

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده علوم کامپیوتر و ریاضی

پروژه دوم یادگیری ماشین

نگارش

سید حسین محمدی

مدرس

دکتر اکبری

آبان 1401

|  |  |
| --- | --- |
| فهرست مطالب | صفحه |

[بخش اول: سؤالات تشریحی 3](#_Toc119847578)

[1-1- رسم و محاسبه به همراه تست درخت تصمیم 3](#_Toc119847579)

[1-2- دقت و صحت 6](#_Toc119847580)

[2-1- جنگل تصادفی و واریانس بالای درخت تصمیم 6](#_Toc119847581)

[2-2- آیا درخت تصمیم حریصانه با کمک از معیار های دیگر، اوپتیمال هست؟ 7](#_Toc119847582)

[3-1- عمق درخت ها یا تعداد درخت ها در جنگل تصادفی؟ 7](#_Toc119847583)

[3-2- دلیل عمکرد بهتر جنگل تصادفی با مقدار ثابت عمق درخت ها. 7](#_Toc119847584)

[بخش دوم: تمرین مطالعه (Explained AI using Random Forest) 8](#_Toc119847585)

[2-1 مقدمه 8](#_Toc119847586)

[2-1 مثال از سیستم هایی که میتوانند از XAI استفاده کنند 8](#_Toc119847587)

[2-1 عملیات مورد نظر 8](#_Toc119847588)

[2-1 مثالی از نتیجه ها 9](#_Toc119847589)

[بخش سوم: پیادهسازی 11](#_Toc119847590)

[3-1- کد نویسی طبقه بندی 11](#_Toc119847591)

[3-1-1- محاسبه گرادیان softmax 11](#_Toc119847592)

[3-1-2- تنظیم threshold 12](#_Toc119847593)

[3-1-3- KFold 12](#_Toc119847594)

[3-1-4- مقایسه نتایج با sklearn 13](#_Toc119847595)

[3-1-5- نتایج کد دستی برای Glass.csv 15](#_Toc119847596)

[3-2- Multiclass classification 18](#_Toc119847597)

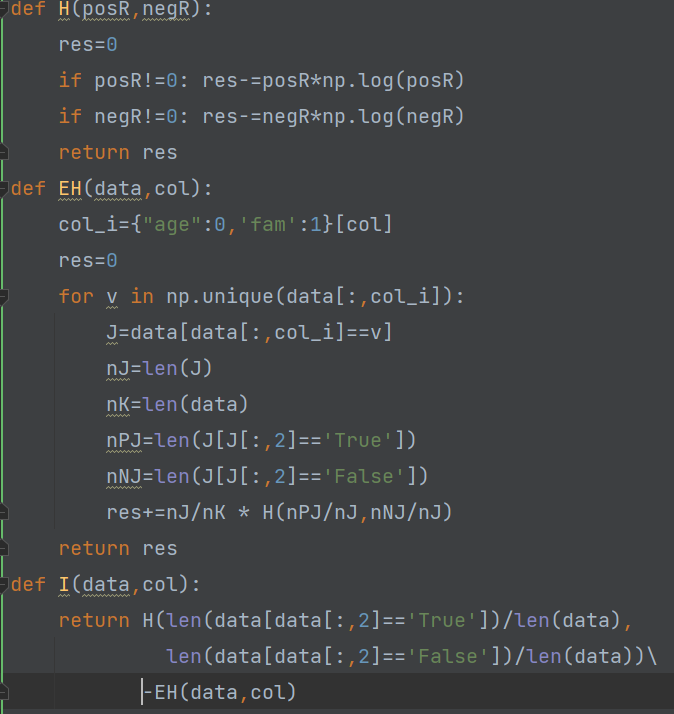
[3-3- Fraud Detection with randomforest 20](#_Toc119847598)

# بخش اول: سؤالات تشریحی

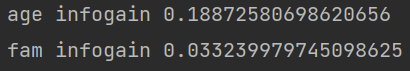
## رسم و محاسبه به همراه تست درخت تصمیم



با کمک از کامپیوتر برای محاسبات، توابع زیر را تعریف کردیم.

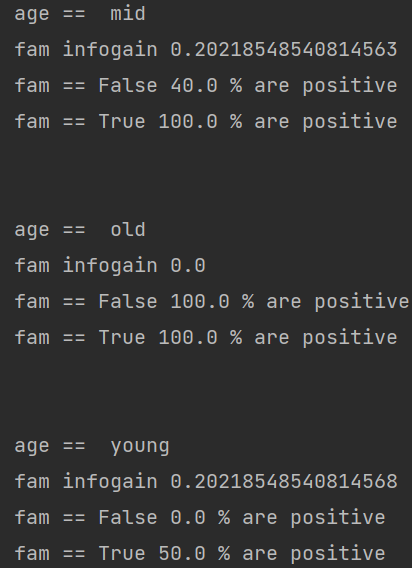


حال InformationGain را برای بچه های ریشه حساب میکنیم.

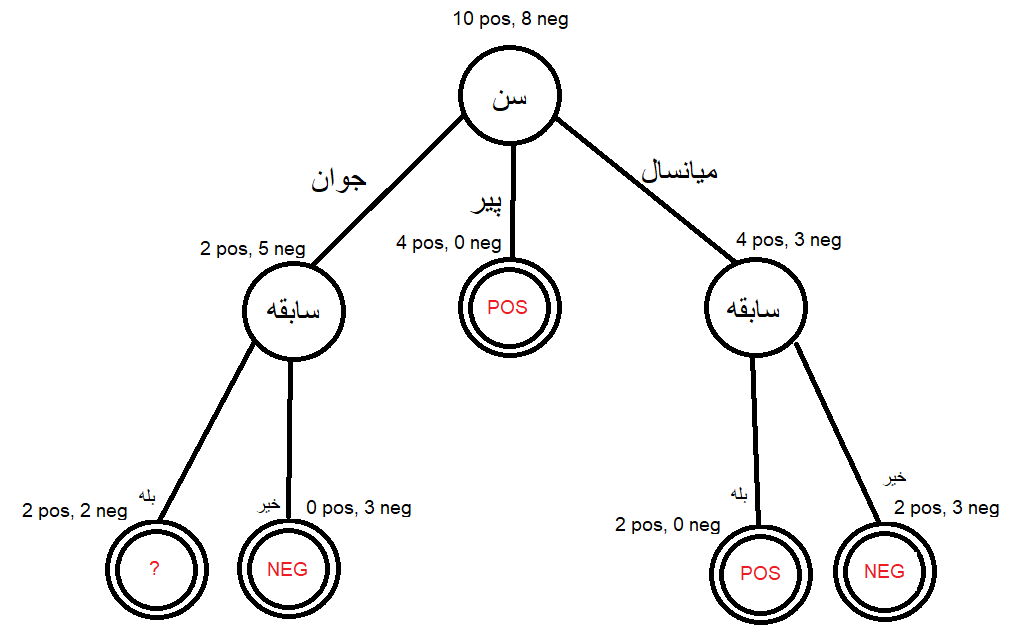


طبق این، ریشه ما سن هست.

