#Second sample
i = 10
print(f"x_example: {x_test.iloc[i]}")
print(f"\ny_example: {y_test.iloc[i]}")

p1 = model2.predict(x_test.iloc[[i]])
print(f"\nPrediction: {p1}")

path = model2.decision_path(x_test.iloc[[i]])
print(f"Path: {path.toarray()}")
```

```
x_example: Age                  31.000
Na_to_K              11.871
Sex_M                 1.000
BP_LOW                0.000
BP_NORMAL             0.000
Cholesterol_NORMAL    1.000
Name: 100, dtype: float64

y_example: 0

Prediction: [0]
Path: [[1 1 1 1 1 0 0 0 0 0 0]]
```

نمونه دوم هم مشابه قبل تحلیل می شود. این نمونه که مربوط به کلاس صفر است، تمام گره های سمت چپ را فعال کرده تا در نهایت به گره پیش بینی مربوط به کلاس صفر رسیده است.

کلاس حقیقی مدل نیز در این حالت صفر بوده است.

قسمت ۳ :

داده ها را مشابه قبل با دستور gdown بارگذاری کرده و داخل دیتافریم با read_csv قرار می‌دهیم.

دیتاست مربوط به امید به زندگی ۲۹۳۸ نمونه از افراد است که دارای ۲۱ ویژگی (feature) مختلف است. هدف ما از این مسئله، طراحی مدلی بر پایه درخت تصمیم است تا امید به زندگی افراد را به خوبی پیش‌بینی کند در نتیجه با یک مسئله رگرسیون روبرو هستیم.

ابتدا باید تعداد خانه‌های دیتافریم با مقدار نامشخص (’Nan‘) را پیدا کنیم. برای این کار از دستور ()isnull.sum. استفاده می کنیم و طبق جدول زیر تعداد داده های نامشخص در هر ستون پیدا می‌شود.

```
Country                            0
Year                               0
Status                             0
Life expectancy                   10
Adult Mortality                   10
infant deaths                      0
Alcohol                          194
percentage expenditure             0
Hepatitis B                      553
Measles                            0
 BMI                              34
under-five deaths                  0
Polio                             19
Total expenditure                226
Diphtheria                        19
 HIV/AIDS                          0
GDP                              448
Population                       652
 thinness  1-19 years             34
 thinness 5-9 years               34
Income composition of resources  167
Schooling                        163
dtype: int64
```

در ادامه با دستور ()dropna. تمام سطر های دارای مقادیر ’Nan‘ را حذف کرده و پس از بررسی دوباره دیتافریم، مشاهده می‌شود که تعداد ’Nan‘ ها به صفر رسیده‌اند.

در این صورت دیتافریم دارای ۱۶۴۹ نمونه خواهد بود.

سپس ابتدا با متد ()drop. ستون مربوط به امید به زندگی (’Life expectancy‘) را از df1 حذف کرده و باقی داده ها را داخل df2 قرار می دهیم. برای حذف داده های توصیفی مشابه قبل از get_dummies. استفاده می کنیم تا با one-hot encoding آن ها را به صورت اعداد درآوریم.

سپس تمام داده های df2 را داخل X ریخته و برای Y از دیتافریم df1 تنها ستون مربوط به امید به زندگی (ستون ۳) را داخل آن قرار می‌دهیم.

به کمک train_test_split داده ها را با نسبت ۸۰ به ۲۰ به آموزش و تست تقسیم می کنیم.

سپس درخت تصمیم برای مسئله رگرسیون را با مدل DecisionTreeRegressor ایجاد می‌کنیم. با تنظیم هایپرپارامتر ها به منظور دستیابی به بهترین نتیجه، مدل نهایی به صورت زیر حاصل می‌شود.

```
DecisionTreeRegressor
DecisionTreeRegressor(ccp_alpha=0.001, max_depth=12, random_state=93)
```

در این مدل عمق درخت برابر ۱۲، و ضریب مربوط به پارامتر هرس کردن برابر 0.001 در نظر گرفته شده که با توجه به این حالت دقت برای داده های اموزشی و تست بالا خواهد بود و به صورت زیر می‌شود.

```
Train Accuracy: 0.9889820427141071
Test Accuracy: 0.9318197442028934
```