"بسمه تعالی"

گزارش تمرین عملی اول یادگیری ماشین – زهرا لطیفی – ۹۹۲۳۰۶۹

فاز اول

ابتدا کتابخانه‌های نامپای و پانداس و همینطور matplotlib برای رسم نمودارها و تصاویر، ایمپورت شده‌اند. سپس مجموعه داده‌های آموزش و تست بارگذاری شده و ابعاد مجموعه داده، تعداد ویژگی‌ها و تعداد کلاس‌ها/برچسب‌ها مشخص شدند.

```
Number of rows is 14 & Number of columns is 5 & Number of features is 4 & Number of clas
ses/labels is 2.
```

در ادامه با دستور `train_data_m.head()` ۵ ردیف اول فایل دیتاست نمایش داده شده است.
در بخش بعد آنتروپی مجموعه داده محاسبه شده است.

سوال ۱: چگونه آنتروپی برای یک مجموعه داده مشخص محاسبه می‌شود؟

محاسبه آنتروپی از منظر ریاضی به شکل زیر خواهد بود:

Total number of training data = 20
data with "Yes" class = 13
data with "No" class = 7
H(S) = - p(Yes) * log2(p(Yes)) - p(No) * log2(p(No))
= - (13/20) * log2(13/20) - (7/20) * log2(7/20) = - (-0.4) - (-0.53) = 0.934

کد هم با همین منطق نوشته شده. در مجموعه داده به ازای هر کلاس، هرجا برچسب داده‌ای با برچسب آن کلاس برابر شده، یافت شده و تعداد آن مشخص شده. سپس آنتروپی با رابطه بالا محاسبه شده و پس از بررسی کامل تمام کلاس‌ها، آنتروپی‌ها با هم جمع شده‌اند.

در مرحله بعد برای هر ویژگی هم آنتروپی محاسبه شده.

سوال ۲: چگونه information gain برای یک ویژگی خاص محاسبه می‌شود؟

برای مجموعه داده صورت سوال مثلا داریم:

For Outlook we have: Sunny, Overcast and Rain

Sunny = 5
Sunny & Yes = 2
Sunny & No = 3

H(Outlook=Sunny) = -(2/5)*log(2/5)-(3/5)*log(3/5) = 0.971

Rain = 5
Rain & Yes = 3
Rain & No = 2

H(Outlook=Rain) = -(3/5)*log(3/5)-(2/5)*log(2/5) = 0.971

Overcast = 4
Overcast & Yes = 4
Overcast & No = 0

H(Outlook=Overcast) = -(4/4)*log(4/4)-0 = 0

با استفاده از آنتروپی هر ویژگی، بهره اطلاعاتی آن محاسبه می‌شود:

IG(Outlook) = p(Sunny) * H(Outlook=Sunny) + p(Rain) * H(Outlook=Rain) + p(Overcast) * H(Outlook=Overcast)
= (5/14)*0.971 + (5/14)*0.971 + (4/14)*0 = 0.693

منطق محاسبه آنتروپی هر ویژگی و بعد از آن بهره اطلاعاتی در کد هم به همین صورت است که در سوال ۲ ذکر شد و مانند بخش قبلی پیاده سازی شده است.

برای هر ویژگی آنتروپی شرطی حساب شده، بهره اطلاعاتی با کم کردن آن از آنتروپی کل بدست آمده است.

سپس مقداری برابر با ۱- برای حداکثر بهره اطلاعاتی قرار داده شده و به ازای تمام IGهای محاسبه شده بررسی می‌کنیم و هرکدام از آن بیشتر بود، جایگزین می‌شود تا جایی که بیشترین بهره پیدا شود. فیچری که این بیشترین بهرا را دارد، توسط تابع find_most_informative_feature برگردانده می‌شود.

تابع generate_sub_tree یک ویژگی را ایجاد می‌کند و ویژگی = مقدار را از مجموعه داده حذف می‌کند. اگر گرهی یک کلاس خالص نباشد، درخت ممکن است حاوی «?» به عنوان یک مقدار باشد. تابع، feature_name را که نام ویژگی که می‌خواهیم به مجموعه داده درختی اضافه کنیم و کوچک کنیم (مثلاً Outlook) و train_data را که دیتاست ماست و برچسب که نام دیتافریم (بازی تنیس) است و class_list که لیست کلاس‌های منحصر به فرد برچسب ([بله، خیر]) است را گرفته و گره درختی با شاخه‌های آن و مجموعه داده به روز شده بر می‌گرداند.

روند کار در تابع make_tree به این صورت است که ابتدا ویژگی با بیشترین بهره اطلاعاتی را پیدا می‌کنیم، یک گره با نام آن ویژگی و مقادیر ویژگی به عنوان شاخه می‌سازیم. اگر کلاس خالص بود، برگ (کلاس) به گره اضافه می‌شود. اگر کلاس ناخالص بود، یک گره قابل گسترش (’?’) به درخت اضافه می‌شود. هر بار مجموعه داده با توجه به کلاس خالص کوچک می‌شود.

شاخه کلاس ناخالص بعدی (’?’) را با مجموعه داده‌ای که حالا به روز شده، گسترش می‌دهیم. این تابع ریکرسیو نوشته شده و هنگامی کارش تمام می‌شود که مجموعه داده پس از به‌روزرسانی‌ها خالی شده باشد و یا شاخه قابل گسترش(ناخالص) دیگری وجود نداشته باشد. پس از اتمام کار، درخت کامل شده را بر می‌گرداند.

