اقدامات اولیه:

در ابتدا به وسیله کتابخانه **pandas** داده ها از فایل ها خوانده و به فرمت **DataFrame** بر روی حافظه نگه داری می کنیم.
پس از بررسی های اولیه و بدست آوردن اطلاعاتی در مورد داده ها، داده ها را برای دسته بندی آماده می کنیم. برای این کار ستون اول داده ها که همان label کلاس است جدا می کنیم تا داده های آموزش از برچسب شان جدا شوند. حال اقدام به دسته بندی با روش های خواسته شده می کنیم.
با توجه به اطلاعات بدست آمده از داده ها، مسئله ای که قصد حل آن را داریم یک دسته بندی **binary** است. تعداد اعضای هر دو کلاس داده با هم برابر است و مشکلی از بابت **Imbalance** بودن داده ها نخواهیم داشت.

در ادامه برای سادگی کار در آینده و کوتاه تر شدن کد، 2 تابع کمکی، یکی برای رسم نمودار **ROC** و دیگری برای نمایش مقادیر ***,accuracy F1 recall precision*** برای هر دسته بندی، تعریف می کنیم.

دسته بندی با استفاده از دسته بند RandomForest:

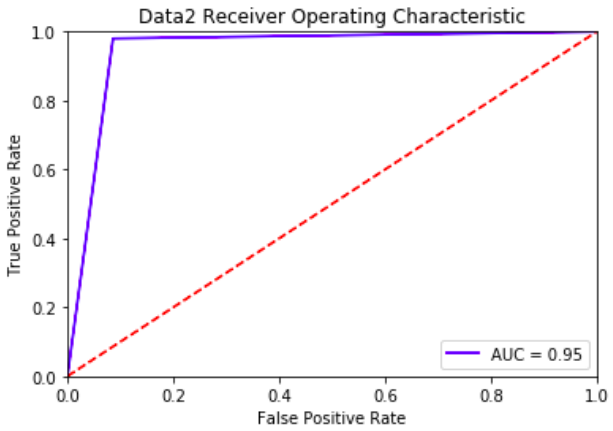
برای دسته بندی به این روش از دسته بند موجود در کتابخانه **sklearn** با نام **ensambles.RandomForestClassifier** استفاده می کنیم.
در ابتدا به پارامتر های دیفالت تعریف شده در خود دسته بند اکتفا کرده و تنها ***max*depth** *را برابر با* ***3*** *قرار می دهیم. همین طور برای مقایسه روش ها با یکدیگر و قابلیت بازسازی نتایج بدست آمده با توجه به وجود randomness در این روش، مقدار* ***random*state** یا همان **random seed** مورد استفاده برای تولید اعداد random را برابر با مقدار دلخواه **7** قرار دادیم.

پارامتر ها به ترتیب عبارتند از :

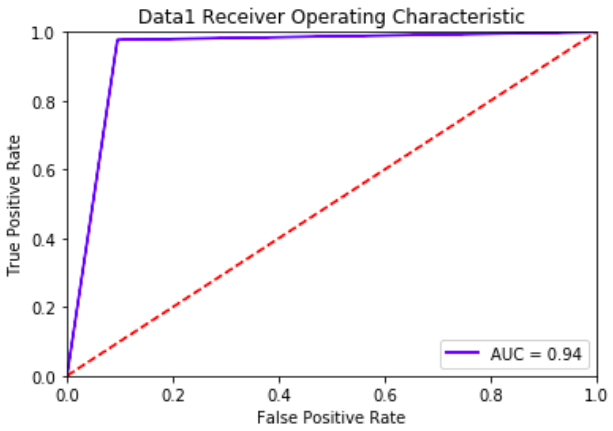
```
criterion='gini'
max_depth=3,
max_features='auto',
max_leaf_nodes=None
random_state= 7
```

پس از آموزش مدل روی داده های آموزشی، نتایج بدست آمده بر روی داده های **validation** به شرح زیر است:

```
RandomForest model for Data2:
accuracy: 0.941860 precision: 0.914894 f1_score: 0.945055
```




```
RandomForest model for Data1:
accuracy: 0.941860 precision: 0.914894 f1_score: 0.945055
```



دسته بندی با استفاده از دسته بند ELM:

