توضیحات پروژه

- **ساخت درخت :**

در هر node بهترین سوال با استفاده از دو فاکتور entropy یا gini index پیدا می‌شود و بر اساس سوال داده ها در فرزندان متفاوتی تقسیم می‌شوند . سپس برای هر فرزند تابع نوشته شده دوباره صدا زده می‌شود تا بهترین سوال بعدی انتخاب شود .

```
def decision_tree(child , Questions , parent_question , compare_base , height =-1):
    if (parent_question != titanic_root ) :
        # if len(instances[1]) == 0 :
        #     return

        #set answer for each child
        if len(Questions) == 0 or height+1 >= 4:
            Set_Answer(parent_question)


    if compare_base == "Entropy":
        question = min_entropy(child , Questions)
    else :
        question = min_gini(child , Questions)

    if question == None :
        return

    child.Question = question
    child.Height = height +1
    parent_question.Attribute = question
    # question.Parent = parent_question
    for ch in question.Children :
        if ch.Answer == -1 and len(ch.Instances) != 0 :
            decision_tree(ch , [x for x in Questions if x != question] , question , compare_base , ch.Height)
    return question
```

توضیحات کد :

(این تابع به صورت بازگشتی نوشته شده است)

در هر بار صدا زده شدن تابع شروط زیر در نظر گرفته می‌شود :

۱. نمونه ها موجود باشند .

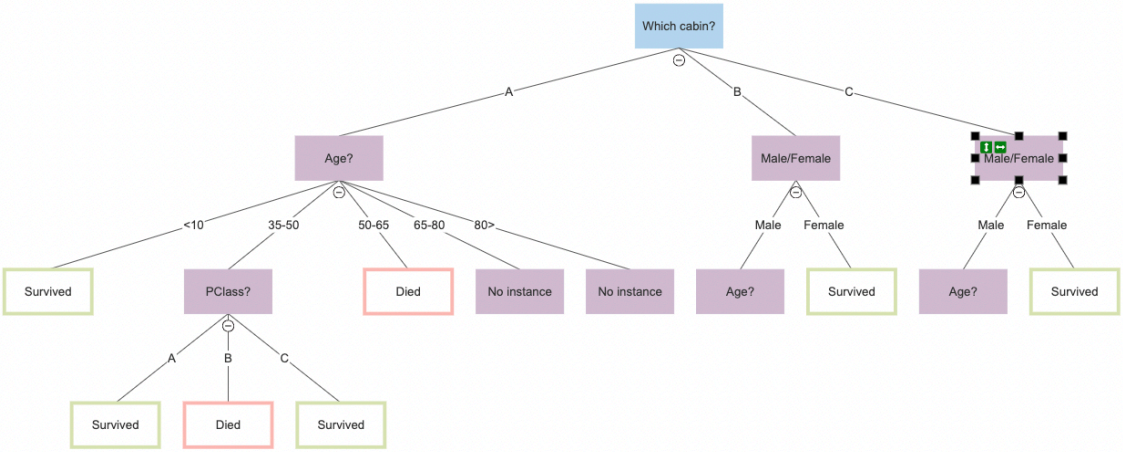
۲. سوالات موجود باشند .

۳.ارتفاع درخت بیش از ۴ نشود .(این عدد با در نظر گرفتن نمونه های این پروژه و تعداد سوالات انتخاب شده است)

۴. به جواب نرسیده باشیم .

در شروط ۲ و ۳ برای فرزند به وجود اومده باید جواب مشخص شود که این جواب با مقایسه نمونه های مثبت و منفی مشخص می‌شود ، اگر تعداد نمونه های منفی بیشتر بود جواب ۰ و در غیر اینصورت ۱ تنظیم می‌شود . جواب در ویژگی Answer از کلاس child ذخیره می‌شود .