در تابع id3 این تابع برای ساختن درخت مربوط به دیتاست Pay Tennis فراخوانی شده است.

تابع پیش بینی در الگوریتم درخت تصمیم، برچسب کلاس یک نمونه ورودی را با پیمایش بازگشتی درخت ساخته شده پیش بینی می‌کند. بررسی می‌کند که آیا گره فعلی یک برگ است یا خیر و در این صورت مقدار آن را

برمی‌گرداند. برای گره‌های داخلی، ویژگی مورد استفاده برای تقسیم را استخراج می‌کند، درخت را بر اساس مقدار ویژگی نمونه حرکت میکند، و فرآیند را تا رسیدن به یک گره برگ تکرار میکند و برچسب کلاس پیشبینی‌شده را ارائه میکند. اگر مقدار ویژگی در زیردرخت یافت نشد، تابع با برگرداندن None به خوبی این کار را انجام می‌دهد.

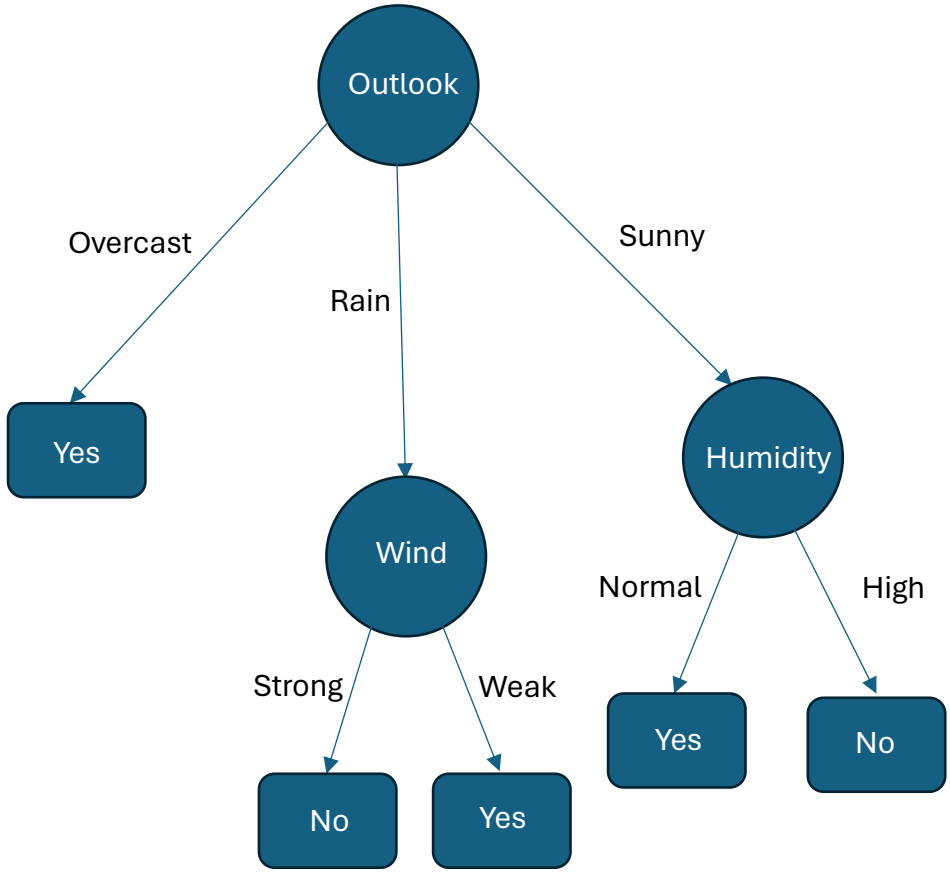
برای ارزیابی مدل یعنی درخت به یک مجموعه داده آزمایشی برچسب دار نیاز داریم. سپس پس از پیش‌بینی می‌توان اختلاف مقدار واقعی و پیشبینی‌شده را بر حسب درصد محاسبه کرد. تابع Evaluating این کار را انجام می‌دهد.

در مرحله بعد درخت را با فراخوانی تابع id3 ساخته و نتایج زیر را گرفتیم:

```
{'Outlook': {'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}},
  'Overcast': 'Yes',
  'Rain': {'Wind': {'Weak': 'Yes', 'Strong': 'No'}}}}

accuracy: 1.0
```

سوال ۳: درخت تصمیم ایجاد شده را رسم کنید و اینکه هر شاخه توسط کدام ویژگی تعیین شده است؟



در بخش Pruning مجموعه داده mushroom را بارگذاری کرده و نتایج زیر را گرفتیم:

```
training accuracy:  1.0
validation accuracy: 0.787
test accuracy: 0.796
```