حال بچه های بعدی را انتخاب میکنیم. لذا فقط یک انتخاب وجود دارد.



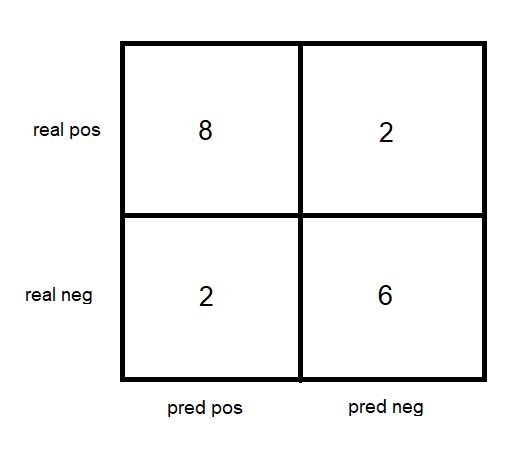
شکل درخت:



نتیجه برای داده تست هر دو مثبت هست. 19 و 20 = TRUE

## دقت و صحت

\*\*اگر تصمیم تصمیم افراد جوان و با سابقه قبلی را مثبت فرض کنیم\*\*



دقت: 0.777

صحت: 0.8

صحت اگر تصمیم تصمیم افراد جوان و با سابقه قبلی را منفی فرض کنیم: 0.6

## جنگل تصادفی و واریانس بالای درخت تصمیم

درخت تصمیم با عمق بالا به راحتی واریانس بالا میگیرد. در یک جنگل، تعداد زیادی درخت با واریانس های بالا وجود دارد برای دسته هایی از ویژگی های وجود دارد. به دلیل اینکه این واریانس ها شباهت به خطا رندوم دارند، ممکن هست همدیگه را لغو کنند. به زبان دیگر خطای یک دسته از درخت ها، خطا دسته دیگر را رقیق میکنند. این ملزوم این هست که تعداد درخت های متمایز زیاد باشد.

## آیا درخت تصمیم حریصانه با کمک از معیار های دیگر، اوپتیمال هست؟

خیر. یافتن یک درخت اوپتیمال یک مسئله NP-complete هست. با فرض آن که درخت اوپتیمال دارای هایپر پارامتر های اوپتیمال برای بدست آوردن بهترین دقت و صحت برای داده تست باشد، جواب خیر میماند زیرا که انتخاب حریصانه نمیتواند ترکیب داده ها را به صورت کلی تشخیص دهد و نیازمند عمق بیشتری برای تفکیک این تفاوتات هست. اما عمق بیشتر باعث افزایش واریانس میشود و دقت مدل روی داده تست را کم میکند.

## عمق درخت ها یا تعداد درخت ها در جنگل تصادفی؟

*افزایش تعداد درخت ها باعث کاهش واریانس میشود ولی اگر عمق درخت ها به اندازه کافی نباشد، ممکن هست باعث افزایش بایاس شود.*

*از طرف دیگر افزایش عمق درخت ها واریانس را افزایش میدهد و حتی با تعداد زیاد درخت، اگر عمق همه بیش از حد زیاد باشد، ممکن هست باعث واریانس غیرقابل تعمیر شود. با تعداد درخت کم با عمق بالا، احتمال ایجاد درخت های متمایز کمتر میشود و باعث افزایش واریانس کل میشود.*

## دلیل عمکرد بهتر جنگل تصادفی با مقدار ثابت عمق درخت ها.

*در یک درخت با عمق بالا، ممکن هست شروطی وجود داشته باشند که باعث افزایش واریانس شود. در یک جنگل با عمق کمتر، هر درخت فقط بخشی از ویژگی ها را میبیند و مجبور به استفاده برخی از ویژگی ها میشود که در یک درخت بزرگتر در عمق بیشتر فقط دیده میشود.*

*با ترکیب چنین درخت هایی میتوان مجموعه عضیمی از شروط را کاوش کرد بدون آن که کل مدل باعث واریانس شود که باعث بهبود عملکرد آن به نسبت یک درخت با عمق برابر میشود.*

# بخش دوم: تمرین مطالعه (Explained AI using Random Forest)

## مقدمه

بسیاری از مدل ها یادگیری ماشین مانند یک جعبه سیاه کار میکنند و نمیتوان به راحتی دلیل رسیدن آن مدل به یک جواب خاص را درک کرد. در درخت های تصمیم میتوان دلایل هر تصمیم و فرضیات (تصمیم های فبلی) را به وضوح دید و مسیری که مجموعه داده میرود را میتوان فهمید. درخت تصادفی شامل مجموعه ای از این درخت ها است و میتوان همین اقدام را برای هر کدام از این درخت ها انجام داد که به این معنا هست که میتوان درخت تصادفی را نیز تا حد بسیار خوبی درک کرد. از این قابلیت میتوان برای بهبود عملکرد برخی سیستم ها استفاده کرد.

## مثال از سیستم هایی که میتوانند از XAI استفاده کنند

داده هایی که unbalanced هستند و یا به هر دلیلی تصمیم درست گرفتن برای آن ها سخت هست باعث لیبل اشتباه میشود و میتوان از XAI برای تخمین خطا و flag زدن آن میشود.

