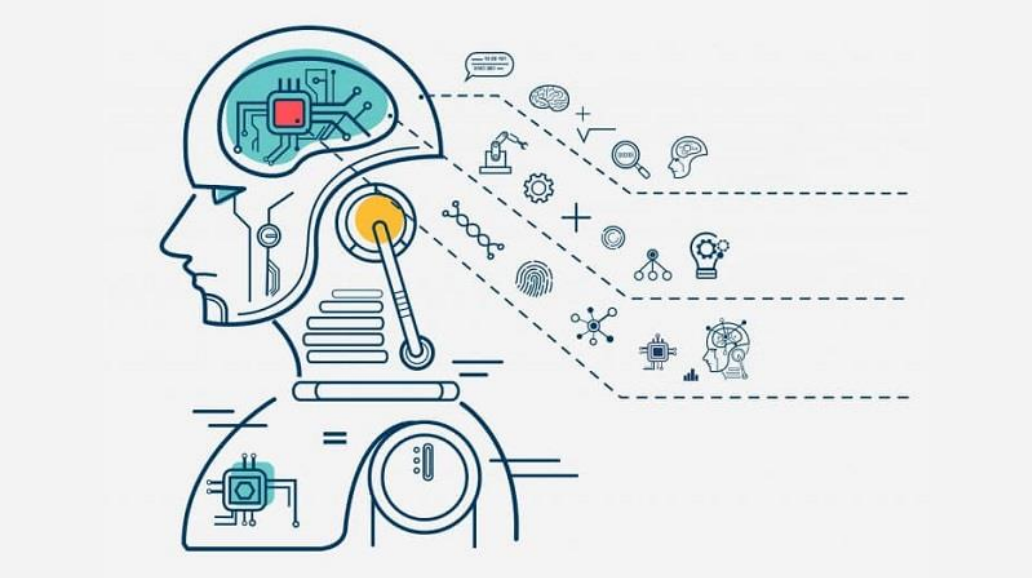
به نام خدا

پروژه چهارم هوش مصنوعی

یادگیری ماشین

نرگس غلامی

۸۱۰۱۹۸۴۴۷



هدف پروژه: هدف از این پروژه آشنایی با روش‌های یادگیری ماشین با کمک کتابخانه sickit-learn می‌باشد

توضیح پروژه: این پروژه در ۴ فاز انجام می‌شود. در فاز صفر بررسی و تجزیه داده‌ها انجام می‌شود و در فاز اول پیش پردازش انجام می‌دهیم سپس در فاز دوم با استفاده از مدل‌های sickit-learn به پیش‌بینی می‌پردازیم و در فاز اخر با روش‌های یادگیری تجمعی آشنا می‌شویم.

فاز صفر:

در این قسمت ابتدا داده‌هایمان را از dataset لود می‌کنیم.

- بررسی ساختار کلی داده‌ها با info

۷ ستون داریم که نوع یکی از آن‌ها عددی می‌باشد و نوع ۶ تای دیگر آن‌ها دسته‌ای می‌باشد. در info تعداد متغیرهای non-null را نیز اعلام می‌کند. در کل برای دریافت یک نمای سریع از دیتاست از این دستور استفاده می‌نماییم.

```
Film data info():
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11059 entries, 0 to 11058
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   type          11059 non-null  object
 1   title         11059 non-null  object
 2   cast          9694 non-null   object
 3   country       8364 non-null   object
 4   release_year  11059 non-null  int64
 5   listed_in     11059 non-null  object
 6   description   11059 non-null  object
dtypes: int64(1), object(6)
memory usage: 604.9+ KB
None
```

- بررسی ساختار کلی داده‌ها با استفاده از describe :

دو نمونه از خروجی تابع ()describe

این تابع یک سری اطلاعات آماری خلاصه از داده ارائه می‌دهد. برای داده‌های عددی مقادیری مانند میانگین، انحراف معبار، چارک اول و ... ذکر می‌شود و برای داده‌های دسته‌ای اطلاعاتی مانند تعداد کل مقادیر، تعداد مقادیر یکتا و... ذکر می‌شود.

```
release_year
count    11059.000000
mean      2014.209603
std          8.959517
min       1925.000000
25%       2013.000000
50%       2017.000000
75%       2019.000000
max       2021.000000
Name: release_year, dtype: float64
```

```
title
count              11059
unique             10957
top       Sister, Sister
freq                   2
Name: title, dtype: object
```

- درصد داده‌های از دست رفته هر ویژگی:

```
type            0.000000
title           0.000000
cast           12.342888
country        24.369292
release_year    0.000000
listed_in       0.000000
description     0.000000
dtype: float64
```

فاز یک:

سوال اول: به بررسی راه‌های مختلف رفع مشکل داده‌های گمشده می‌پردازیم. که دو روش آن را در این قسمت بررسی می‌کنیم:

- پر کردن داده‌ها با میانگین یا میانه و یا مد:

مزایای این روش: جلوگیری از دست دادن داده ها که منجر به حذف سطرها یا ستون ها می شود و همچنین با یک مجموعه داده کوچک به خوبی کار می کند و پیاده سازی آن آسان است.

معایب این روش: این روش واریانس مجموعه داده را تغییر می دهد و در نتیجه ما برآورد کمتری نسبت به واریانس دیتای واقعی داریم .در مقایسه با سایر روش‌ها ضعیف عمل می کند .یکی دیگر از معایب احتمالی استفاده از میانگین برای مقادیر از دست رفته این است که دلیل از دست رفتن مقادیر در مرحله اول می تواند به خود مقادیر گم شده بستگی داشته باشد. به عنوان مثال اگر در سطح شهر بخواهیم یک آزمایش در مورد سلامتی افراد بگیریم افرادی که از سلامتی کمتری برخوردارند به علت عدم علاقه به ابراز آن در آزمایش شرکت نمی‌کنند و اگر ما میانگین را در مقادیر از دست رفته جایگذاری بکنیم پیشبینی مناسبی انجام نداده‌ایم.

- حذف کردن ستون‌های دارای مقادیر گمشده

مزایای این روش: این مدل train با حذف تمام مقادیر از دست رفته، یک مدل قوی ایجاد می کند.

معایب این روش: بسیاری از اطلاعات از دست می‌رود. اگر درصد مقادیر از دست رفته در مقایسه با مجموعه داده کامل، بیش از حد باشد، ضعیف عمل می کند.