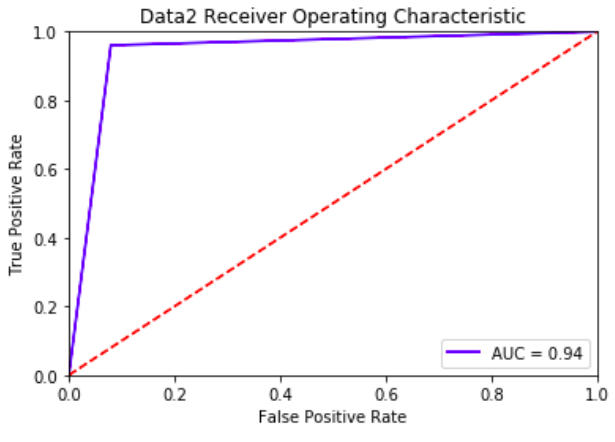
برای دسته بندی به این روش، چون کتابخانه ای برای این روش موجود نیست، باید روش را پیاده سازی کنیم. برای راحتی، سعی شده نهایت استفاده از توابع آماده موجود در کتابخانه های مختلف از قبیل ***keras***(که ساختار کلی روش یعنی شبکه عصبی و فرآیند **feed forward** را با استفاده از این کتابخانه انجام خواهیم داد) و ***numpy*** (که برای بدست آوردن وزن لایه های میانی به صورت **random** و یافتن شبه وارون برای ماتریس وزن ها از آن استفاده خواهیم کرد)، برده شود.
برای این کار 3 کلاس تعریف کرده ایم. کلاس اول نقش کلاس پایه و دو کلاس دیگر یکی برای دسته بندی **2 کلاسه یا binary** تعریف کرده ایم و دیگری را برای حالاتی که بیش از 2 کلاس داریم.
پس از تنظیم وزن های لایه اول، آن ها را با تنظیم عبارت **false** برای پارامتر **Trainable**، ثابت نگه می داریم تا در هنگام آموزش لایه آخر تغییر نکنند.

تعداد نورون های لایه ورودی با توجه به اندازه داده ها **110** در نظر گرفته شده. برای افزایش کارایی مدل و افزودن **non-linearity** بیشتر به مدل، **3** لایه هر کدام با **10** نورون و توابع فعال ساز به ترتیب ***relu, linear, relu*** به عنوان لایه های میانی به مدل اضافه می کنیم. در نهایت یک لایه با **2** نورون (چون قصد استفاده از**binary_crossentropy** به عنوان **loss function** را داریم) با تابع فعال ساز **sigmoid** به مدل اضافه کرده و مدل را complie می کنیم تا آماده فاز آموزش شود.
برای ادامه لازم است که برچسب ها را به صورت ***one-hot encode*** در بیاوریم. برای این کار هم از کلاس **OneHotEncoder** موجود در کتابخانه **sklearn** استفاده می کنیم. در نهایت نیز با تنظیم پارامتر های لازم برای تکمیل فرآیند، داده ها را برای آموزش لایه آخر به مدل می دهیم.
پارامتر های استفاده شده برای آموزش مدل:

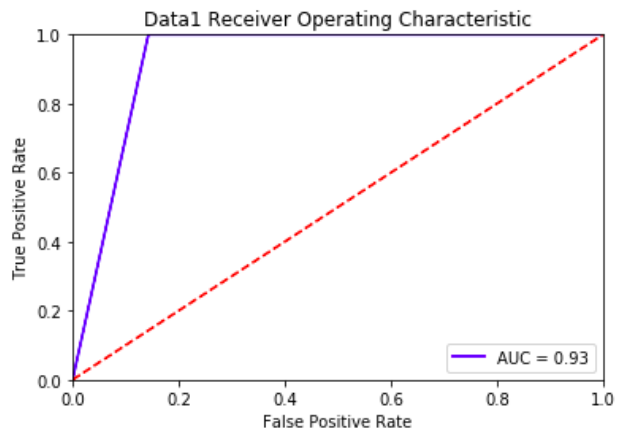
```
loss='binary_crossentropy'
optimizer=Adam(lr=0.001)
metrics=['accuracy']
```

نتایج بدست آمده در این روش به شرح زیر اند:

```
ELM model for Data2:
accuracy: 0.930233 precision: 0.880000 f1_score: 0.936170
```