چنین لیبل اشتباهی را ما با در نظر گیری دلایل رسیدن به چنین لیبلی تشخیص میدهیم و با کمک از مدلی دیگر یا هر روش های دیگر این خطا را حل میکنیم.

برای مثال در تشخیص نوع حمله به یک شبکه ممکن هست به مشکل لیبل اشتباه به دلیل شباهت برخی از حملات برخورد کنیم. در اینصورت میتوان با کمک از قابلیت توصیف دلایل تصمیم مدل هایی همچون درخت ها، لیبل صحیح را بدست آورد.

## عملیات مورد نظر

فرض کنید که جنگل تصادفی شما بر روی داده عملکرد خوبی دارد اما برخی از لیبل ها را جابجا میزند. به زبان دیگر برخی از کلاس ها را شبیه به هم میبیند. برای مثال داده های unbalanced یا ویژگی های مشابه.

این کلاس ها که با هم اختلال دارند را جدا کنید. حال دسته بندی جدیدی براساس اختلال نداشتن لیبل ها ایجاد کنید (هر لیبل جدید نماینده تعدادی از لیبل های قبلی هست که با هم اختلال ندارند). حال یک مدل جدید بر روی این دسته بنده آموزش دهید.

از مدل جدید میتوان برای دسته بندی دقیق تر دسته های قبلی استفاده کرد. اما 2 مسئله وجود دارد:

اول فرض کنید برای تعدادی از داده ها در دو مدل در دسته بندی های متناقض قرار گیرند. در این صورت کدام یک را باید قبول کرد.

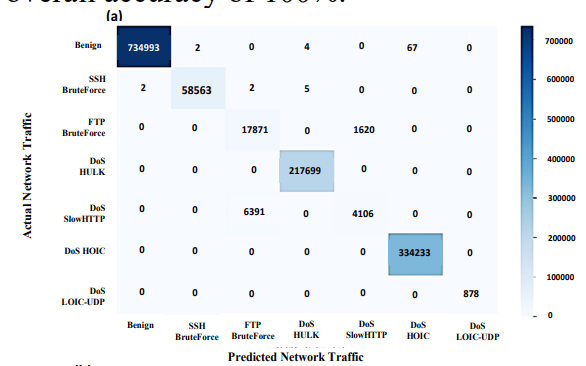
دوم آنکه استفاده از 2 جنگل جدا گانه ممکن هست به صرفه نباشد.

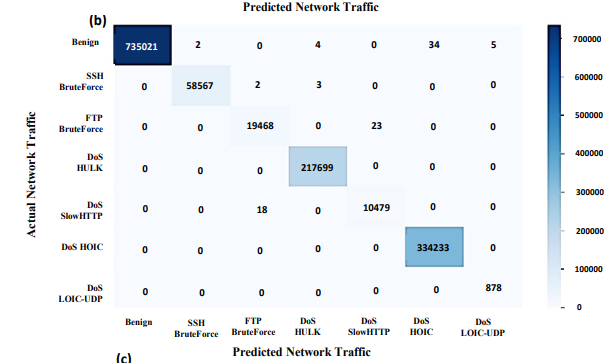
برای حل هر دو این مشکل ها میتوان از یک CAM (Credibility Assessment Module) استفاده کرد. ما میتوانیم برای هر داده، اعتبار آن را تخمین بزنیم و با کمک از این اعتبار، تصمیم بهتر بگیریم.

در پی این هدف، ما لیستی از اهمیت ویژگی ها برای رسیدن به یک تصمیم را بدست میآوریم که از جنگل تصادفی به راحتی بدست می آید.   
حال داده خود را به همراه تصمیمی که توسط اولین جنگل صورت گرفته را در نظر بگیرید. میتوان به راحتی مسیر پیموده شده (اهمیت ویژگی ها برای آن داده) را بدست آورد. حال این لیست از اهمیت داده را با مقدار اهمیت که داده ها با همان لیبل دارند را با هم مقایسه میکنیم. اگر مشابه باشند، یعنی اعتبار تصمیم بالست و در غیر اینصورت میتوانیم داده را به جنگل دوم برای تخمین دقیق تر ارسال کنیم.

در عمل ما کاری شبیه به یک autoencoder انجام میدهیم. به این شکل که از داده یک لیست جدیدی میسازیم (که در اینجا لیست اهمیت ویژگی ها هست) و آن را با مقداری که انتظار میرود برای آن لیبل باشد مقایسه میکنیم. به زبان دیگر یک نوع توزیع جدید از داده بدست میآوریم و آن را با توزیع داده آموزش مقایسه میکنیم.

## مثالی از نتیجه ها





تصحیح لیبل در مسئله یافتن حمله در شبکه.

در این مسئله به دلیل حجم داده در شبکه، نیازمند رسیدن به تصمیم به صورت realtime و سریع میباشیم و استفاده از 2 مدل برای هر داده به صرفه نیست.   
همچنین دانستن اعتبار یک نتیجه میتواند در اعتماد کاربر و شبکه نیز کمک کند.

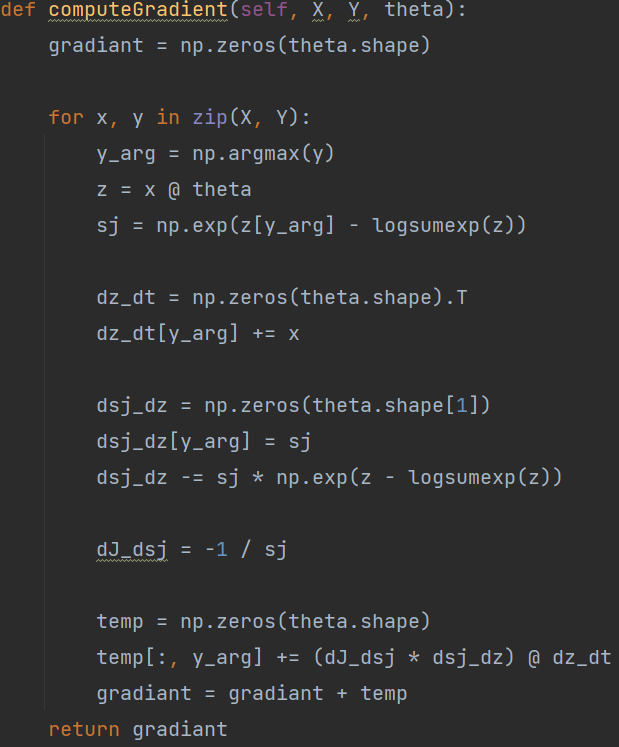
# بخش سوم: پیادهسازی

## کد نویسی طبقه بندی

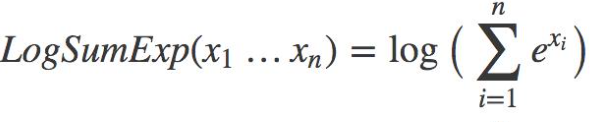
## محاسبه گرادیان softmax

بنده اشتباهی برای این تمرین softmax را بجای sigmoid نوشتم. Softmax برای 2 کلاس، همان sigmoid هست ولی میتواند تعداد کلاس های بیشتر را نیز پردازش کند.