فاز دوم

در این فاز کتابخانه‌های پانداس، نامپای، matplotlib، seaborn، sklearn ایمپورت شده‌اند. داده‌های مربوط به دادن/ندادن وام بارگذاری شده و اطلاعات این مجموعه داده با دستور print(dataset.describe()) بدست آمدند:

```
             Initial payment  Last payment  Credit Score  House Number
count        1000.00000      1000.00000    1000.000000   1000.000000
mean          294.34300      12465.88400     528.042000   4027.011000
std           115.81539       1440.15617     259.796059    565.164179
min           100.00000      10005.00000     100.000000   3003.000000
25%           195.00000      11201.50000     302.000000   3545.000000
50%           289.50000      12450.00000     516.500000   4041.500000
75%           398.00000      13678.25000     753.500000   4507.000000
max           500.00000      14996.00000     997.000000   5000.000000
```

سپس با دستور print(dataset.info()) هم مروری بر اطلاعات دیتاست کردیم:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Result           1000 non-null   object
 1   Initial payment  1000 non-null   int64
 2   Last payment     1000 non-null   int64
 3   Credit Score     1000 non-null   int64
 4   House Number     1000 non-null   int64
dtypes: int64(4), object(1)
memory usage: 39.2+ KB
None
```

در ادامه نوع داده‌ها را با دستور print(dataset.dtypes) مشخص کردیم:

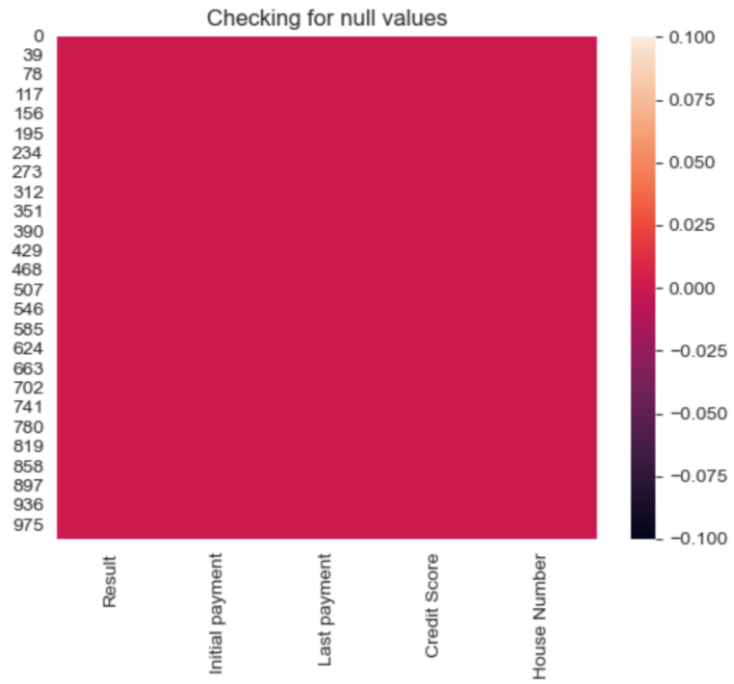
```
Result            object
Initial payment    int64
Last payment       int64
Credit Score       int64
House Number       int64
dtype: object
```

تعداد داده‌ها و ویژگی‌ها را استخراج کردیم.

```
Number of instances is 1000. & Number of features is 5.
```

با دستور print(dataset.isnull().sum()) بررسی کردیم که آیا مقداری miss شده و یا نه.