در بین دو روش بالا روش پر کردن مقادیر را انتخاب می‌کنیم. برای داده‌های عددی مقادیر را با میانگین پر می‌کنیم و برای داده‌های دسته‌ای مقادیر را با مد، زیرا پر کردن مقادیر بهتر از از دست دادن اطلاعات است.

```
newFilmData = deepcopy(filmData)
numeric_columns = filmData.select_dtypes(include=['number']).columns
newFilmData[numeric_columns] = filmData[numeric_columns].fillna(filmData.mean(numeric_only=True))

Categorical_columns = filmData.select_dtypes(exclude=['number']).columns
newFilmData[Categorical_columns] = filmData[Categorical_columns].transform(lambda a: a.fillna(a.mode()[0]))
```

سوال دوم: اول باید به این موضوع اشاره کرد که الگوریتم‌های مبتنی بر درخت نسبتاً نسبت به ویژگی‌ها حساس نیستند. اگر در مورد آن فکر کنیم، می‌بینیم یک درخت تصمیم تنها یک گره را بر اساس یک ویژگی واحد تقسیم می‌کند. درخت تصمیم یک گره را بر روی یک ویژگی تقسیم می‌کند که همگنی گره را افزایش می‌دهد. این تقسیم در یک ویژگی تحت تأثیر سایر ویژگی‌ها نیست. بنابراین، عملاً هیچ تأثیری از ویژگی‌های باقی‌مانده بر روی تقسیم وجود ندارد. این همان چیزی است که آنها را نسبت به مقیاس ویژگی‌ها تغییرناپذیر می‌کند.با این حال به شرح این دو روش می‌پردازیم و کد آن‌ها را بر روی سال انتشار امتحان می‌کنیم.

Normalization : نرمالیزیشن یک تکنیک scaling می‌باشد که مقادیر شیفت می‌خورند و تغییر مقیاس پیدا می‌کنند تا به یک بازه‌ای بین صفر و یک برسند و اسم دیگر آن Min-Max scaling می‌باشد.

فرمول Normalization : X_{min} و X_{max} به ترتیب مینیموم مقدار و ماکسیمم مقدار مقادیر دیتاست ما هستند.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardization : این نیز یکی دیگر از تکنیک‌های scaling می‌باشد که مقادیر در آن مقادیر حول میانگین با یک انحراف معیار واحد متمرکز می‌شوند. به این صورت میانگین ویژگی‌ها صفر می‌شود و توزیع دارای انحراف معیار واحد است.

این فرمول Standardization است. که μ میانگین و σ انحراف معیاراست.

$$X' = \frac{X - \mu}{\sigma}$$

حال به بررسی این بپردازیم کدام روش باید انتخاب شود.

استانداردسازی زمانی که با ویژگی (Features) و داده‌هایی با مقیاس‌های مختلف سروکار دارید بسیار مهم است.

نرمال سازی برای زمانی مناسب است که متغیرهای ما توزیع گاوسی نداشته باشند و خوب است که دامنه‌ی ویژگی‌ها بین صفر و یک باشد. برای مثال، در داده‌های تصویری به دلیل اینکه دامنه پیکسل‌های رنگی بین صفر تا ۲۵۵ است، نرمال سازی بهتر از استانداردسازی است.

همانطور که درابتدا ذکر شد هیچکدام از تغییر مقیاس‌های بالا مناسب پروژه‌ی ما نمی‌باشد ولی در کل اگر مناسب بود نرمال‌سازی انتخاب بهتری می‌بود. البته یک کار خوب دیگر این است که هر دو روش امتحان شود و دیده شود که کدام مناسب‌تر است.

فعلا در این مرحله یک داده‌ی عددی در اختیار داریم که بر روی آن نرمال سازی و استاندارد سازی را طبق تکه کد روبرو امتحان می‌کنیم.

در نرمال‌سازی همان‌طور که مشاهده می‌شود داده‌ها به بازه صفر و یک مپ شده‌اند و در استاندارد سازی میانگین داده‌ها صفر (در محاسبات نزدیک صفر شده است که احتمالا به علت ذخیره محدود اعشار اعداد بوده است) و همچنین انحراف معیار برابر یک است.

```
norm = MinMaxScaler().fit(filmData[['release_year']])
x_train_norm = norm.transform(filmData[['release_year']])
x_train_norm
```

✓ 0.5s

```
array([[0.98958333],
       [1.        ],
       [1.        ],
       ...,
       [0.92708333],
       [0.94791667],
       [0.86458333]])
```

```
standard = StandardScaler().fit(filmData[['release_year']])
x_train_standard = standard.transform(filmData[['release_year']])
print(x_train_standard)
print(np.mean(x_train_standard))
print(np.std(x_train_standard))
```

✓ 0.3s

```
[[ 0.64631375]
 [ 0.75793196]
 [ 0.75793196]
 ...
 [-0.02339552]
 [ 0.19984091]
 [-0.69310478]]
-1.2014783577822473e-14
1.0
```

سوال سوم: **از روش‌هایی که می‌توانیم داده‌های دسته‌ای را به داده‌های عددی نگاشت بکنیم. چهار مورد را توضیح می‌دهیم.**

Label Encoding : Label Encoding به تبدیل برچسب‌ها به شکل عددی اشاره دارد تا آنها را به فرم قابل خواندن ماشین تبدیل کند.

سپس الگوریتم‌های ماشین لرنینگ می‌توانند به روشی بهتر تصمیم بگیرند که این برچسب‌ها چگونه باید کار کنند.

Ordinal Encoding: در رمزگذاری ترتیبی، به هر مقدار دسته منحصر به فرد یک مقدار صحیح اختصاص داده می شود و این مدل مخصوص داده‌های ترتیبی است. مثلا شنبه : ۱، یکشنبه :۲،دوشنبه : ۳،

این روش رمزگذاری ترتیبی یا رمزگذاری عدد صحیح نامیده می شود و به راحتی قابل برگشت است. غالباً از اعداد صحیح استفاده می شود و از صفر شروع می‌شود. برای برخی از متغیرها، ordinal encoding ممکن است کافی باشد. این مقادیر صحیح، یک رابطه منظم طبیعی بین یکدیگر دارند و الگوریتم های یادگیری ماشین ممکن است بهتر قابلیت درک این رابطه را داشته باشند.

One-Hot Encoding: برای متغیرهای دسته‌ای که هیچ رابطه ترتیبی وجود ندارد، رمزگذاری ترتیبی ممکن است در بهترین حالت کافی نباشد یا در بدترین حالت برای مدل گمراه کننده باشد. اینجاست که متغیر رمزگذاری شده ترتیبی حذف می شود و یک متغیر باینری جدید برای هر دسته‌ی منحصر به فرد در متغیر اضافه می شود.

هر بیت نشان دهنده یک دسته بندی ممکن است. اگر متغیر نمی تواند به طور همزمان به چندین دسته تعلق داشته باشد، تنها یک بیت در گروه می تواند "روشن" باشد. به این کدگذاری one-hot می‌گویند.