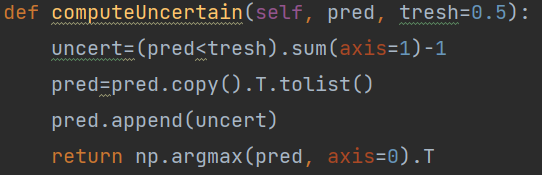
در پی این هدف، برای بهبود سرعت، تلاش کردم گرادیان را به صورت وکتوری انجام دهم. اما به دلیل ابعاد بالای تنسور ها مربوطه، صرف نظر کردم و به صورت وکتوری برای هر داده جداگونه محاسبه کردم.



یک مشکل که با آن مواجه شدم، مقدار بسیار زیاد e^x بود که باعث خطا overflow میشد. با درنظر گیری اینکه ما مقدار لوگاریتم مجموعه آن ها را میخواهیم، برای حل این مشکل از تابع logsumexp استفاده کردم.



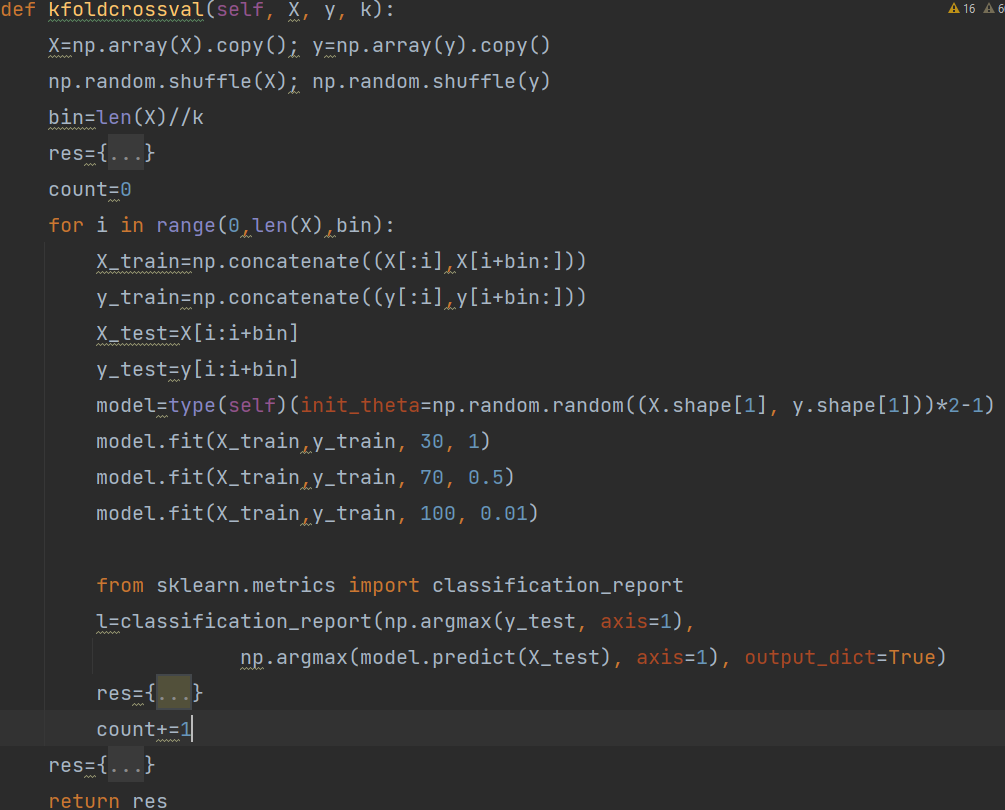
## تنظیم threshold



در اینجا، یک کلاس خروجی جدید به اسم uncertain میسازیم که مقادیری که به حد کافی قطعیت ندارند را نشان میدهد.

## KFold

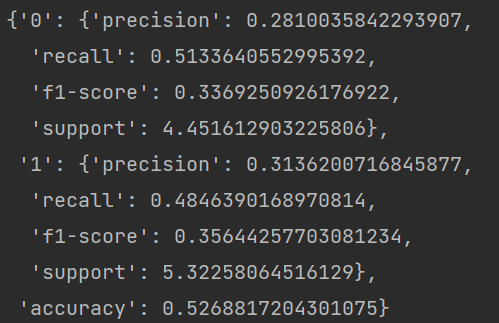
در این مرحله، داده را به len\_x/k بخش تقسیم میکنیم و مدل را جداگانه روی هر کدام تمرین میدهیم و از خطای آن ها میانگین میگیریم.



## مقایسه نتایج با sklearn

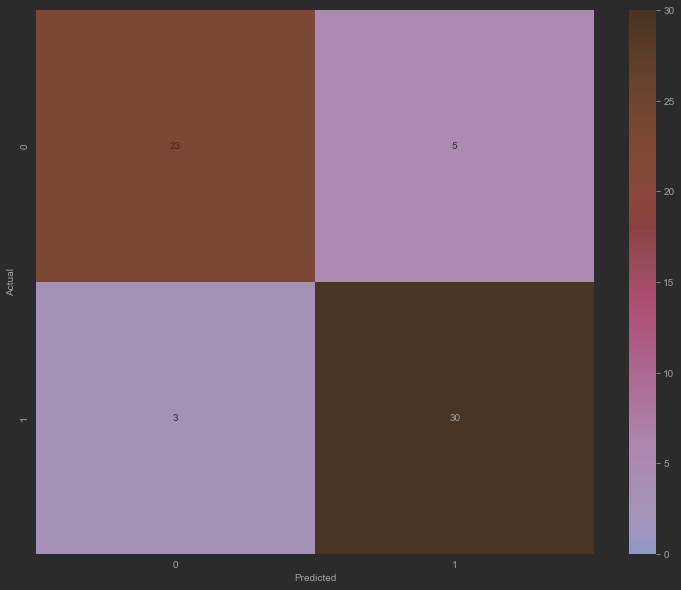
برای این مرحله، ابتدا داده heart را نرمال میکنیم و سپس یک ویژگی bias به آن اضافه میکنیم.