بخشی از درخت به دست آمده با استفاده از مولفه آنتروپی :



در انتها می‌توان با استفاده از تابع Information Gain میتوان درخت را هرس کرد و از ارتفاع درخت کم کرد . من این کار را با در نظر گرفتن شرط ماکسیمم ارتفاع در تابع اصلی ساخت درخت انجام دادم و بخش پیاده سازی این تابع کامنت شده است .

- **محاسبه آنتروپی و** Gini Index

برای محاسبه آنتروپی با استفاده از فرمول آن ، ابتدا تعداد نمونه های منفی هر فرزند محاسبه و شانس انتخاب شدن آن به دست آورده شد . برای هر فرزند اینکار تکرار شد و در نهایت آنتروپی هر سوال برابر با جمع وزن دار حاصل این فرمول قرار گرفت .

چون ماکسیمم فرزندان ممکن برای یک سوال ۶ در نظر گرفته شده است ، مبنای لگاریتم در فرمول آنتروپی ۶ قرار داده شده است .

برای محاسبه Gini index نیز همین روند ، با در نظر گرفتن فرمول آن پیش برده شده است .

- **تابع های BFS و Test :**

به ترتیب برای رسم درخت و به دست آوردن دقت درخت خروجی پیاده سازی شده اند . برای مثال برای نمونه های رستوران خروجی و دقت به شکل زیر می‌شود.

```
Patron? | alternate? | WaitEstimate? | alternate? |

Bar? |


Restaurant Tree accuracy: 87.5
```

که یعنی در مرحله اول سوال Patron پرسیده می‌شود سپس به ترتیب از چپ به راست از فرزندان Alternate ، WaitEstimate، Alternate پرسیده می‌شوند ، سپس از فرزند alternate سوال Bar پرسیده میشود . چون داده های موجود کم بودند ارتفاع درخت به دو رسید .

دقت به دست آمده با ۵۰ درصد داده آموزشی و ۵۰ درصد داده تست ۸۷/۵ شده است .

- **توضیحات درباره ساختار کلاس ها :**

دو کلاس Child و Question به شکل زیر تعریف شده اند :

کلاس Question:

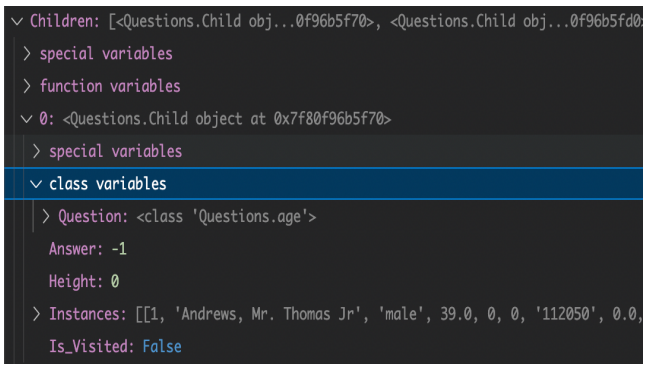
```
class Question :
    Children = []
    Data = "NoQuestion"
    Entropy = 1
    GiniIndex = 1
    def __init__(self):
        pass

class Child:
    def __init__(self):
        self.Answer = -1
        self.Instances = []
        self.Question = None
        self.Is_Visited = False
        self.Height = -1
```

هر سوال از Question ارث بری شده است که علاوه بر ویژگی های تعریف شده یک متد Split مختص به خود دارد که با کمک آن بچه ها در کلاس هایی از جنس Child در آرایه Children ذخیره می‌شوند .
در ویژگی Data اسم سوال ذخیره می‌شود .
در Entropy و GiniIndex مقادیر آنتروپی و جینی ایندکس برای مراحل بعد و پیدا کردن بهترین سوال ذخیره می‌شوند . این داده ها بعدا در هرس نیز کاربرد دارند .