Dummy Encoding : طرح Dummy Encoding شبیه به کدگذاری one-hot است. این روش رمزگذاری مانند قبلی، داده‌های دسته‌ای را به مجموعه‌ای از متغیرهای باینری تبدیل می کند. در روش one-hot ، برای N دسته در یک متغیر، از N متغیر باینری استفاده می‌کند. رمزگذاری ساختگی پیشرفت کوچکی نسبت به رمزگذاری one-hot دارد و از N-1 ویژگی برای نشان دادن N دسته استفاده می کند.

در این پروژه از label encoding استفاده می‌کنیم. زیرا با توجه به حجم بالای داده‌های یکتای برنامه تعداد بیتی که برای one-hot نیاز داریم بسیار بالاست و زمانی محاسبه هم بالا می‌برد. از آن جایی که داده‌های ما ترتیبی ندارند از ordinal encoding استفاده نمی‌کنیم.

از تکه کد زیر برای label encoding استفاده شد:

```
newFilmData['type'] = newFilmData[['type']].apply(preprocessing.LabelEncoder().fit_transform)
newFilmData['country'] = newFilmData[['country']].apply(preprocessing.LabelEncoder().fit_transform)
```

سوال چهارم: من در این قسمت سه ایده را مطرح می‌کنم.

ایده‌ی اول که از همه ساده‌تر است این است که برای فیلم‌هایی که بیش از یک ژانر دارند بقیه ژانرها جز ژانر اول حذف شود و یا با یک الگوریتمی پرتکرارترین ژانر را که همه جز آن پاک شوند و آن باقی بماند. این ایده، ایده‌ی مناسبی نمی‌باشد زیرا تصمیم‌گیری بر اساس همه‌ی ژانرها بهتر است.

ایده دوم ایده‌ی اضافه کردن ستون‌هاست. از آنجایی که تعداد ژانرها از سه تا تجاوز نمی‌کند ستون ژانر را به سه ستون تقسیم می‌کنیم (ژانر یک، ژانر دو و ژانر سه) و برای آن‌هایی که کمتر از سه ژانر دارند، ژانر اضافی را NULL می‌گذاریم.

ایده‌ی سوم ایده‌ی اضافه کردن سطرهاست. فرضا همان فیلم را با همان اطلاعات(جز ژانر) دوباره به ردیف‌هایمان اضافه بکنیم ولی ژانر هر سطر متفاوت باشد.

روش انتخابی من روش دوم می‌باشد. دو ستون تحت عنوان listed_in2 و listed_in3 اضافه کردم و ژانرهای بعدی را در آن‌ها ریختم و بعد هر کدام از ژانرها را لیبل زدم.

```
listed_in_data = newFilmData['listed_in'].str.split(',')
newFilmData['listed_in2'] = None
newFilmData['listed_in3'] = None
for i in range(len(newFilmData)):
    newFilmData['listed_in'][i] = listed_in_data[i][0]
    if len(listed_in_data[i]) > 1:
        newFilmData['listed_in2'][i] = listed_in_data[i][1]
    if len(listed_in_data[i]) > 2 :
        newFilmData['listed_in3'][i] = listed_in_data[i][2]
```

```
newFilmData['listed_in'] = newFilmData[['listed_in']].apply(preprocessing.LabelEncoder().fit_transform)
newFilmData['listed_in2'] = newFilmData[['listed_in2']].apply(preprocessing.LabelEncoder().fit_transform)
newFilmData['listed_in3'] = newFilmData[['listed_in3']].apply(preprocessing.LabelEncoder().fit_transform)
```

استخراج ویژگی از متن:

برای بدست آوردن این مورد از tf-idf استفاده شد. به این صورت که اگر امتیاز بیشتر ۰.۷ بود به لیست ویژگی‌ها اضافه شود.

در مجموع از description و title ۴۹ ویژگی انتخاب شد که به شرح زیر است:

```
vectorizer = CountVectorizer(analyzer='word',
                             token_pattern=r'\b[a-zA-Z]{3,}\b',
                             ngram_range=(1, 2), min_df = 10)
x = newFilmData['description'] + newFilmData['title']
count_vectorized  = vectorizer.fit_transform(x)
tfidf_transformer = TfidfTransformer(smooth_idf=True, use_idf=True)
vectorized = tfidf_transformer.fit_transform(count_vectorized)
pd.DataFrame(vectorized.toarray(),
             index=['sentence '+str(i)
                    for i in range(1, 1+len(x))],
             columns=vectorizer.get_feature_names_out())

featureNames = vectorizer.get_feature_names_out()
feature = []
for col in vectorized.toarray():
       for i in range((len(col))):
              if col[i] > 0.7:
                     if featureNames[i] not in feature:
                            feature.append(featureNames[i])
```

```
['chicago', 'sugar', 'christmas', 'anger', 'freedom', 'hitler', 'rock', 'trucks', 'physics', 'season', 'sheep', 'short', 'rainbow', 'restaurant',
'monkey', 'joe', 'del', 'knock', 'global', 'bear', 'las', 'machine', 'names', 'cats', 'test', 'before', 'amazon', 'gear', 'act', 'series', 'fishing',
'impact', 'chicken', 'junior', 'agent', 'zoo', 'adventures', 'dinosaurs', 'luna', 'marvel', 'animal', 'hawaii', 'toys', 'fish', 'mickey', 'rocket',
'moments', 'cartoon', 'spider']
49
```

برای بدست آوردن این مورد از tf-idf استفاده شد. به این صورت که اگر امتیاز بیشتر ۰.۷ بود به لیست ویژگی‌ها اضافه شود.

همچنین از بازیگران ۱۰ ویژگی انتخاب شد:

```
vectorizer = CountVectorizer(analyzer='word',
                             token_pattern=r'\b[a-zA-Z]{3,}\b',
                             ngram_range=(2, 2), min_df = 10 , max_features = 10 )
vectorized = vectorizer.fit_transform(newFilmData['cast'])
actors = pd.DataFrame(vectorized.toarray(),
             index=['sentence '+str(i)
                    for i in range(1, 1+len(newFilmData['cast']))],
             columns=vectorizer.get_feature_names())

actNames = vectorizer.get_feature_names_out()
newFilmData = newFilmData.join(actors.reset_index(drop = True))
del newFilmData["cast"]
actNames
```

```
array(['anupam kher', 'david attenborough', 'juan pablo', 'julie tejwani',
       'naseeruddin shah', 'rukh khan', 'rupa bhimani', 'shah rukh',
       'takahiro sakurai', 'yuki kaji'], dtype=object)
```

این ویژگی‌ها با این منطق انتخاب شدند که اگر در بیشتر از ۸ فیلم مختلف حضور داشتند به ویژگی‌هایمان اضافه می‌شدند.