حال برای آزمایش توان مدل، KFold نوشته شده را برای آن اجرا میکنیم.



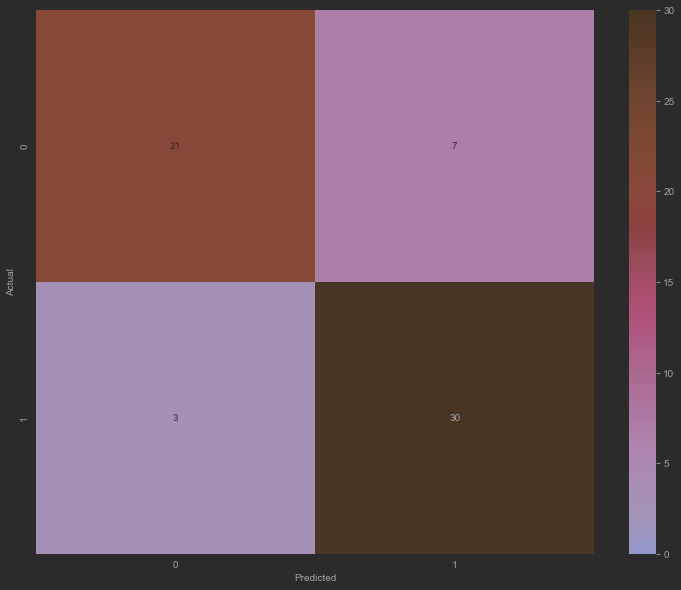
حال داده را به 2 بخش تمرینی و تستی تقسیم میکنیم. در این تقسیم بندی، نسبت تعداد کلاس ها را ثابت نگه داشتیم. با داده تمرینی، آموزش دادیم و برای از داده تست استفاده کردیم.

نتیجه تست کد دستی:



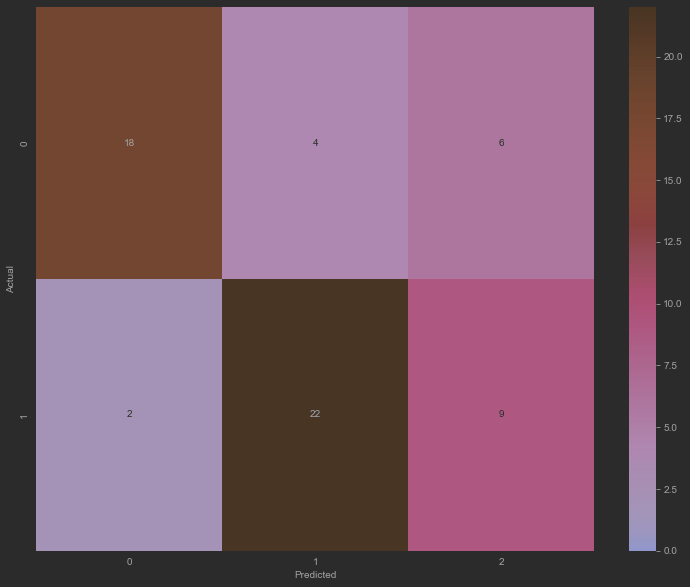
نتیجه تست sklearn:

نتیجه آن قابل مقایسه و تقریبا یکی میباشد.



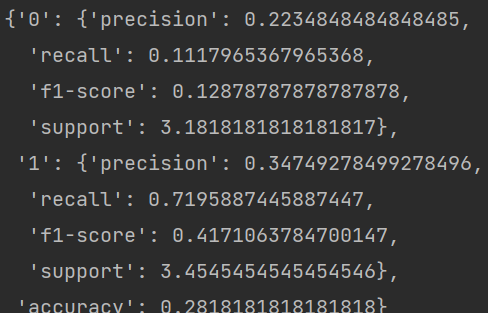
نتیجه تست کد دستی با threshold=0.7 (ستون سوم شامل مقادیری هست که دستبندی نشدند):

صحت و دقت با مقادیر مختلف threshold بهبود خاصی نداشته.

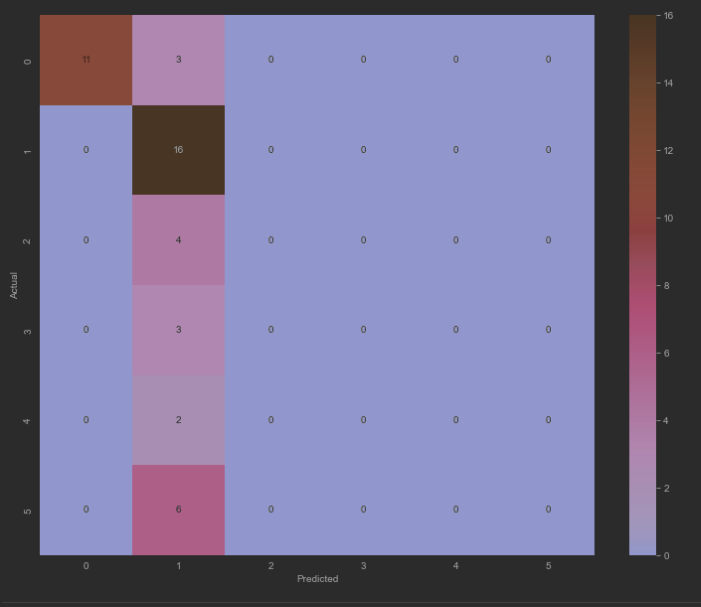
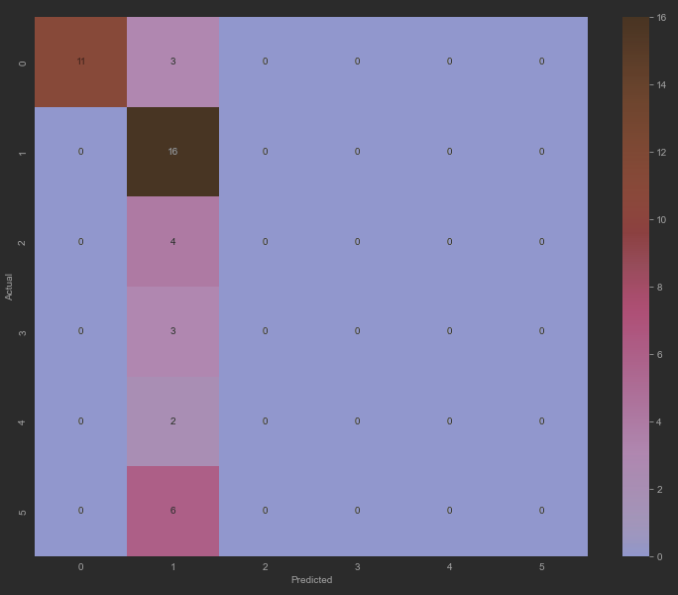