یک نمونه از کلاس Question در تصویر رو به رو مشاهده می‌شود .(بهترین سوال در ریشه)

کلاس Child:

هر کدام از بچه های درخت از این کلاس ارث بری می‌کنند . این کلاس برای سادگی فهم درخت پیاده سازی شده است .
به ترتیب در این کلاس جواب کلاس ، نمونه های موجود ، بهترین سوال بعدی(بهترین سوالی که از نمونه های موجود می‌توان پرسید) ، ویزیت شده بودن یا نبودن(برای پیاده سازی BFS) و ارتفاع بچه از Root ذخیره شده است .



یک نمونه از کلاس Child مشاهده می‌شود .
(فرزند اول در ارتفاع ۱) که مشاهده می‌شود بهترین سوال برای پرسیدن از این فرزند سوال Age است .

- **خواندن داده ها**

داده ها با کمک تابع کمکی read_csv از کتابخانه pandas در یک لیست تعریف شده اند . از این داده ها یک کلاس از جنس Child ساخته می‌شود .
اگر ۸۰ درصد داده ها به عنوان train در نظر گرفته شوند ، دقت درخت بیشتر می‌شود ، اما ممکن است باعث overtrain شدن ماشین شود(هرچه تعداد نمونه های آموزشی کمتر باشد ممکن است دقت بالاتر برود اما باعث کاهش تضمین بودن درستی درخت می‌شود) . بنابراین بهترین تفکیک ،در نظر گرفتن ۵۰ـ۷۰ درصد داده ها برای آموزش و مابقی برای تست است . در خواندن و تفکیک اطلاعات برای فیلد های Nan یا None ، می‌توان روش های مختلف را در نظر گرفت .برای مثال اگر اطلاعاتی راجع به سن فرد نداریم ، می‌توانیم بیشترین سن موجود در جامعه مثلا ۳۰ سال را در نظر بگیریم .
همچنین می‌توانیم به صورت رندوم جوابی را خروجی دهیم ، برای ساخت این درخت دو حالت آزمایش شده است: اگر جواب این داده ها را ۱ فرض کنیم و یا صفر . که مشاهده شد در حالت دوم دقت به مراتب بیشتر می‌شود .

- **چالش ها و ایده بهبود**

در پیاده سازی کلاس های مرتبط با هر سوال و ارتباط دادن این کلاس ها با یکدیگر کمی با مشکل مواجه شدم . در ابتدا برای پیاده سازی کلاس Child از tuple با flag های مختلف استفاده کردم که فایل آن نیز موجود است .
علاوه بر آن چون question ها ممکن بود در ارتفاع های متفاوت درخت تکرار شوند ، این تکرار باعث عوض شدن فیلد های سوال قبل پرسیده شده میشد که ساختار درخت را بهم میزد .این مشکل با ساخت نمونه از هر سوال در هر بار صدا زدن درخت حل شد .
برای بهبود ارتقاع درخت و سرعت ساخت درخت می‌توان از هرس کردن با در نظر گرفتن حداقل اختلاف آنتروپی والد و فرزند ۰.۳ استفاده کرد یا می‌توان در حین ساخته شدن درخت اینطور در نظر گرفت که اگر تعداد نمونه های مثبت خیلی بیشتر از تعداد نمونه های منفی است (یا برعکس) در همان مرحله جواب مشخص شود و عمق درخت بیشتر نشود .
برای بهبود دقت درخت می‌توان در نظر گرفت که اگر اطلاعاتی از نمونه در یک فیلد خاص نداریم حالات مختلف برای تفکیک این نمونه را در نظر بگیریم . سپس محاسبه کنیم که به طور کلی فرض گرفتن کدام حالت به ما بهترین جواب را میدهد؟ مثلا اگر pclass را نداریم می‌توانیم یک بار با فرض A بودن کلاس ، یکبار B بودن کلاس و... دقت را محاسبه کنیم . اینکار ممکن است وقتگیر باشد اما زمانی که اطلاعات را برای داده های زیادی نداریم دقت را افزایش می‌دهد .
برای تفکیک و split کردن در هر کلاس من داده ها را با مشاهده تفکیک کردم . میتوان اینکار را با آزمایش کردن حالات مختلف و محاسبه بهترین دقت انجام داد .