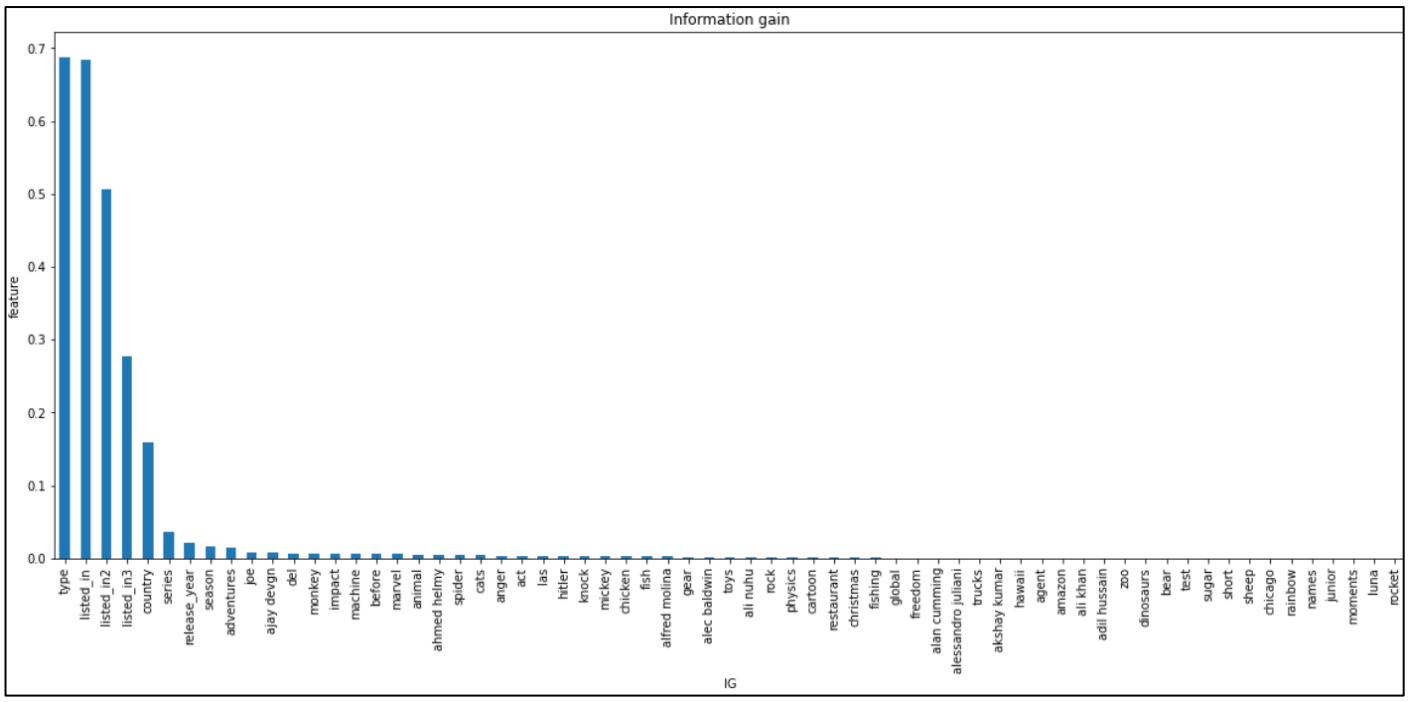
حال این چند ویژگی که انتخاب شدند را به ستون ویژگی‌هایمان اضافه می‌کنیم.

ممکن است استخراج ویژگی اضافه خیلی به یادگیری ستون هدف کمک نکند و از طرفی نیز ممکن است خطا را هم بیشتر بکند زیرا انگاری داده‌ها با داده‌های یادگیری fit می‌شوند و در داده‌های تست به خوبی عمل نمی‌کنند. در حقیقت overfitting اتفاق می‌افتد پس لزوما هر چه ویژگی بیشتر انتخاب شود به نفع train نمی‌باشد.

بررسی روابط بین ویژگی‌ها

ابتدا با استفاده از کتابخانه sklearn و تابع mutual_info_classifier ، information gain داده‌ها را بدست می‌آوریم و بعد آن‌ها را پلات می‌کنیم. همان‌طور که مشاهده می‌شود، تایپ با ژانر یک وابستگی بیشتری دارد.

```
target = newFilmData['type']
newFilmData.drop(columns = 'type', axis = 1, inplace=True)
gainInfo = mutual_info_classif(newFilmData, target)
gainInfo = pd.Series(mutual_info_classif(newFilmData, target))
gainInfo.index = newFilmData.columns
gainInfo.sort_values(ascending=False).plot.bar(figsize=(20, 8))
plt.ylabel("feature")
plt.xlabel("IG")
plt.title("Information gain")
```



این نمودار در ادامه به ما این کمک را می‌کند که متوجه بشویم که هر کدام از این داده‌ها چقدر به حدس ستون هدف ما (ستون تایپ) کمک می‌کند. هر چه information gain بیشتر باشد به این معناست که آن ویژگی کمک کننده‌تر است. همچنین بر اساس ترتیبی که این نمودار دارد متوجه می‌شویم که چگونه باید decision tree خود را بسط بدهیم. هر چه متغیری information gain بیشتری داشته باشد در نقطه بالاتری بسط داده می‌شود.

فاز دو:

متغیر max_depth نشان‌دهنده‌ی حداکثر عمق درخت می‌باشد. اگر None باشد، گره‌ها تا زمانی که همه برگ‌ها خالص شوند(به نتیجه‌ی مشخص برسند) یا تا زمانی که همه برگ‌ها کمتر از min_samples_split نمونه‌ها داشته باشند، گسترش می‌یابند. در مورد تاثیر عمق بر دقت در سوال دو توضیح بیشتر داده می‌شود.

متغیر min_samples_split حداقل تعداد نمونه مورد نیاز برای تقسیم یک گره داخلی می‌باشد.

کد درخت تصمیم:

```
def decisiionTree(newFilmData, t, target):
    y = target
    X_train, X_test, y_train, y_test = train_test_split(newFilmData, y, test_size= t, random_state=1)
    clf = DecisionTreeClassifier().fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_trainPred = clf.predict(X_train)
    print("Accuracy of test is " , accuracy_score(y_test, y_pred), " for test size", t)
    print("Accuracy of train is " , accuracy_score(y_train, y_trainPred))
```

سوال اول:داده ها را به نسبت ۳۰ به ۷۰ تقسیم کردم. زیرا از طرفی اطلاعات کم دادن به داده‌ی train باعث underfitting می‌شود چون داده‌ی ورودی ما اطلاعات زیادی برای یادگیری ندارد. حالتی که داده‌ها را ۴۰ به ۶۰ تقسیم می‌کنیم به این صورت است.

همچنین اطلاعات زیاد دادن به آن باعث overfitting می‌شود زیرا انگار داده‌ها خود را با داده‌ی یادگیری ست می‌کنند، مثل حالتی که داده‌ها را ۲ به ۹۸ می‌دهیم.

در نتیجه کاری که باید انجام شود یک تقسیم معقول بین train و تست می‌باشد.