## نتایج کد دستی برای Glass.csv

چون کد softmax هست، میتوان چند کلاس را نیز طبقه بندی کرد.



sklearn و کد هر دو دقیقا یک نتیجه را میدهند.

## Multiclass classification

برای این بخش همانند قبل، داده را ابتدا نرمال میکنیم و به به 2 قسمت آموزشی و تستی تقسیم میکنیم (با نسبت ثابت بین کلاس ها).

با کمک از GridSearchCV بهترین مقادیر را برای hyperparameter ها میابیم.

همچنین دقت میانگین آن را با Kfold بدست میآوریم.

در نهایت نیز یک confusion\_matrix از آن نماش میدهیم

1. RandomForestClassifier

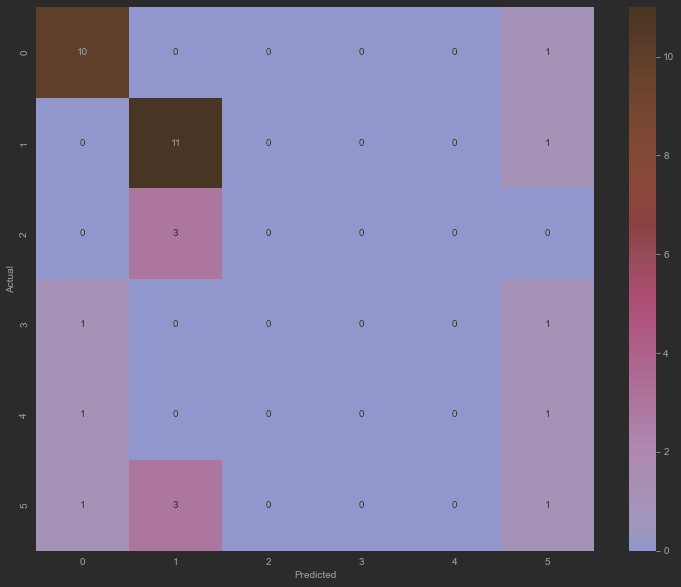
Best hyperparameter:



Kfold acc mean:



Confusion Matrix:



1. LogisticRegression

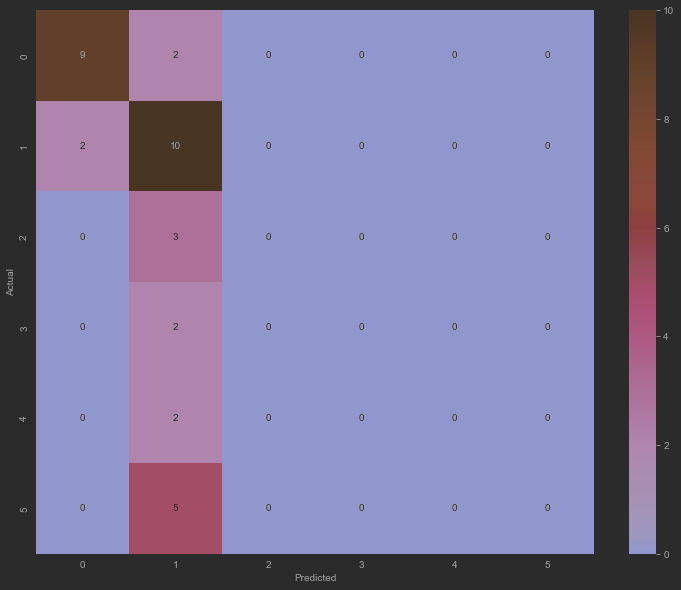
Best hyperparameter:



Kfold acc mean:



Confusion Matrix:



1. DecisionTreeClassifier

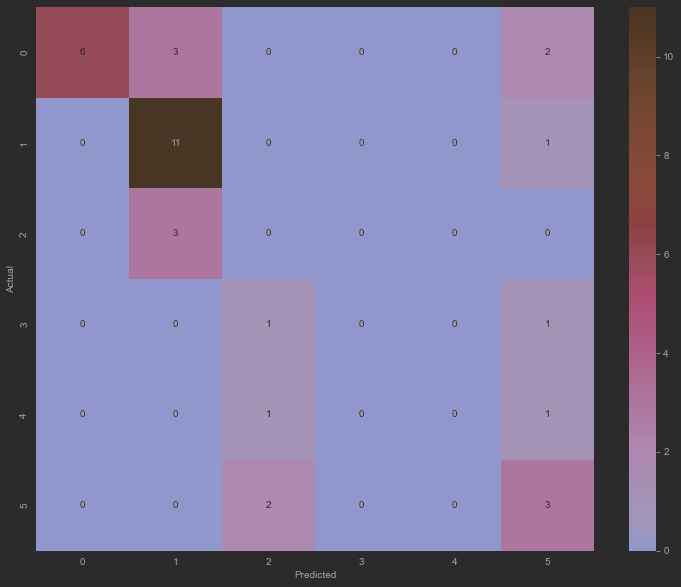
Best hyperparameter:



Kfold acc mean:



Confusion Matrix:



## Fraud Detection with randomforest

مقدار زیادی از از این تمرین شامل تمیز کردن و بهبود داده ورودی هست.