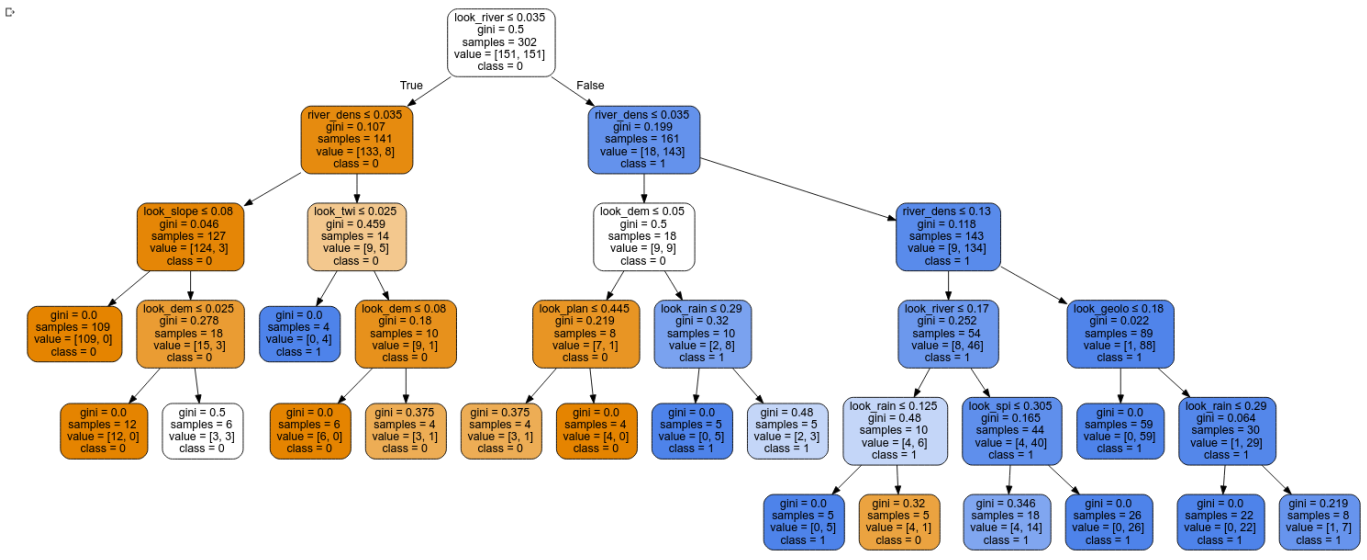

```
ELM model for Data1:
accuracy: 0.930233 precision: 0.880000 f1_score: 0.936170
```



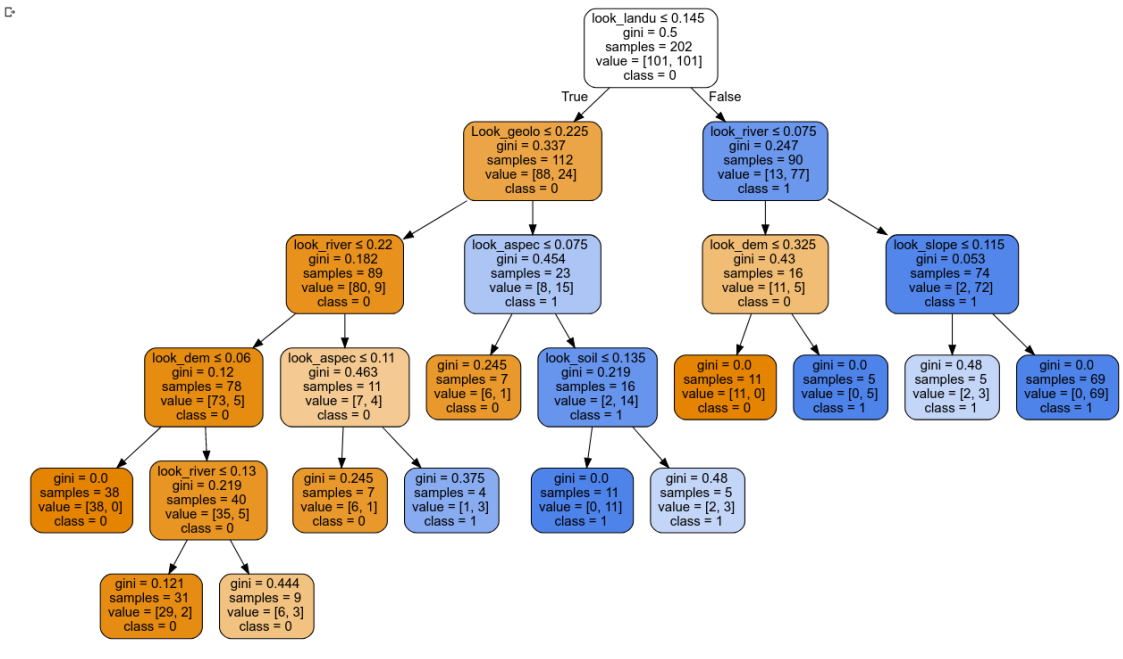
دسته بندی با استفاده از دسته بند درخت تصمیم با الگوریتم C4.5:

برای دسته بندی به این روش، با توجه به این که الگوریتم استفاده شده در پیاده سازی موجود این روش در کتابخانه **sklearn** از درخت های ***CART*** استفاده می کند، بررسی هایی بر روی تفاوت های این دو روش انجام شد و این نتیجه حاصل شد که می توان از پیاده سازی موجود در این کتابخانه به گونه ای استفاده کرد که با دقت زیادی مشابه الگوریتم **C4.5** عمل کند. برای همین به جای پیاده سازی از صفر، از پیاده سازی موجود در کتابخانه **sklean** استفاده خواهیم کرد. تفاوت های این دو روش تنها در **روش هرس کردن** است که فعلا برای ما کاربردی ندارد.
برای جلوگیری از **overfitting** و پیچیده شدن بیش از حد درخت، حداکثر عمق درخت را **5** و حداقل داده های لازم برای دسته بندی را **4** در نظر می گیریم.
درخت بدست آمده برای هر یک از دیتاست ها به صورت زیر است:

Data1

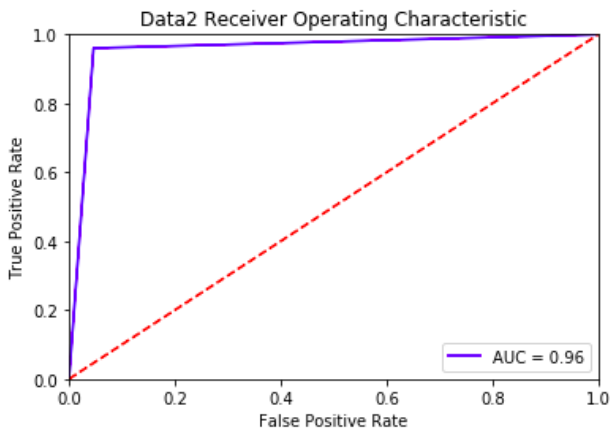


Data2

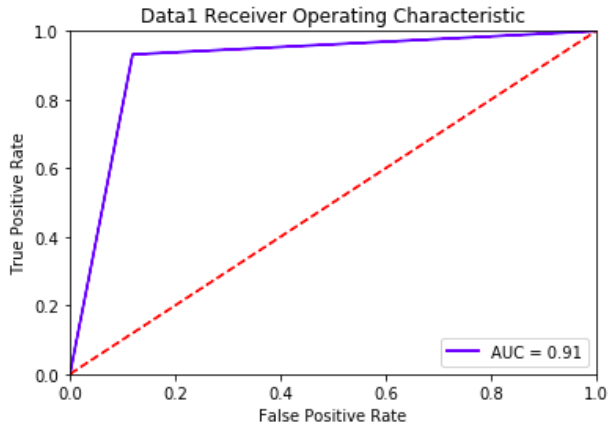


نتایج بدست آمده به صورت زیر است:

```
Tree model for Data2:
accuracy: 0.906977 precision: 0.891304 f1_score: 0.911111
```