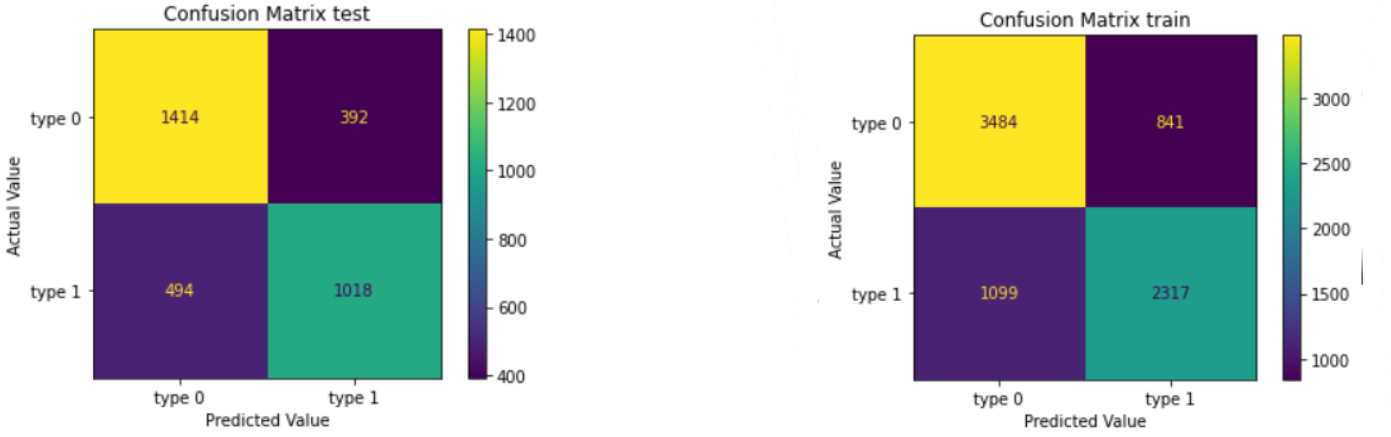
خروجی برنامه به ازای تست سایزهای مختلف:

```
Accuracy of test is  0.990990990990991  for test size 0.01
Accuracy of train is  1.0
Accuracy of test is  0.9881167656936192  for test size 0.7
Accuracy of train is  1.0
Accuracy of test is  0.9948764315852924  for test size 0.3
Accuracy of train is  1.0
```

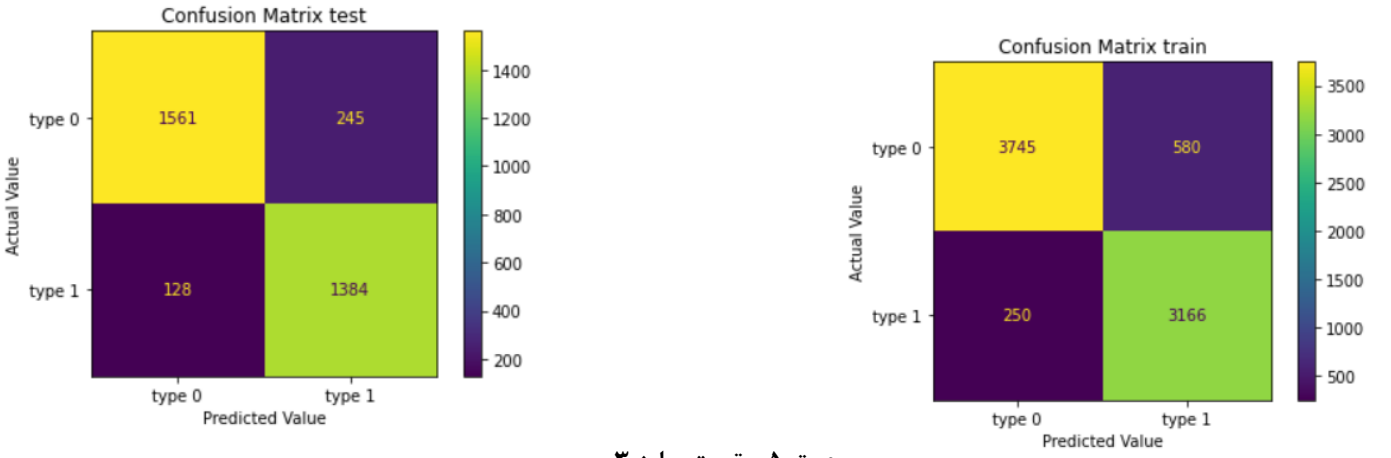
همانطور که مشاهده می‌شود برای سایز ۰.۰۱ برای سایز تست، داده‌ها overfit می‌شوند و در نتیجه دقت پایین می‌آید. وقتی هم داده کمی برای یادگیری داشته باشیم نیز underfitting اتفاق می‌افتند. سایز ۰.۳ نسبت به بقیه دقت بیشتری دارد.

در زیر confusion-matrixهای زیر برای داده‌های با عمق ۵ و به ترتیب تست سایز ۰.۸ و ۰.۰۱ و دوباره به ترتیب برای داده‌ی تست و سپس داده‌ی ترین رسم شده است. می‌دانیم دقت هر دوی این مدل‌ها کم می‌باشد. حال به بررسی confusion-matrix می‌پردازیم.

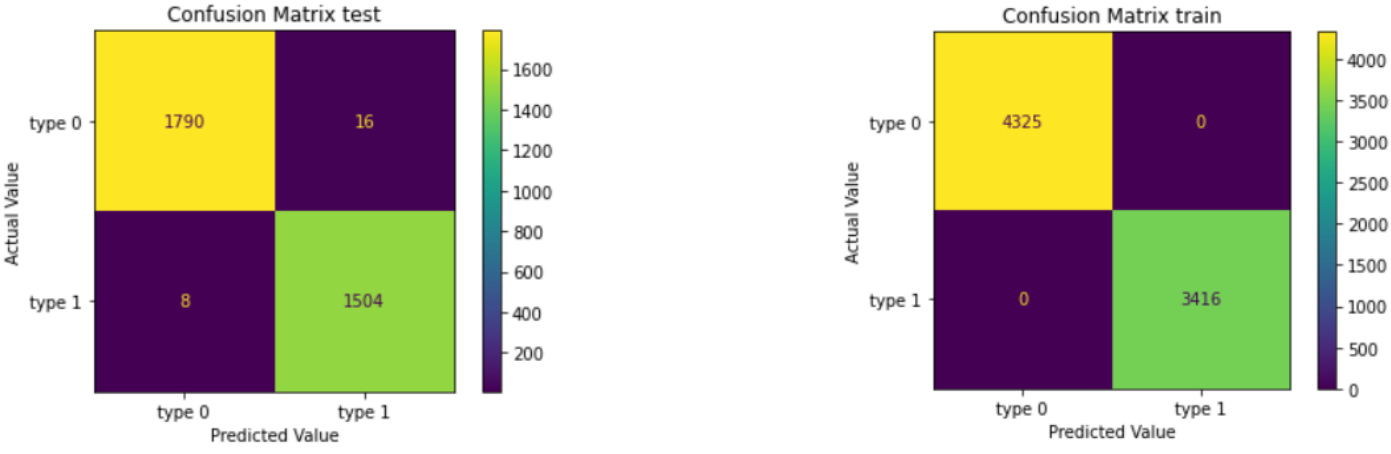
آن اعدادی که در ستون صفر و ردیف صفر هستند به معنای آن داده‌هایی هستند که به درستی در دسته صفر تشخیص داده شده‌اند. آن عددی که در ردیف یک و ستون یک هست نیز به همین صورت است. ولی مثلا عددی که در ردیف صفر و ستون یک است به معنای این است که در واقع در دسته‌ی صفر بوده‌اند ولی به اشتباه در دسته یک جایگذاری شده‌اند. ردیف یک و ستون صفر نیز به این صورت است که در اصل متعلق به دسته یک بوده است ولی در صفر تشخیص داده شده‌اند.