ابتدا داده های application\_data و previous\_application را با هم بر اساس ستون SK\_ID\_CURR ادقام میکنیم. 5.35% از داده ها ما اشتراکی ندارند و آن ها را حذف میکنیم.

حال داده را برای مقادیر عددی (که ترتیب ندارند) نرمال میکنیم.

سپس برای هر ستون که مقدار غیر عددی (و بدون ترتیب) دارند را در نظر میگیریم و مقادیری که در اقلیت هستند (کمتر 0.5% از داده هستند) را با مقدار other جایگزین میکنیم.

از حذف این مقادیر و ستون هایی که عمدتا از یک مقدار تشکیل میشوند، خودداری میکنیم زیرا که ستون هدف ما unbalanced هست و این امر ممکن هست قدرت پیشبینی ما را کاهش دهد. همچنین درخت های تصمیم ویژگی ها خوب را انتخاب میکنند و داشتن ویژگی اضافه باعث کاهش قدرت آن نمیشود (اما کند تر خواهد بود)

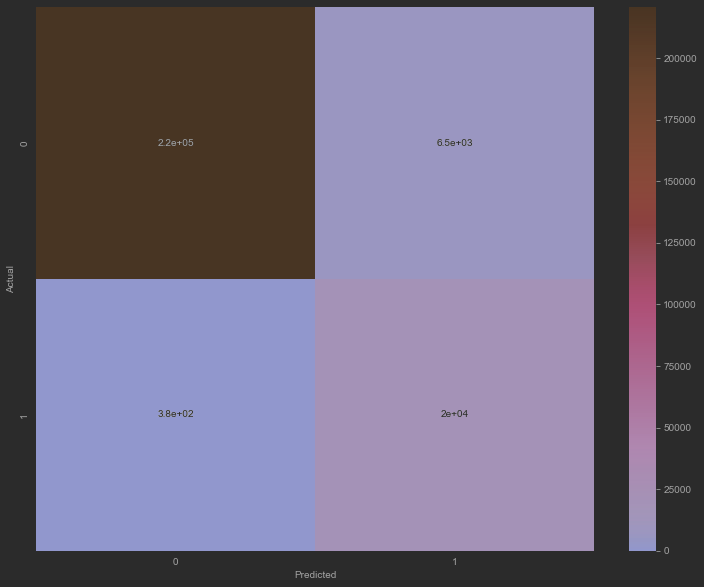
حال ستون های با 2 کلاس را با 0 و 1 جایگزین میکنیم. ستون ها با کلاس های بیشتر را با BinaryEncoder به چند ستون تبدیل میکنیم. علت استفاده از BinaryEncoder بجای روش های محبوب مانند onehotencode این هست که کل داده را در کمترین تعداد ستون ذخیره میکند و برای درخت ها مهم هست که تعداد ستون کم باشد.

مقادیر نا موجود را با 0 جایگزین میکنیم و یک ستون جدید برای نمایش نبود داده اضافه میکنیم.

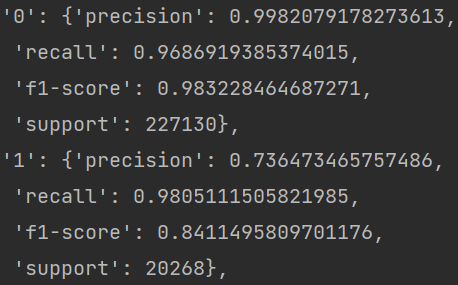
نهایتا 100 ستون را بر اساس روش های filter (استفاده از روش های آماری که ستون هدف را در نظر میگیرند مانند f\_classif) جدا میکنیم

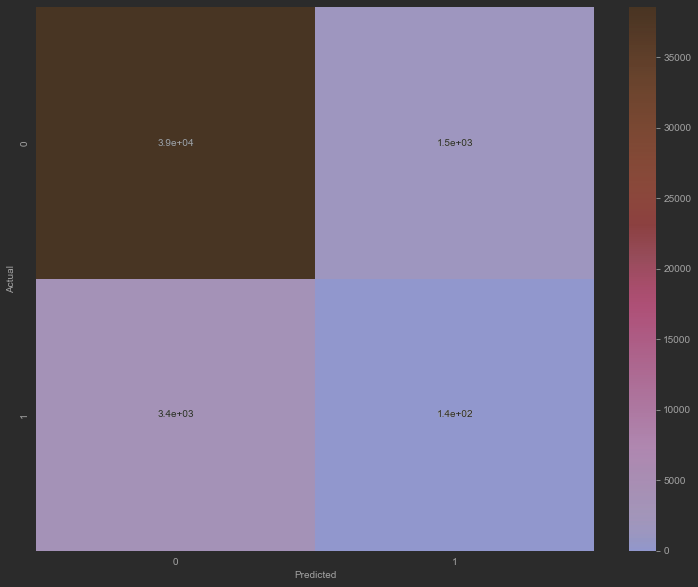
سرانجام داده را به بخش آموزشی و تست با رعایت نسبت کلاس ها، تقسیم میکنیم.

جنگل تصادفی را با پارامتر class\_weight آموزش میدهیم. با اینکه یکی از کلاس ها ده برابر کمتر هست، مقادیر مختلف class\_weight فرا تر از 12:1 تاثیر کمی دارد. پارامتر مهم دیگر عمق درخت (=24) هست زیرا که مدل به راحتی بر روی داده آموزشی overfit میشود و برای کاهش این مسئله از مقدار کمتر عمق استفاده میکنیم. تلاش برای استفاده از GridSearchCV به دلیل حجم داده بی نتیجه بود و راه ساده ای برای کنترول مقادیر نبود.

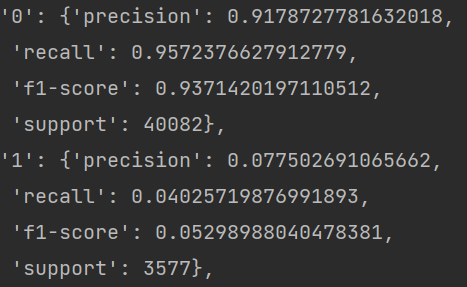
خروجی برای داده آموزشی:

معیار ها:



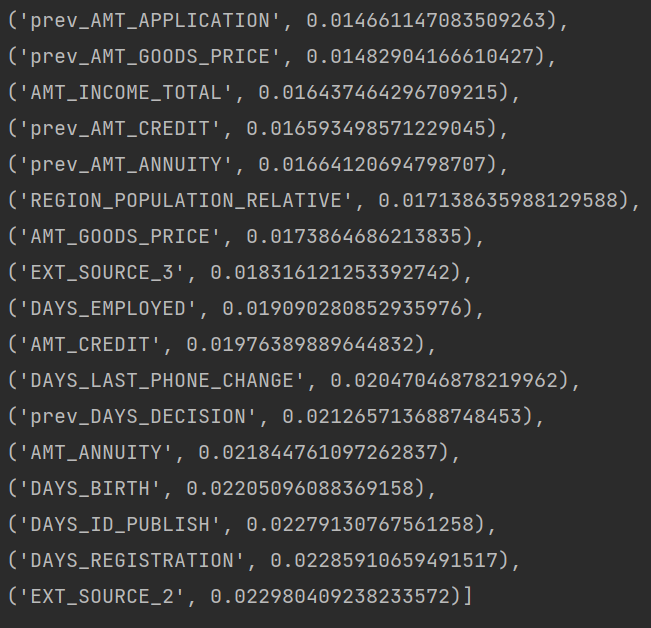
خروجی برای داده تست (هر کاری انجام دادم، بهتر نشد ☹)

معیار ها:

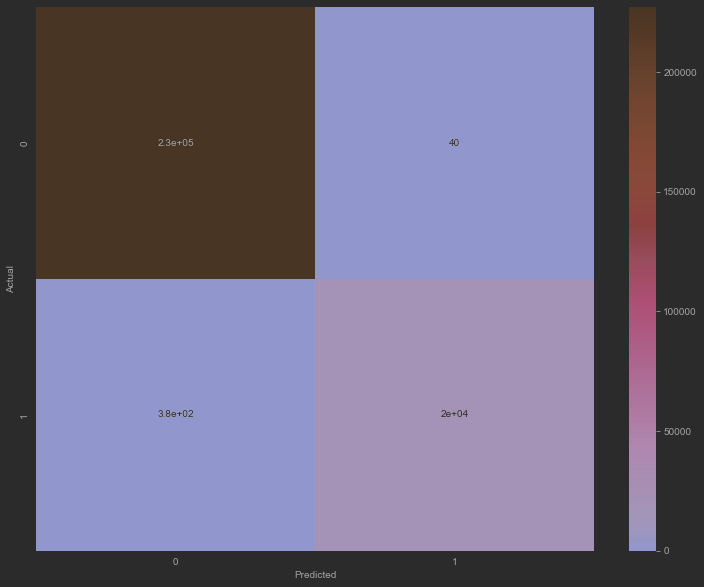


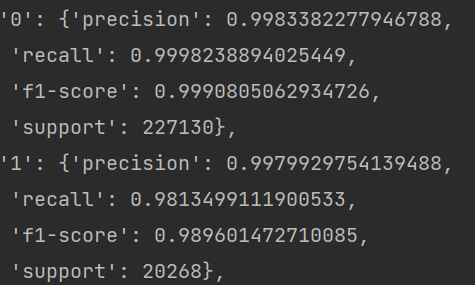
برترین ویژگی ها:

سن، وضعیت محل زندگی، اطلاعات شغلی، زمان ثبت نام و ...

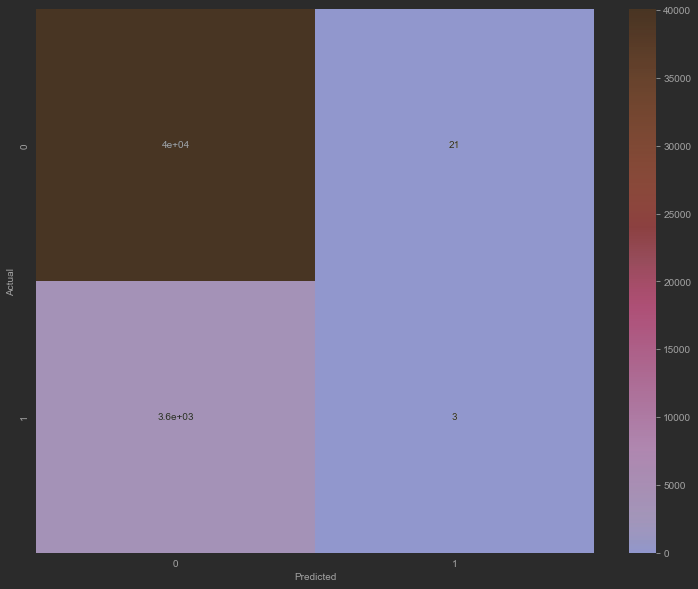


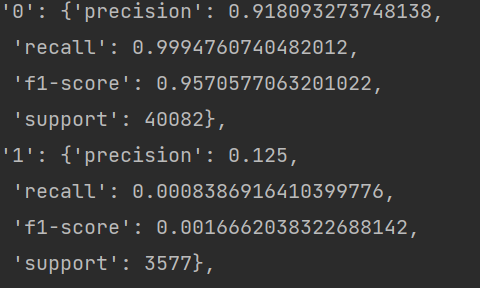
\*\*نتیجه با فقط 10 ویژگی اول:





تست:





\*\*جنگل تصادفی و درخت تصمیم به صورت خودکار بهترین ویژگی ها را انتخاب میکند و ویژگی های نامربوط را دور میریزد (با محدودیت عمق). به همین دلیل استفاده از کل ستون ها لزومی ندارد و میتواند به سرعت نیز کمک کند.



Amirkabir University of Technology  
(Tehran Polytechnic)

Department of Computer Science and Math

Machine Learning Assignment #2

By

Seyed Hossein Mohammadi

Taught by

Dr. Akbari

Nov of 2022