```
Result            0
Initial payment   0
Last payment      0
Credit Score      0
House Number      0
dtype: int64
```

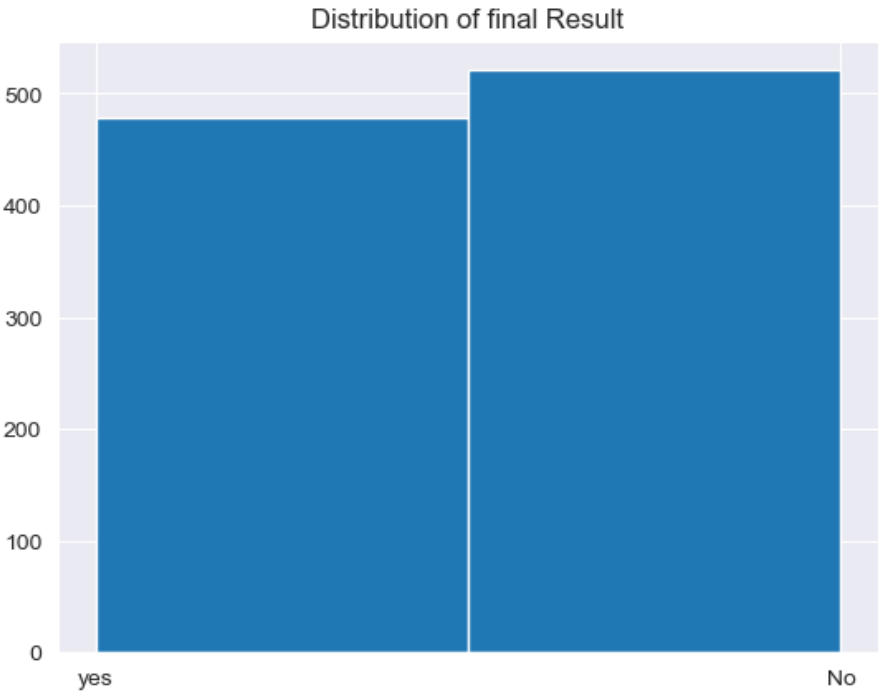


سوال ۴: چرا لازم است مجموعه داده به مجموعه های آموزش و تست تقسیم شود؟

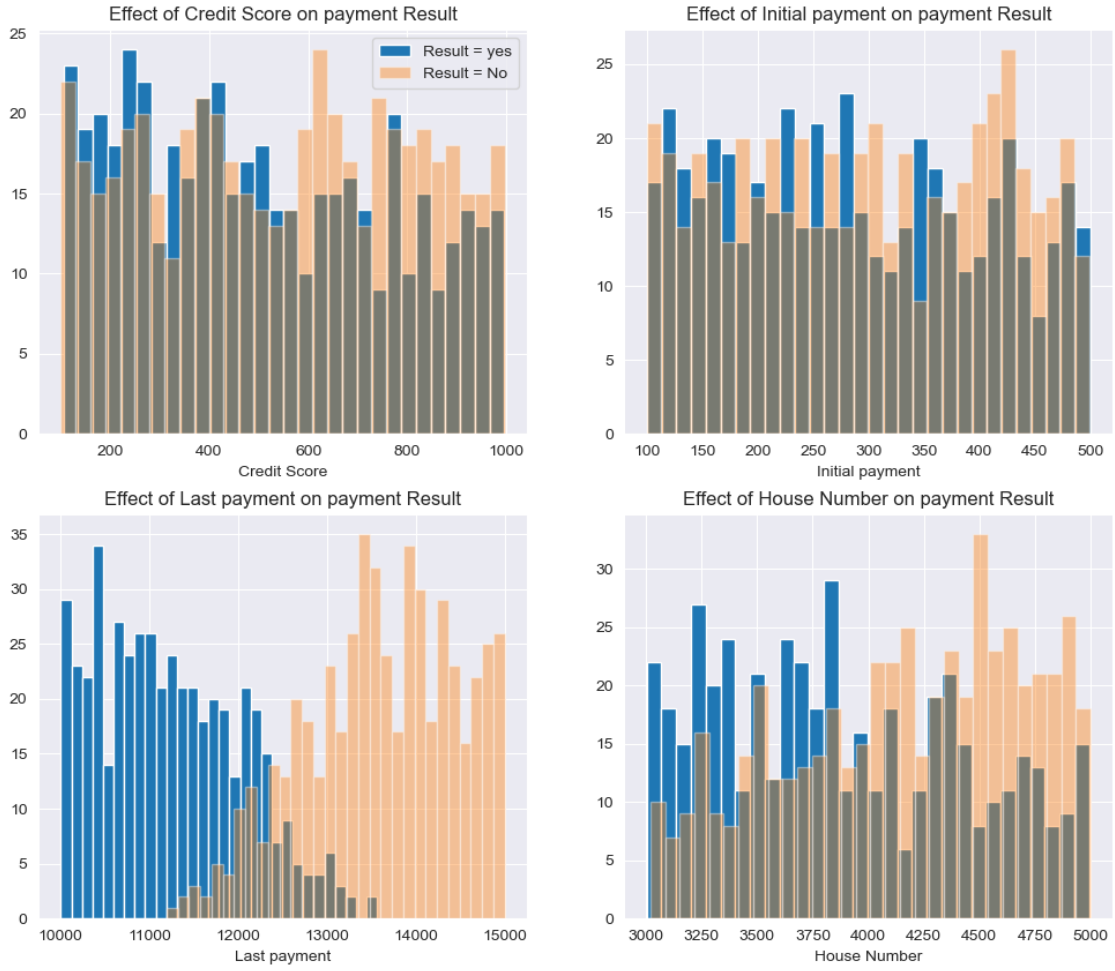
از آنجایی که ممکن است مدل دچار Overfitting شده و داده‌ها را حفظ کند، باید بخشی از داده‌ها را به عنوان داده‌های test جدا کنیم که مدل هنگام آموزش آن‌ها را ندیده باشد. سپس با مابقی داده ها مدل را آموزش داده و نهایتا با این داده‌ها آن را تست می‌کنیم.

Visualization

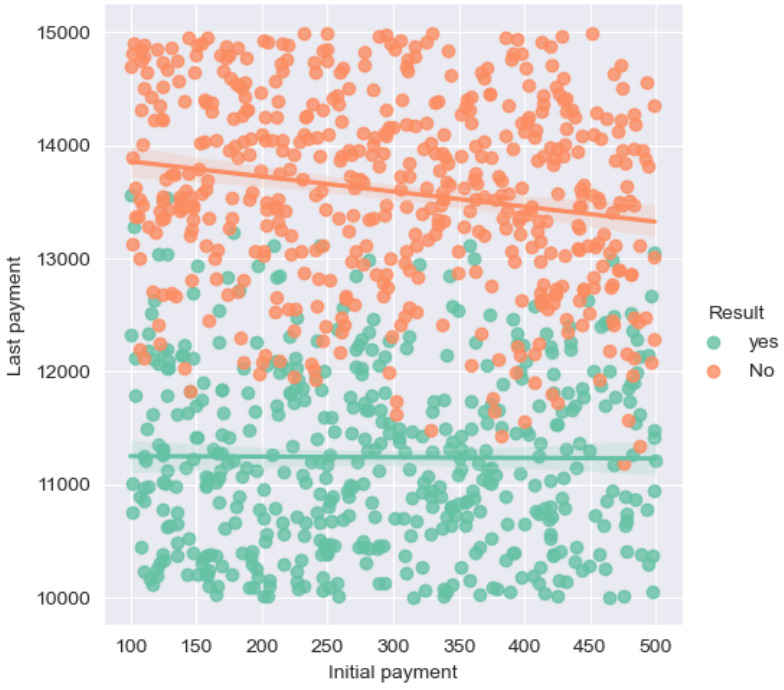
در این بخش به چند صورت داده‌های دیتاست را پردازش کردیم. ابتدا هیستوگرام م ربوط به نسبت کل داده‌های دارای برچسب yes,no را رسم کردیم.



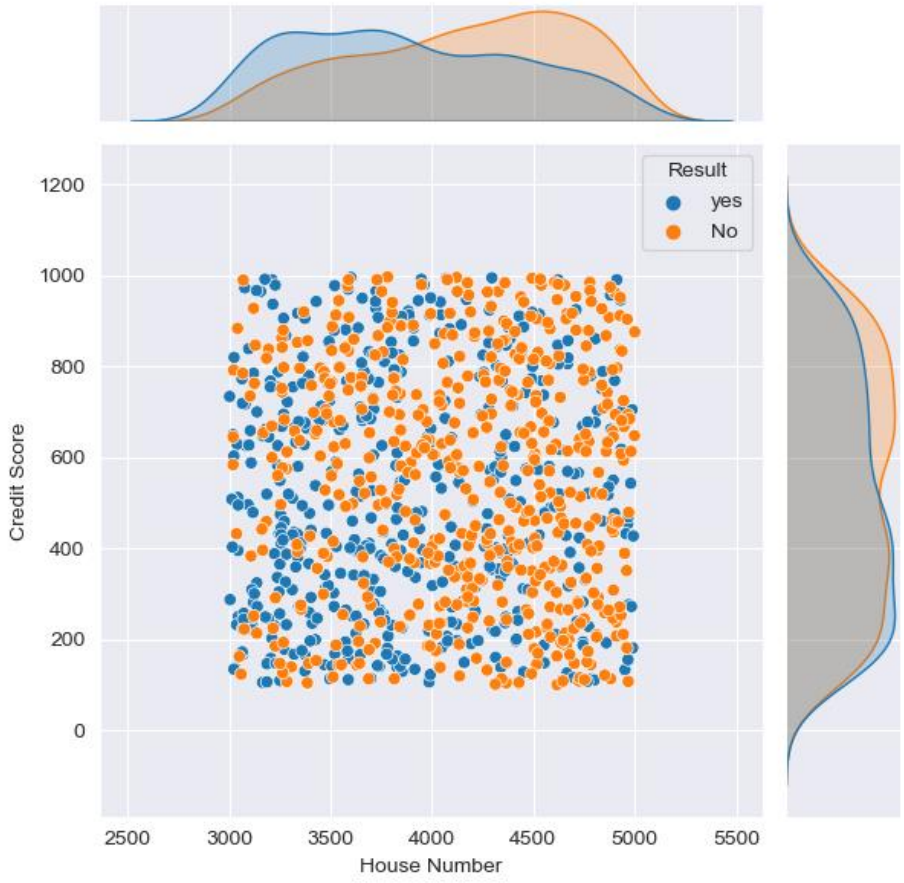
سپس هیستوگرام‌هایی رسم شده که میزان تاثیر هر داده بر روی نتیجه وام دادن را نمایش می‌دهد:



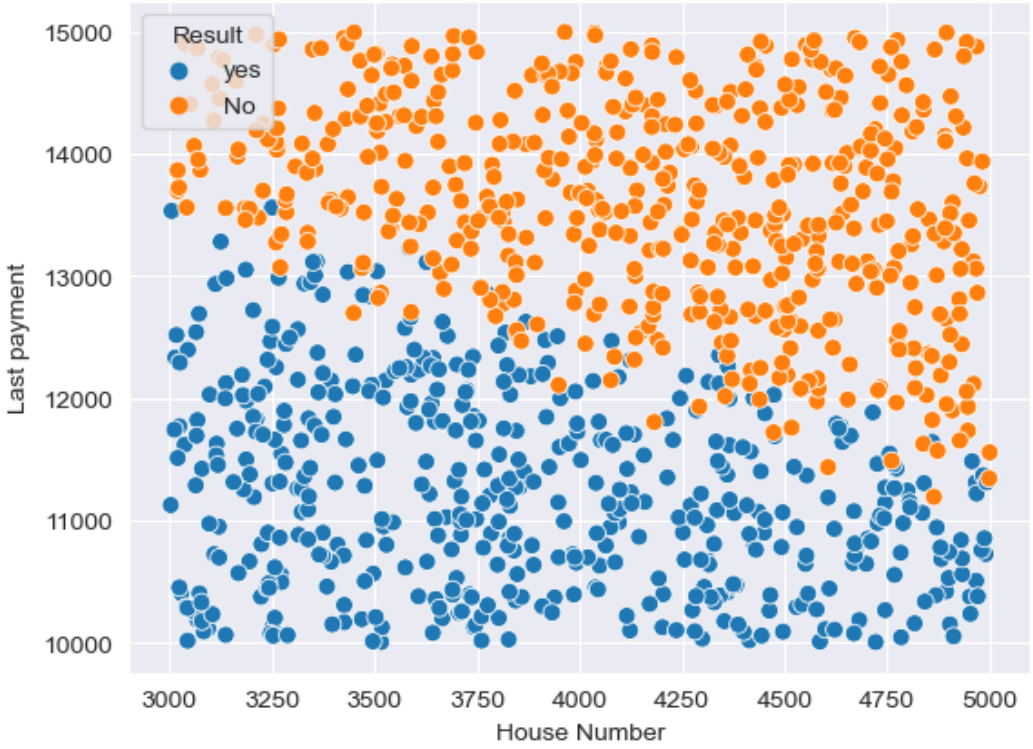
سپس با دستور sns.lmplot() رابطه بین پرداخت اولیه و نهایی را بررسی کردیم:



یک بار هم trend بین House Number و Credit Score بررسی شد:



همینطور Scatter بین House Number و پرداخت آخر ترسیم شد:



نوبت به ساختن مدل می‌رسد. فیچرها به عنوان ورودی و لیبل‌ها به عنوان خروجی تعریف می‌شوند. سپس مجموعه داده را به دو قسمت تست و آموزش تقسیم می‌کنیم و با استفاده از تابع DecisionTreeClassifier() درخت تصمیم خود را می‌سازیم. در این مرحله مدل را به داده‌های آموزش فیت می‌کنیم تا درخت بر اساس داده‌های آموزش شکل بگیرد.

سوال ۵: چرا fit کردن مدل با داده‌های آموزشی ضروری است؟

زیرا از Underfitting جلوگیری کرده و باعث می‌شود داده‌های آموزش به خوبی یاد گرفته شوند و بر اساس آن‌ها درخت ساخته شود.

در بخش Prediction با استفاده از دستور DT.predict() نتیجه را بر روی داده‌های آموزش و تست اعمال کر ده و نتایج را چاپ کردیم.

سوال ۶: خروجی پیش‌بینی‌شده در طبقه‌بندی چه چیزی را نشان می‌دهد؟

مدل با داده‌هایی که تا به حال دیده نشده‌اند آزمایش می‌شود و نتیجه را برای مجموعه داده تست اعلام می‌کند. با بررسی این نتیجه می‌توان دقت حقیقی مدل را بررسی کرد. نتیجه این بخش برای درخت من به شکل زیر است:

```
array(['yes', 'yes', 'No', 'yes', 'No', 'yes', 'yes', 'yes', 'No', 'No',
       'No', 'No', 'yes', 'No', 'No', 'yes', 'yes', 'No', 'yes', 'No',
       'No', 'yes', 'No', 'yes', 'yes', 'No', 'No', 'yes', 'No', 'No',
       'No', 'yes', 'yes', 'yes', 'yes', 'No', 'No', 'No', 'yes', 'No',
       'yes', 'yes', 'yes', 'No', 'No', 'yes', 'yes', 'yes', 'No', 'No',
       'yes', 'No', 'yes', 'yes', 'yes', 'yes', 'No', 'yes', 'No', 'yes',
       'yes', 'No', 'yes', 'yes', 'No', 'yes', 'yes', 'yes', 'No', 'No',
       'No', 'No', 'No', 'yes', 'No', 'yes', 'yes', 'No', 'yes', 'No',
       'No', 'No', 'No', 'yes', 'No', 'yes', 'No', 'yes', 'yes', 'No',
       'yes', 'yes', 'yes', 'yes', 'yes', 'No', 'yes', 'yes', 'yes',
       'yes', 'No', 'No', 'yes', 'yes', 'No', 'yes', 'yes', 'yes', 'No',
       'yes', 'yes', 'yes', 'yes', 'No', 'No', 'yes', 'yes', 'yes', 'No',
       'No', 'No', 'No', 'yes', 'yes', 'No', 'yes', 'yes', 'yes', 'No',
       'No', 'yes', 'yes', 'No', 'yes', 'yes', 'yes', 'No', 'yes', 'yes',
       'No', 'No', 'yes', 'yes', 'yes', 'No', 'yes', 'yes', 'yes', 'yes',
       'No', 'yes', 'No', 'No', 'No', 'No', 'yes', 'No', 'No', 'No',
       'yes', 'yes', 'No', 'yes', 'yes', 'No', 'yes', 'No', 'yes', 'No',
       'No', 'yes', 'yes', 'No', 'yes', 'No', 'yes', 'yes', 'No', 'No',
       'No', 'No', 'No', 'yes', 'No', 'No', 'yes', 'yes', 'yes', 'yes',
       'yes', 'No', 'yes', 'No', 'yes', 'No', 'No', 'No', 'yes', 'yes',
       'No', 'No', 'No', 'yes', 'yes', 'No', 'No', 'yes', 'yes', 'No',
       'No', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes',
       'yes', 'No', 'yes', 'yes', 'No', 'No', 'yes', 'No', 'yes', 'yes',
       'No', 'No', 'yes', 'No', 'yes', 'yes', 'yes', 'yes', 'No', 'No',
       'No', 'No', 'yes', 'yes', 'No', 'yes', 'yes', 'No', 'yes', 'No',
       'No', 'No', 'No', 'yes', 'No', 'No', 'No', 'No', 'No', 'No', 'No',
       'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'No', 'No', 'No', 'No',
       'yes', 'No', 'yes', 'yes', 'yes', 'No', 'No', 'No', 'No', 'yes',
       'No', 'yes', 'No', 'No', 'No', 'yes', 'yes', 'No', 'yes', 'yes',
       'yes', 'No', 'No', 'yes', 'No', 'yes', 'yes', 'No', 'No', 'yes',
       'yes'], dtype=object)
```

در بخش ارزیابی، با معیارهای گفته شده و دستورات زیر، نتایج زیر را گرفتیم:

```
trainAcc = accuracy_score(outPut_train, outPutTrainPred)
testAcc = accuracy_score(outPut_test, outPutTestPred)
```

```
precision = precision_score(outPut_test, outPutTestPred, pos_label = 'yes')
recall = recall_score(outPut_test, outPutTestPred, pos_label = 'yes')
f1score = f1_score(outPut_test, outPutTestPred, pos_label = 'yes')
```

```
Decision tree train/test accuracies: 0.959/0.937
Precision = 0.919
Recall = 0.961
F1 Score = 0.939
```

سوال ۷: معیارهای استفاده شده برای ارزیابی عملکرد طبقه‌بندی چگونه تفسیر می شوند؟

دقت - دقت رایج‌ترین معیار عملکرد است و صرفاً نسبتی از مشاهدات مورد انتظار صحیح به کل مشاهدات است. اگر ما دقت بالایی داشته باشیم، ممکن است فکر کنیم که مدل ما بهترین است اما تنها زمانی که مجموعه داده‌های متقارن با مقادیر تقریباً یکسان مثبت اشتباه و منفی اشتباه داشته باشیم اینطور است.

accuracy = TP+TN/TP+FP+FN+TN

صحت - صحت نسبت مشاهدات مثبتی است که به درستی پیش‌بینی شده‌اند به کل مشاهدات مثبت پیش‌بینی‌شده.

Precision = TP/TP+FP

بازیابی - بازیابی نسبت مشاهدات مثبت پیش‌بینی‌شده درست به همه مشاهدات در کلاس کلی yes است.

recall = TP/TP+FN

F1 Score - F1 Score میانگین وزنی Precision و Recall است. بنابراین، این امتیاز هر دوی مثبت اشتباه و منفی اشتباه را در نظر می‌گیرد. به طور شهودی، درک آن به اندازه دقت آسان نیست، اما به طور کلی F1 مفیدتر از دقت است، به خصوص اگر توزیع نابرابر کلاس داشته باشیم. اگر هزینه‌های مثبت اشتباه و منفی اشتباه یکسان باشد، دقت بهتر عمل می‌کند. اگر هزینه های مثبت اشتباه و منفی اشتباه بسیار متفاوت است، بهتر است هم به Precision و هم Recall نگاه کنیم.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Hyperparameter tuning

در این بخش، مقادیر مختلفی را برای پارامترهای random_state، max_depth و min_samples_leaf درنظر گرفته و با کد

```
clf = GridSearchCV(DT, parameters, n_jobs= 1)
clf.fit(inPut_train, outPut_train)
print(clf.best_params_)
```

ترکیبی که بهترین نتیجه را می‌داد، بدست آوردیم:

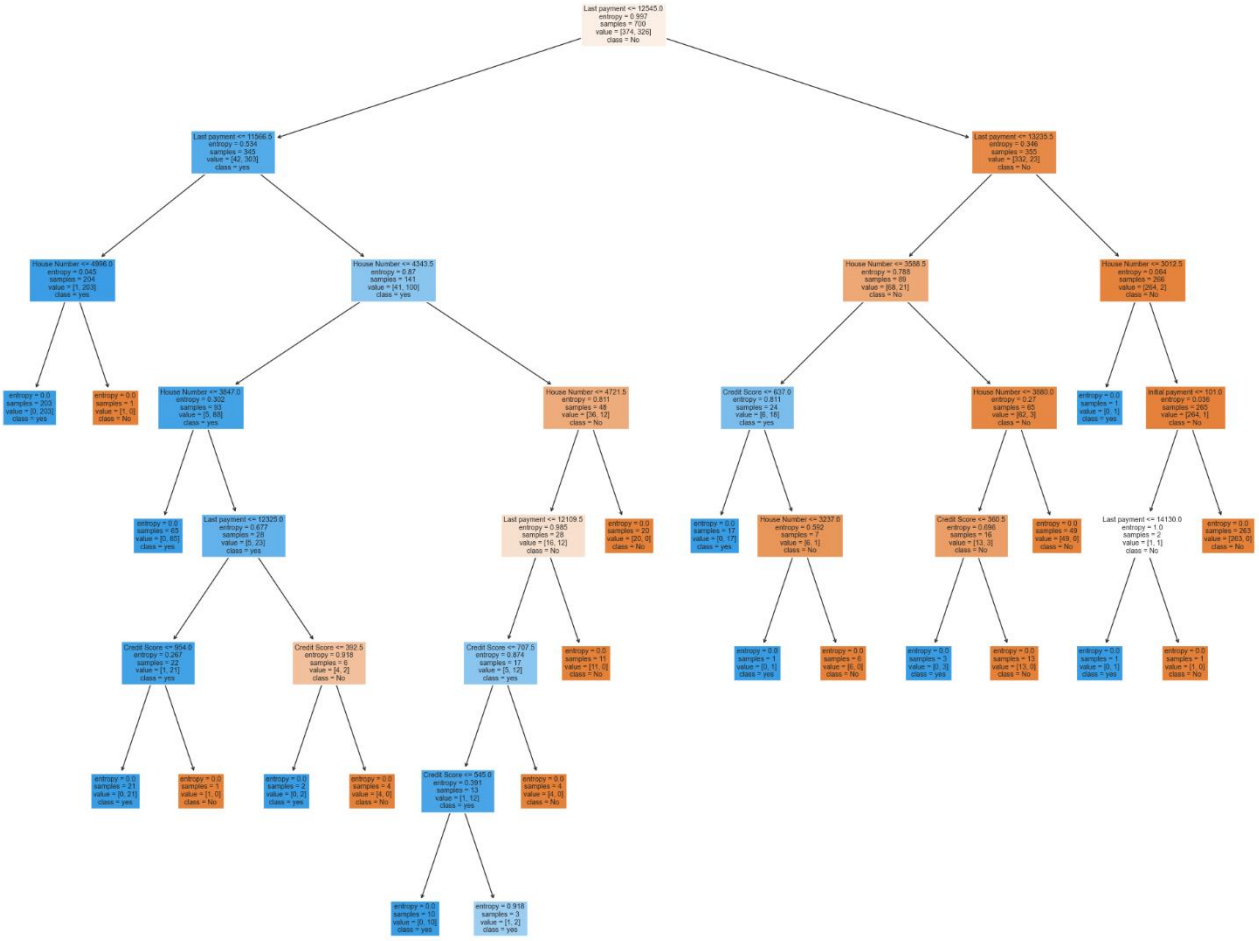
```
{'max_depth': 7, 'min_samples_leaf': 1, 'random_state': 42}