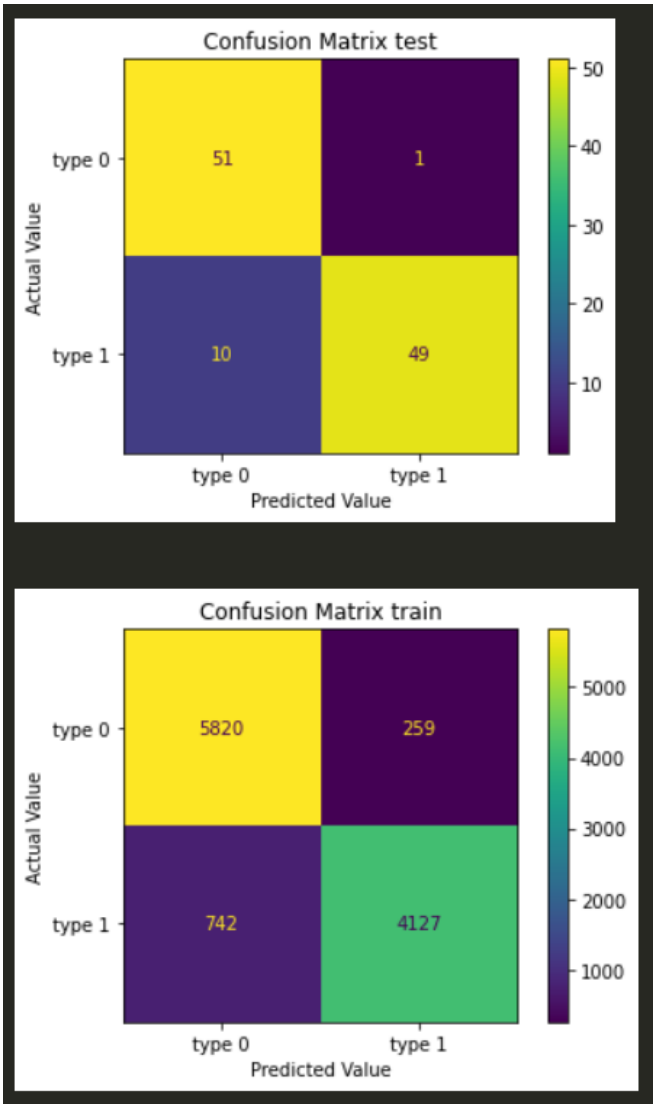
عمق دو و تست سایز ۰.۳



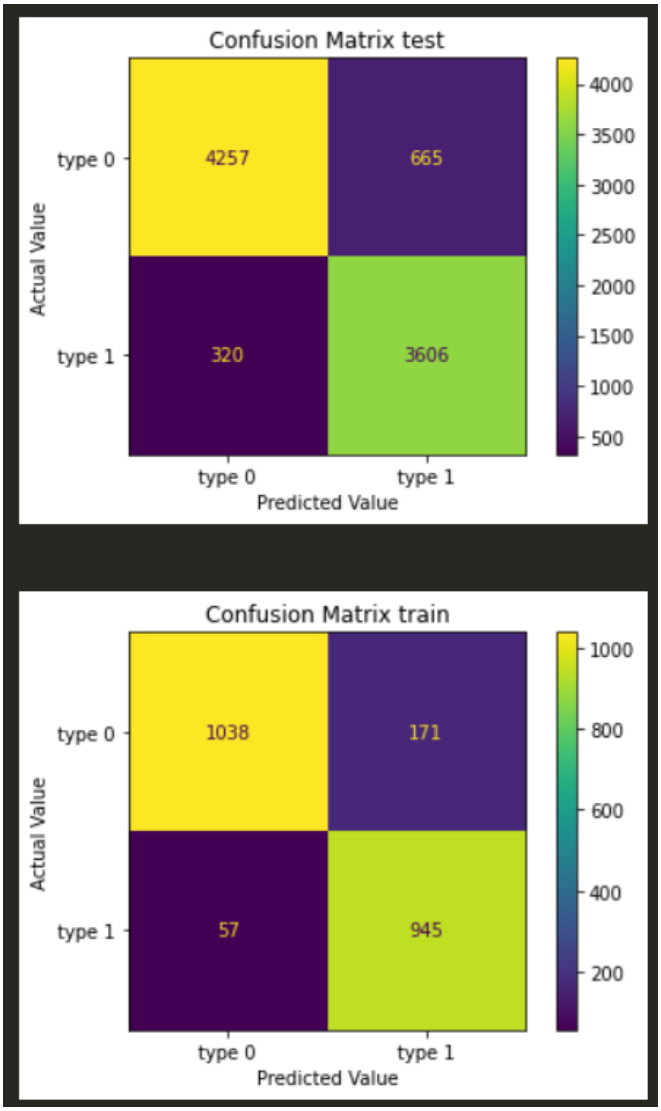
عمق ۵ و تست سایز ۰.۳



عمق ۱۵ و تست سایز ۰.۳



عمق ۵ و تست سایز ۰.۰۱



عمق ۵ و تست سایز ۰.۸

کد این قسمت از برنامه:

```
def pltConfusionMatrix(y_test, y_pred , str):
    matrix = confusion_matrix(y_test, y_pred)
    clf = ConfusionMatrixDisplay(matrix, display_labels=['type 0', 'type 1'])
    clf.plot()
    clf.ax_.set(title='Confusion Matrix ' + str,
                xlabel='Predicted Value',
                ylabel='Actual Value')
    plt.show()

def decisiionTreeMatrix(newFilmData, depth, tSize, target):
    y = target
    X_train, X_test, y_train, y_test = train_test_split(newFilmData, y, test_size= tSize, random_state=1)
    clf = DecisionTreeClassifier(max_depth = depth).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_trainPred = clf.predict(X_train)
    pltConfusionMatrix(y_test, y_pred, 'test')
    pltConfusionMatrix(y_train, y_trainPred, 'train')
```

سوال دوم: اگر max_depth را خیلی زیاد بکنیم خطر overfitting داریم و اگر عمق درخت کم باشد خطر underfitting داریم و دقت در هر دو حالت پایین می‌آید.