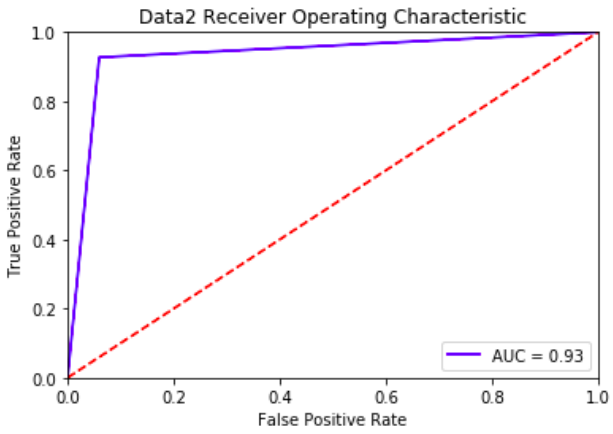

```
Tree model for Data1:
accuracy: 0.906977 precision: 0.891304 f1_score: 0.911111
```



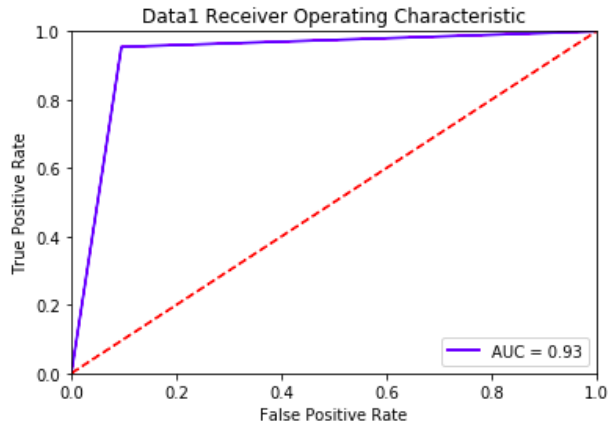
دسته بندی با استفاده از MLP:

برای دسته بندی به این روش، از کلاس ***MLPClassifier*** موجود در کتابخانه **sklearn** استفاده می کنیم.
مشابه قبل **random_state** را برای قابلیت مقایسه و بازسازی نتایج برابر **7** قرار می دهیم. در ابتدا از پارامتر های دیفالت استفاده می کنیم.
نتایج بدست آمده با پارامتر های دیفالت به شرح زیر می باشند:

```
MLP model for Data2:
accuracy: 0.930233 precision: 0.913043 f1_score: 0.933333
```




```
MLP model for Data1:
accuracy: 0.930233 precision: 0.913043 f1_score: 0.933333
```



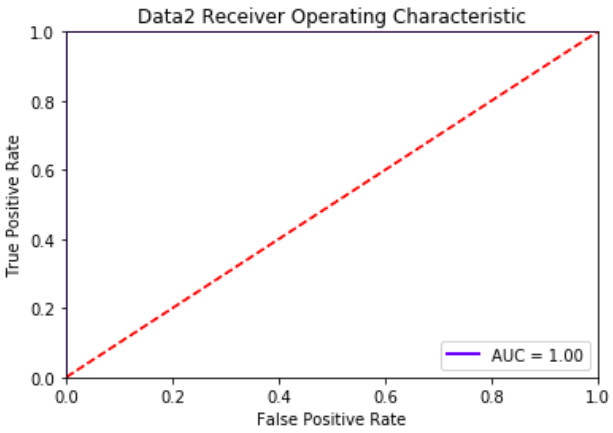
مشاهده می شود که نتایج بدست آمده با پارامتر های دیفالت قابل بهبود هستند. اولین قدم مشخص کردن تعداد لایه ها و نورون های هر لایه شبکه است. با توجه به تعداد تصاویر و کلاس ها تعداد لایه ها **3** و تعداد لایه های هر نورون **100** در نظر گرفته شد. برای بهبود فرآیند آموزش از الگوریتم **adam** در فرآیند back-propigation استفاده می کنیم، ضریب **regularization** را نیز برابر با **0.0001** قرار می دهیم و در نهایت شروط توقف فرآیند را تعیین می کنیم.

پارامتر های استفاده شده به صورت زیر است:

```
hidden_layer_sizes=(100,100,100)
solver='adam'
alpha=0.0001
max_iter=500
tol=0.000000001
random_state=7
```

نتایج بدست آمده به شرح زیر است:

```
MLP model for Data2:
accuracy: 0.988372 precision: 1.000000 f1_score: 0.988506
```




```
MLP model for Data1:
accuracy: 0.988372 precision: 1.000000 f1_score: 0.988506
```