New Decision tree train/test accuracies: 0.999/0.963
New Precision = 0.949
New Recall = 0.980
New F1 Score = 0.965
```

سپس بار دیگر درخت تصمیم را با پارامترهای جدید ساختیم که نتیجه به شرح زیر بود:

```
|--- feature_1 <= 12545.00
|   |--- feature_1 <= 11566.50
|   |   |--- feature_3 <= 4996.00
|   |   |   |--- class: yes
|   |   |--- feature_3 >  4996.00
|   |   |   |--- class: No
|   |--- feature_1 >  11566.50
|   |   |--- feature_3 <= 4343.50
|   |   |   |--- feature_3 <= 3847.00
|   |   |   |   |--- class: yes
|   |   |   |--- feature_3 >  3847.00
|   |   |   |   |--- feature_1 <= 12325.00
|   |   |   |   |   |--- feature_2 <= 954.00
|   |   |   |   |   |   |--- class: yes
|   |   |   |   |   |--- feature_2 >  954.00
|   |   |   |   |   |   |--- class: No
|   |   |   |   |--- feature_1 >  12325.00
|   |   |   |   |   |--- feature_2 <= 392.50
|   |   |   |   |   |   |--- class: yes
|   |   |   |   |   |--- feature_2 >  392.50
|   |   |   |   |   |   |--- class: No
|   |   |--- feature_3 >  4343.50
|   |   |   |--- feature_3 <= 4721.50
|   |   |   |   |--- feature_1 <= 12109.50
|   |   |   |   |   |--- feature_2 <= 707.50
|   |   |   |   |   |   |--- feature_2 <= 545.00
|   |   |   |   |   |   |   |--- class: yes
|   |   |   |   |   |   |--- feature_2 >  545.00
|   |   |   |   |   |   |   |--- class: yes
|   |   |   |   |   |--- feature_2 >  707.50
|   |   |   |   |   |   |--- class: No
|   |   |   |   |--- feature_1 >  12109.50
|   |   |   |   |   |--- class: No
|   |   |   |--- feature_3 >  4721.50
|   |   |   |   |--- class: No
|--- feature_1 >  12545.00
|   |--- feature_1 <= 13235.50
|   |   |--- feature_3 <= 3588.50
|   |   |   |--- feature_2 <= 637.00
|   |   |   |   |--- class: yes
|   |   |   |--- feature_2 >  637.00
|   |   |   |   |--- feature_3 <= 3237.00
|   |   |   |   |   |--- class: yes
|   |   |   |--- feature_3 >  3237.00
|   |   |   |   |   |--- class: No
|   |   |--- feature_3 >  3588.50
|   |   |   |--- feature_3 <= 3880.00
|   |   |   |   |--- feature_2 <= 360.50
|   |   |   |   |   |--- class: yes
|   |   |   |   |--- feature_2 >  360.50
|   |   |   |   |   |--- class: No
|   |   |   |--- feature_3 >  3880.00
```

```
|   |   |   |   |--- class: No
|   |--- feature_1 >  13235.50
|   |   |--- feature_3 <= 3012.50
|   |   |   |--- class: yes
|   |   |--- feature_3 >  3012.50
|   |   |   |--- feature_0 <= 101.00
|   |   |   |   |--- feature_1 <= 14130.00
|   |   |   |   |   |--- class: yes
|   |   |   |   |--- feature_1 >  14130.00
|   |   |   |   |   |--- class: No
|   |   |   |--- feature_0 >  101.00
|   |   |   |   |--- class: No
```



برای مشاهده تصویر واضح‌تر درخت تصمیم به فایل نوت بوک مراجعه شود.