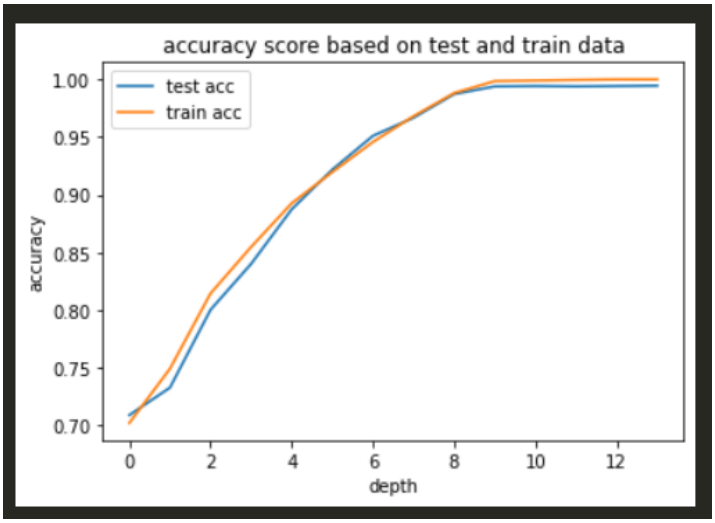
نتیجه‌های مختلف را در روبرو مشاهده می‌کنید:

```
Accuracy of test is  0.7091621458710067 for drpth  1
Accuracy of train is  0.7023640356543083
Accuracy of test is  0.7329716696805304 for drpth  2
Accuracy of train is  0.7493863841880894
Accuracy of test is  0.8004822182037372 for drpth  3
Accuracy of train is  0.8146234336649012
Accuracy of test is  0.8402652200120555 for drpth  4
Accuracy of train is  0.8549283038367136
Accuracy of test is  0.8875828812537673 for drpth  5
Accuracy of train is  0.8927787107608836
Accuracy of test is  0.9219409282700421 for drpth  6
Accuracy of train is  0.9197778064849502
Accuracy of test is  0.9511754068716094 for drpth  7
Accuracy of train is  0.945872626275675
Accuracy of test is  0.9677516576250753 for drpth  8
Accuracy of train is  0.9682211600568402
Accuracy of test is  0.9882459312839059 for drpth  9
Accuracy of train is  0.9882444128665547
Accuracy of test is  0.9948764315852924 for drpth  10
Accuracy of train is  0.9985789949618912
Accuracy of test is  0.9945750452079566 for drpth  11
Accuracy of train is  0.999095724066658
Accuracy of test is  0.9942736588306209 for drpth  12
Accuracy of train is  0.9996124531714249
Accuracy of test is  0.9927667269439421 for drpth  13
Accuracy of train is  1.0
Accuracy of test is  0.9927667269439421 for drpth  14
Accuracy of train is  1.0
```

همان‌طور که مشاهده می‌شود رفته رفته دقت برای داده‌ی یادگیری زیاد می‌شود و دقت برای داده‌ی تست هم تا یک جایی زیاد می‌شود و بعد کمی افت می‌کند.

علت overfitting این است که درخت را با عمق زیاد بررسی می‌کنیم و underfitting این است که درخت را با عمق کم پیش‌بینی می‌کنیم.

نمودار این دو معیار را در کنار هم مشاهده می‌کنید:



همچنین با استفاده از تابع gridsearchCV مقداردقت بهینه بین ورودی‌های این تابع مشخص شد که به ازای عمق ۴۰ و min_samples_split برابر با ۲ بود.

```
def findBestHyper(target):
    X_train, X_test, y_train, y_test = train_test_split(newFilmData, target, test_size= 0.3, random_state=1)
    clf = GridSearchCV(DecisionTreeClassifier(),
                       {'min_samples_split': np.arange(2, 3, 4),
                       'max_depth': [5, 10, 20, 30, 40, 50]},
                       cv = 4, return_train_score = False, scoring = 'accuracy')
    clf.fit(X_train, y_train)
    pd.DataFrame(clf.cv_results_).head(10)
    accuracy = clf.best_score_
    print(accuracy)
    print(clf.best_params_)

findBestHyper(target)
```

✓ 0.5s

```
0.9937996508424626
{'max_depth': 40, 'min_samples_split': 2}
```

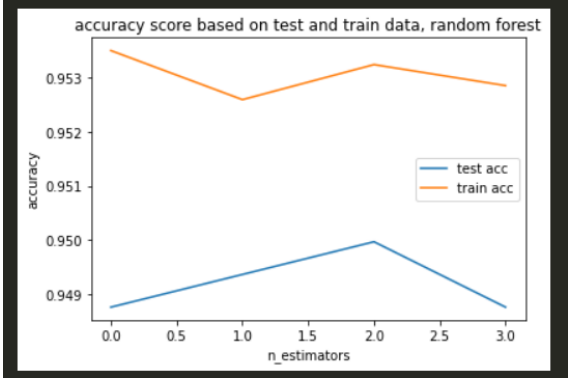
فاز سه:

صحبت در مورد هایپرپارامترهای random forest:

n_estimators: این تعداد درختانی است که می خواهید قبل از حداکثر رای‌گیری یا میانگین پیش بینی‌ها بسازید.

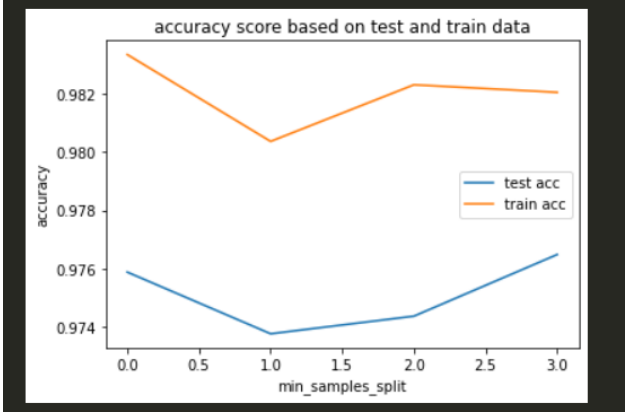
به ترتیب برای داده‌های روبرو *n_estimators* ۵۰، ۱۰۰ و ۱۵۰ و ۲۰۰ در نظر گرفته شده است. همان‌طور که مشاهده می‌شود دقت بیشتر متعلق به ورودی ۱۵۰ (ایندکس ۲) است.

```
Accuracy of test is  0.9487643158529234
Accuracy of train is  0.9534943805709857
Accuracy of test is  0.9493670886075949
Accuracy of train is  0.9525901046376437
Accuracy of test is  0.9499698613622665
Accuracy of train is  0.9532360160186022
Accuracy of test is  0.9487643158529234
Accuracy of train is  0.9528484691900271
```



min_samples_split: در درس اهمیت حداقل اندازه برگ نمونه را متوجه شدیم. برگ، گره انتهایی درخت تصمیم است. یک برگ کوچکتر مدل را مستعد گرفتن نویز در داده های یادگیری می کند و باید بهینه‌ترین حالت ممکن را برای درخت پیدا کرد.

```
Accuracy of test is  0.9758890898131405
Accuracy of train is  0.9833354863712699
Accuracy of test is  0.9737793851717902
Accuracy of train is  0.9803642940188606
Accuracy of test is  0.9743821579264618
Accuracy of train is  0.9823020281617362
Accuracy of test is  0.976491862567812
Accuracy of train is  0.9820436636093528
```



در تست روبرو حالت‌های مختلف با min_samples_split های مختلف رسم شده است. Min_samplest_split = 3 بعد از Min_samplest_split = 0 تقریبا بیشترین دقت را داراست.

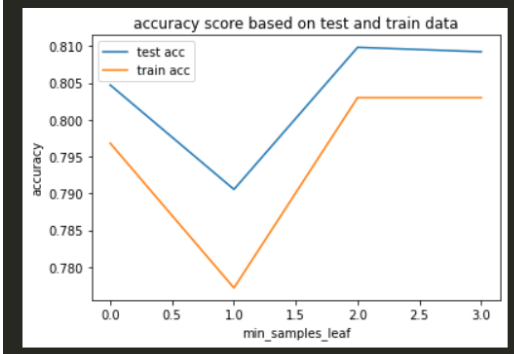
(این قسمت از کد پاک شده است)

:min_samples_leaf

یکی دیگر از hyper parameter ها min_samples_leaf است که حداقل تعداد نمونه لازم برای قرار گرفتن در یک گره برگ را نشان می‌دهد.

در تست روبرو حالت‌های مختلف با **min_samples_leaf** های مختلف رسم شده است. **min_samples_leaf** = 3 بیشترین دقت را داراست.

```
Accuracy of test is  0.8047016274864376
Accuracy of train is  0.7967962795504456
Accuracy of test is  0.7905364677516576
Accuracy of train is  0.7771605735693063
Accuracy of test is  0.8098251959011453
Accuracy of train is  0.8029970288076476
Accuracy of test is  0.8092224231464737
Accuracy of train is  0.8029970288076476
```



کد random forest :

```python
def RandomForest(newFilmData, n, m, target):
    X_train, X_test, y_train, y_test = train_test_split(newFilmData, target, test_size=0.3, random_state=42)
    clf = RandomForestClassifier(n_estimators = n, min_samples_leaf = m).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_trainPred = clf.predict(X_train)
    trainList = accuracy_score(y_test, y_pred)
    testList = accuracy_score(y_train, y_trainPred)
    print("Accuracy of test is " , trainList)
    print("Accuracy of train is " , testList)
    return trainList, testList
```

مقایسه نتایج با decision tree :

نتایج decision tree نسبت به random forest بهتر بود.

توضیح قسمت آخر:

درک واریانس، بایاس در decision tree مهم است. در decision tree ، اگر درخت کم عمق باشد، ممکن است بایاس بالایی داشته باشد، (underfitting اتفاق بیفتد) از طرفی اگر درخت خیلی عمیق باشد، واریانس بالایی خواهد داشت، (over fitting اتفاق می‌افتد) بنابراین برای دستیابی به یک تریدآف واریانس، بایاس خوب در decision tree ، باید مقدار پارامترهای فوق را تنظیم کنیم تا مدل مناسب شود.

از طرف دیگر در random forest تکنیکی وجود دارد که در آن تعدادی درخت تصمیم با هم ترکیب می‌شوند تا نتیجه را به دست آورند. این فرآیند ترکیبی از bootstrapping و aggregation است. ایده اصلی پشت random forest این است که تعداد زیادی از درختان با واریانس بالا و بایاس کم با هم ترکیب می شوند تا جنگل واریانس با بایاس کم را ایجاد کنند. از آنجایی که بر روی درختان مختلف توزیع می شود و هر درخت مجموعه متفاوتی از داده ها را می بیند، بنابراین در جنگل تصادفی به طور کلی overfitting رخ نمی‌دهد. و از آنجایی که آن‌ها از درختان کم بایاس نیز ساخته شده اند، underfitting نیز اتفاق نمی افتد.

بنابراین برای برآورده شدن این تریدآف random forest مدل بهتری می‌باشد.

اگر از معیار دقت بخواهیم بررسی بکنیم در این پروژه نتیجه این بوده است که decision tree بهتر تصمیم گرفته است .

نتیجه گیری کلی:

در حالت کلی استفاده از decision tree برای تصمیم‌گیری و پیش‌بینی، راحت‌تر و سریع‌تر است، ولی خطر overfitting و underfitting داریم که با انتخاب مقادیر مناسب می‌توانیم احتمال این دو اتفاق کم بکنیم. از طرف دیگر رندوم فارست را داریم که این دو خطر برایشان کمتر اتفاق می‌افتد زیرا ویژگی‌ها به صورت رندوم انتخاب می‌شوند و دائما امتحان می‌شوند. همچنین رندوم فارست برای داده‌های بزرگ مناسب‌تر است.

ارائه راهکارهایی برای توسعه و بهبود پروژه:

خسته نباشید دست شما درد نکنه.

منابع:

https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e

https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization

https://blog.faradars.org/standardization-and-normalization-in-python

https://towardsdatascience.com/normalization-vs-standardization-which-one-is-better-f29e043a57eb

https://sokanacademy.com/blog/%D8%AF%D8%B1%D8%A2%D9%85%D8%AF%DB%8C-%D8%A8%D8%B1-%D8%A2%D9%85%D8%A7%D8%B1-%D8%A8%D8%A7-%D8%A7%D8%B3%D8%AA%D9%81%D8%A7%D8%AF%D9%87-%D8%A7%D8%B2-%D9%84%D8%A7%DB%8C%D8%A8%D8%B1%D8%B1%DB%8C-numpy-%D9%88-%D8%B2%D8%A8%D8%A7%D9%86-%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87%D9%86%D9%88%DB%8C%D8%B3%DB%8C-python

/https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python

https://datascience.stackexchange.com/questions/14324/handling-a-feature-containing-multiple-values

https://www.datacamp.com/community/tutorials/decision-tree-classification-python

https://towardsdatascience.com/entropy-and-information-gain-in-decision-trees-c7db67a3a293

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

/https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm