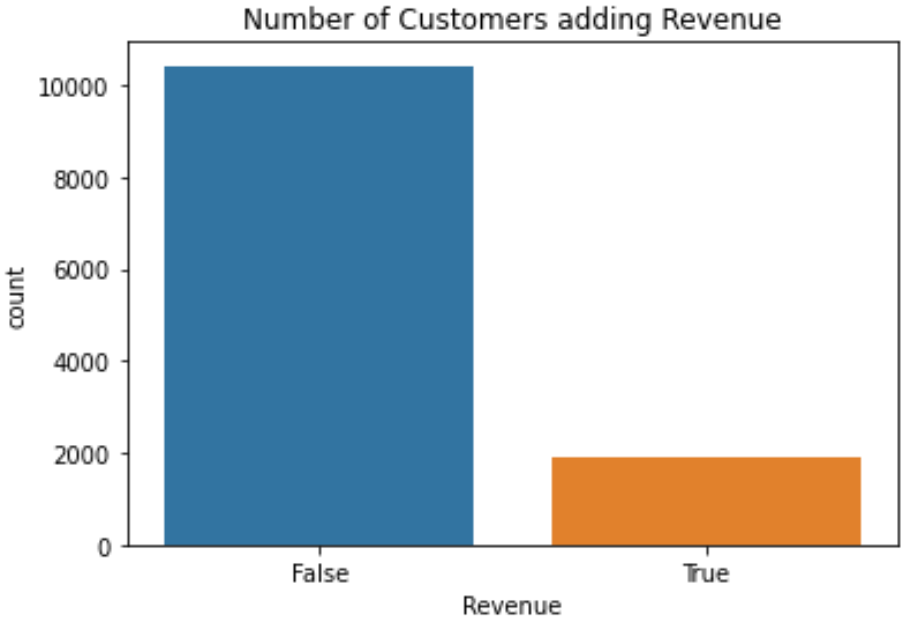
به نام خدا

گزارش تمرین ۷

علی عدالت ۸۱۰۱۹۹۳۴۸

ابتدا به بررسی مجموعه داده می پردازیم. در زیر توزیع مقادیر ممکن برای لیبل هر سطر آمده است. لیبل ستون Revenue است که قصد داریم برای داده دیده نشده پیش بینی کنیم. در این تمرین با یک داده نامتوازن برای طبقه بندی مواجه هستیم.



همانطور که دیده می شود، تعداد نمونه یک کلاس حداقل ۵ برابر تعداد نمونه کلاس دیگر است. مقدار Revenue برابر ۱ یا true تعدادی برابر ۲۰۰۰ نمونه دارد. کلاس دیگر یا مقدار دیگر Revenue تعدادی بیش از ۱۰۰۰۰ نمونه دارد. این عدم توازن می تواند فرآیند یادگیری مدل را دچار مشکل کند. می‌تواند باعث بایاس مدل شود به طوری که همیشه مقدار صفر پیش بینی کند. برای انجام طبقه بندی در این شرایط ما نیاز به متریک های ارزیابی مناسب برای حالت داده نامتوازن داریم. اهمیت استفاده از معیارهای مناسب این است که بایاس مدل در پیش بینی یک کلاس

بیشتر از دیگری باید در مقدار معیار دیده شود. بر این اساس می توان به میزان خوب بودن عملکرد مدل در واقعیت پی برد. برای معیارهای ارزیابی ما از accuracy، f1-score، roc auc و neg log loss استفاده می‌کنیم. معیارهای roc auc و f1-score برای ارزیابی چنین دیتاست‌هایی هستند. معیار f1 نوعی میانگین دو معیار recall و precision است. Recall به false negative و precision به false positive اهمیت می‌دهد. در مسئله ما هر دو به یک اندازه اهمیت دارند. یک مدل خوب مدلی است که دو معیار recall و precision برای آن بالا باشد. معیار f1-score بر اساس این دو معیار است و بالا بودن آن نشان دهنده عملکرد بهتر مدل است. نکته مهم این است که بر اساس این تعریف اگر مدل به دلیل عدم توازن دارای بایاس در پیش بینی باشد، مقدار f1-score پایین خواهد بود. پس این معیار، بایاس ناشی از عدم توازن داده را منعکس می‌کند و عملکرد واقعی مدل را نشان می دهد و عدم توازن باعث گمراهی آن نمی‌شود. معیار f1-score مقدار f1 را برای کلاس ۱ که تعداد کمتری دارد نشان می دهد. دلیل انتخاب آن این است که مدلی مناسب است که بتواند در پیش بینی کلاس کمتر در عدم توازن، عملکرد خوبی داشته باشد. این معیار بر اساس شمارش موارد درست و غلط در پیش بینی با توجه به لیبل واقعیت است. معیار roc auc یک مقدار مثبت کمتر از یک است. اگر این مقدار برابر 0.5 باشد یعنی مدل عملکردی برابر مدلی دارد که به صورت رندم پیش بینی می‌کند. هر چه این مقدار نزدیک تر به یک باشد، یعنی مدل ما بهتر است. عدم توازن روی مقدار این معیار تاثیری ندارد و این معیار در هر شرایطی عملکرد واقعی مدل را نشان می دهد. این معیار راهی برای مرتب کردن مدل‌ها بر اساس عملکرد است. معیار دیگر که یک Probabilistic Metric است، معیار neg log loss یا log loss است. معیارهای احتمالی به طور خاص برای تعیین کمی میزان عدم قطعیت در پیش‌بینی‌های طبقه‌بندی‌کننده طراحی شده‌اند. اینها برای مشکلاتی مفید هستند که در آن کمتر به پیش‌بینی‌های نادرست در مقابل صحیح کلاس علاقه‌مندیم و بیشتر به عدم قطعیت مدل در پیش‌بینی‌ها و جریمه کردن آن پیش‌بینی‌هایی که اشتباه هستند اما بسیار مطمئن هستند علاقه‌مندیم. در مورد مجموعه داده های نامتعادل حساس است. ممکن است مجبور شوید وزن‌های کلاسها را به گونه این قرار دهید که خطاهای کلاس اقلیت را بیشتر جریمه کند یا می‌توانید پس از

متعادل کردن مجموعه داده خود از آن استفاده کنید. چون ما از SMOTE برای متوازن کردن استفاده می‌کنیم، این معیار حساسیت کمتری خواهد داشت و مقادیر معنا دار تری را نشان می‌دهد و می‌توان از آن استفاده کرد. معیار دیگر accuracy است که همه می‌شناسند. این معیار در حالت عدم توازن ممکن است عملکرد را به درستی نشان ندهد.

همچنین برای مقابله با این عدم توازن باید تعداد نمونه کلاس با تعداد کمتر را افزایش دهیم. برای این کار ما از SMOTE استفاده می‌کنیم. یک رویکرد برای حل مشکل مدل‌ها با مجموعه داده های نامتعادل، نمونه برداری بیش از حد از طبقه اقلیت است. ساده‌ترین روش شامل کپی کردن مثال‌ها در کلاس اقلیت است، این مثال‌ها هیچ اطلاعات جدیدی به مدل اضافه نمی‌کنند و کمک کننده نخواهند بود. در عوض، نمونه های جدید را می توان از نمونه های موجود ترکیب کرد. نمونه‌های جدیدی بر اساس موارد موجود و ترکیب آنها ساخت. این یک نوع افزایش داده برای کلاس اقلیت است و به عنوان تکنیک Synthetic Minority Oversampling Technique یا به اختصار SMOTE شناخته می‌شود.

در اینجا به بررسی تایپ ویژگی‌ها در دیتاست می‌پردازیم. برای این موضوع به صورت زیر عمل می‌کنیم.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

همانطور که دیده می‌شود، تمام ویژگی‌ها به جز دو مورد ماه و نوع تماشاگر visitotType عددی هستند. در زیر تعداد nan های ویژگی‌ها را می‌بینیم.

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

همانطور که دیده می‌شود، هیچ مقدار nan نداریم. در زیر به توسشف داده‌های عددی می‌پردازیم و تعدادی آماره را بر ویژگی محاسبه می‌کنیم.

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468
std	3.321784	176.779107	1.270156	140.749294	44.475503
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000
50%	1.000000	7.500000	0.000000	0.000000	18.000000
75%	4.000000	93.256250	0.000000	0.000000	38.000000
max	27.000000	3398.750000	24.000000	2549.375000	705.000000

ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	OperatingSystems
12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
1194.746220	0.022191	0.043073	5.889258	0.061427	2.124006
1913.669288	0.048488	0.048597	18.568437	0.198917	0.911325
0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
184.137500	0.000000	0.014286	0.000000	0.000000	2.000000
598.936905	0.003112	0.025156	0.000000	0.000000	2.000000
1464.157213	0.016813	0.050000	0.000000	0.000000	3.000000
63973.522230	0.200000	0.200000	361.763742	1.000000	8.000000

Browser	Region	TrafficType
12330.000000	12330.000000	12330.000000
2.357097	3.147364	4.069586
1.717277	2.401591	4.025169
1.000000	1.000000	1.000000
2.000000	1.000000	2.000000
2.000000	3.000000	2.000000
2.000000	4.000000	4.000000
13.000000	9.000000	20.000000

ویژگی‌های مربوط به Informational، Administration و product نشان دهنده انواع صفحه هستند. این ویژگی ها داده nominal هستند. با توجه به نوع این ویژگی‌ها برای پر کردن جای خالی باید از مقدار میانه استفاده کرد. مقدار صفر برای این ویژگی‌ها به معنی تهی بودن است که می‌توان معادل nan در نظر گرفت. در زیر تبدیل صفر به nan انجام شده است. مقدار کمینه این ویژگی ها صفر است که یعنی مواردی از این نوع nan ها دارند.

```
for cols in ['Administrative','Informational','ProductRelated']:
    df[cols].replace(0, np.nan, inplace= True)
```

در مدت زمان های مربوط به انواع صفحه، ما 0 را به عنوان حداقل مقدار در مدت زمان داریم (زمان نمی تواند صفر باشد). زمانی اتفاق می‌افتد که نوع صفحه 0 باشد که قبلاً آن را خالی در نظر گرفتیم، بنابراین می‌توانیم آن را به NaN تبدیل کرده و آن را نیز خالی در نظر بگیریم. در اینجا لازم نیست نگران نرخ های صفر باشیم. به عنوان مثال نرخ پرش دارای مقادیر 0 است، زیرا بسیاری از موارد وجود دارد که نرخ پرش می تواند 0 باشد زیرا کاربر باید وب سایت را دوست داشته باشد و به سایر صفحات وب برای انجام معاملات رفته باشد تا نرخ پرش داشته باشیم. در بیشتر موارد ممکن نیست یا با احتمال کم رخ می‌دهد. برای تبدیل زمان‌های صفر به صورت زیر عمل می‌کنیم.

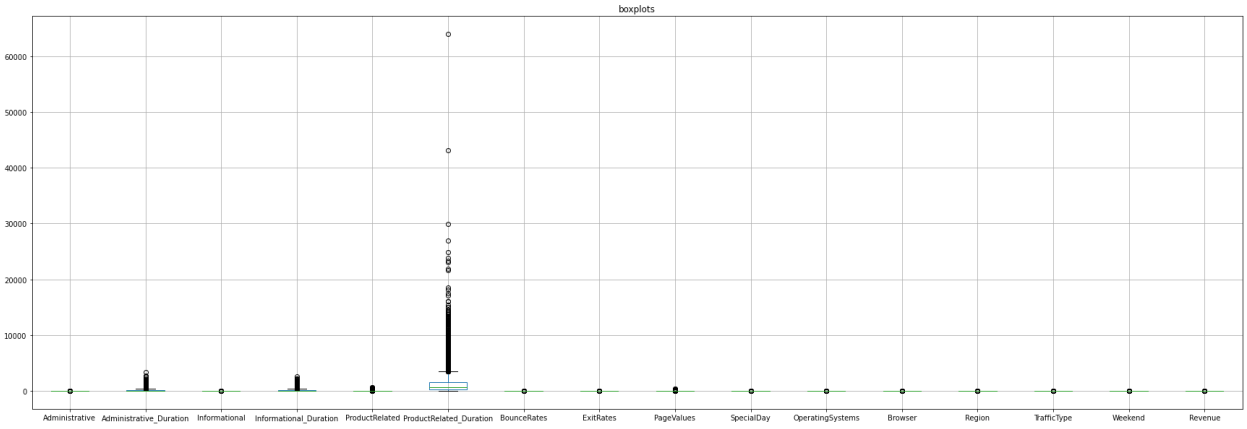
```
for cols in ['Administrative_Duration','Informational_Duration','ProductRelated_Duration']:
    df[cols].replace(0, np.nan, inplace= True)
```

تعداد nan ها بعد از این تبدیلات به صورت زیر خواهد شد. مشکلات دیگری در داده نداریم.

```
Administrative             5768
Administrative_Duration    5903
Informational              9699
Informational_Duration     9925
ProductRelated               38
ProductRelated_Duration     755
BounceRates                   0
ExitRates                     0
PageValues                    0
SpecialDay                    0
Month                         0
OperatingSystems              0
Browser                       0
Region                        0
TrafficType                   0
VisitorType                   0
Weekend                       0
Revenue                       0
dtype: int64
```

بیایید ویژگی های عددی را از نظر داشتن داده پرت و چولگی در توزیع بوسیله‌ی باکس پلات بررسی کنیم. همانطور که دیده می‌شود در تمام باکس پلات ها ما نقطه مشکی یعنی داده پرت داریم. همچنین توزیع تمام ویژگی ها دارای چولگی است. برخی ویژگی‌ها مقدار چولگی بالایی دارند. این موارد نشان می‌دهد که توزیع ویژگی‌های دارای nan نرمال نیست و برای پر کردن باید از میانه استفاده کنیم. فقط داده‌های عددی nan دارند. برای پر کردن جاهای

خالی می‌توان از یک راه دیگر هم استفاده کرد. از رگرسیون بر اساس دیگر ویژگی‌ها غیر از لیبل برای پر کردن استفاده کنیم.




```
df.skew()
```

```
Administrative             1.586495
Administrative_Duration    4.577785
Informational              2.669395
Informational_Duration     3.457944
ProductRelated             4.339583
ProductRelated_Duration    7.209611
BounceRates                2.947855
ExitRates                  2.148789
PageValues                 6.382964
SpecialDay                 3.302667
OperatingSystems           2.066285
Browser                    3.242350
Region                     0.983549
TrafficType                1.962987
Weekend                    1.265962
Revenue                    1.909509
```

ما ویژگی‌های bool داشتیم که به int تبدیل می‌کنیم. همچنین ویژگی های غیر عددی را به شکل زیر عددی می‌کنیم. از labelencoder استفاده می کنیم.

```
df["Revenue"] = df["Revenue"].astype(int)
df["Weekend"] = df["Weekend"].astype(int)
```

We have encoded the required features

```
from sklearn.preprocessing import LabelEncoder

encoded_features=['Month','VisitorType']

label_encoder = LabelEncoder()
for col in encoded_features:
    df[col] = label_encoder.fit_transform(df[col])
```

برای پر کردن به کمک رگرسیون به شکل زیر عمل می کنیم.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

from sklearn.linear_model import LinearRegression

lr = LinearRegression()

trainx = df.drop(['Revenue'], axis=1)
ty = df['Revenue']

train_col_names = list(trainx.columns.values.tolist())

mice_imputer = IterativeImputer(random_state=42, estimator=lr,
                                max_iter=10, n_nearest_features=4, imputation_order = 'roman')
trainx = mice_imputer.fit_transform(trainx)

trainx = pd.DataFrame(trainx)

trainx.columns = train_col_names
```

برای پر کردن با میانه هم به روش زیر عمل می کنیم.

```
for cols in ['Administrative','Informational','ProductRelated']:
    median_value=trainx[cols].median()
    trainx[cols]=trainx[cols].fillna(median_value)
```

```
for cols in ['Administrative_Duration','Informational_Duration','ProductRelated_Duration','BounceRates','ExitRates']
    mean_value=trainx[cols].median()
    trainx[cols]=trainx[cols].fillna(mean_value)
```

برای پیدا کردن بهترین روش تمیز کردن و پر کردن جاهای خالی ما نیاز به مقایسه این دو روش داریم. در روش استفاده از رگرسیون از لیبل استفاده نمی کنیم چرا که باعث می‌شود اطلاعات لیبل‌های داده تست آینده در ویژگی هایی از آموزش دیده شوند. عملا باعث می شود که لیبل داده تست را ببینیم. برای مقایسه از طبقه بندی بر اساس داده‌های حاصل استفاده می کنیم. هر بار از یک مدل با یک سری پارامتر برای طبقه بندی استفاده می کنیم. قبل از طبقه بندی از اسکیل ویژگی ها و SMOTE به شکل پایپ لاین استفاده می کنیم. اسکیل باعث می شود که بازه تمام ویژگی ها در یک رنج قرار گیرد که از overfit جلوگیری می کند. SMOTE مشکل متوازن نبودن را حل می‌کند. نکته مهم این است که SMOTE از یک سری نمونه موجود برای تولید نمونه استفاده می کند. ممکن است از داده تست برای تولید داده برای آموزش استفاده شود. برای جلوگیری از این اتفاق از پایپ لاین استفاده می‌کنیم که داده تست و آموزش را می تواند کامل جدا کند. برای بدست آوردن عملکرد مدل از 5-fold برای cross validation استفاده می‌کنیم. در زیر روش طبقه بندی آمده است. برای مقایسه باید نتایج دو روش را ببینیم. در زیر نتایج به ترتیب آمده است. اولی برای استفاده از رگرسیون و دومی برای استفاده از میانه است.

```
0.884 accuracy with a standard deviation of 0.025
0.651 f1-score with a standard deviation of 0.075
0.901 auc with a standard deviation of 0.039
-0.308 neg_log_loss with a standard deviation of 0.035
```

```
0.882 accuracy with a standard deviation of 0.029
0.653 f1-score with a standard deviation of 0.084
0.903 auc with a standard deviation of 0.044
-0.303 neg_log_loss with a standard deviation of 0.051
```

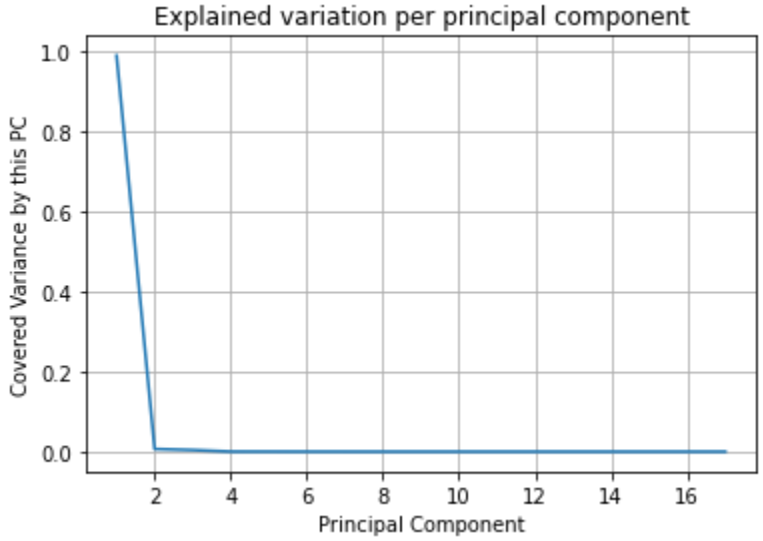
```
sm = SMOTE( random_state=101) #,sampling_strategy='minority', k_neighbors=5)
# X_svm_smote, y_svm_smote = svm_smote.fit_resample(trainx,ty)

# # X_train_svm, X_test_svm, y_train, y_test = train_test_split(X_svm_smote,y_svm_smote, test_size=0.20, random_st

sc = StandardScaler()
# X_train = sc.fit_transform(X_svm_smote)
```

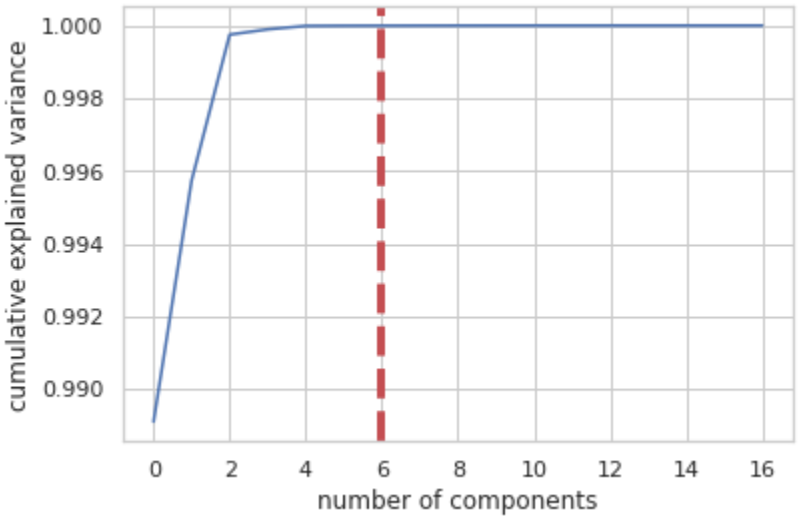
```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(random_state= 59)

pipeline = imbpipeline(steps = [['smote', sm],
                                ['scaler', sc],
                                ['classifier', random_forest_model]])

scores = cross_val_score(pipeline, trainx,ty, cv=5)
print("%0.3f accuracy with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx,ty, cv=5, scoring='f1')
print("%0.3f f1-score with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='roc_auc')
print("%0.3f auc with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='neg_log_loss')
print("%0.3f neg_log_loss with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
```

بر اساس نتایج، auc برای حالت استفاده از میانه بهتر است. می‌دانیم که این معیاری برای مرتب کردن مدل‌ها و روش ها بر اساس عملکرد است و به عدم توازن حساس نیست. پس بهترین روش برای پر کردن جاهای خالی استفاده از میانه است. به همین روش جاهای خالی را پر می کنیم.

برای فهمیدن وجود نفرین ابعاد، باید دو مورد بررسی شود. آیا می توان داده ها را به فضای با بعد کمتر برد؟ اگر ابعاد ویژگی داده‌ها را کم کنیم، عملکرد در طبقه بندی چگونه خواهد بود؟ با افزایش تعداد ویژگی ها، برای یافتن پاسخ صحیح باید تعداد داده ها را به صورت تصاعدی افزایش دهیم. این نفرین ابعاد است. روش‌های کاهش ابعاد به ما کمک می‌کنند تا تعداد ویژگی‌ها را کاهش دهیم در حالی که مقدار داده‌ها ثابت است. این نفرین را کاهش می دهد. ما می توانیم پاسخ درست را راحت تر و سریع تر پیدا کنیم. این امر همچنین امکان یافتن پاسخ مناسب را در برخی موارد که در شرایط اولیه امکان پذیر نبود، می دهد. کاهش ابعاد باعث کاهش تعداد پارامترها و افزایش سرعت آموزش می شود. حجم داده ها نیز کاهش می یابد. پس زمانی در مجموعه داده، نفرین ابعاد داریم که بتوان تقریبا کل

واریانس را با تعداد کمتر ویژگی پوشش داد و عملکرد مدل طبقه بند با این کاهش بعد به صورت چشمگیر کاهش نیابد. برای بررسی این موضوع از روش کاهش بعد PCA استفاده می‌کنیم. ابتدا بررسی می کنیم آیا با تعداد کمتر بعد می توان تقریبا کل واریانس را پوشش داد یا نه. از pca با تعداد کامپوننت به تعداد ویژگی‌ها استفاده می کنیم. میزان واریانس کامپونت‌ها و مجموع واریانس پوششی تا هر کامپوننت را در زیر می‌بینید.





درصد پوشش تا هر کامپوننت برای ۸ تای اول به صورت دقیق در زیر آمده است.

	Cumulative Variance Ratio	Explained Variance Ratio
0	0.989097	9.890974e-01
1	0.995737	6.639960e-03
2	0.999750	4.012801e-03
3	0.999896	1.453737e-04
4	0.999990	9.428595e-05
5	0.999994	4.478175e-06
6	0.999996	1.631505e-06
7	0.999997	1.498690e-06
8	0.999999	1.248538e-06
9	0.999999	7.907180e-07

همانطور که مشاهده می شود، 6 کامپوننت اول تقریبا 100% واریانس را پوشش می دهند. یعنی با تعداد کمتری از ویژگی ها می توان تمام واریانس و اطلاعات دقیق داده ها در بعد اصلی را به دست آورد. این می تواند نشان دهنده وجود نفرین ابعاد و تعداد بالای ویژگی ها باشد. برای فهمیدن وجود نفرین ابعاد و اطمینان پیدا کردن از آن باید به طبقه بندی بپردازیم. یک بار داده در فضای اصلی را مانند قبل با پایپ لاین و مدل randomforest طبقه بندی می‌کنیم. نتایج در زیر آمده است. جاهای خالی با بهترین روش که در قبل یافتیم، پر شده است.

```
0.880 accuracy with a standard deviation of 0.028
0.650 f1-score with a standard deviation of 0.087
0.901 auc with a standard deviation of 0.044
-0.317 neg_log_loss with a standard deviation of 0.033
```

در زیر نتایج مرتبط با کاهش بعد با PCA آمده است. بر اساس میزان پوشش داده ها در PCA، داریم که ۹ کامپوننت اول پوشش بسیار نزدیک به ۱۰۰ درصد را محقق می‌کنند. پس از ۹ کامپوننت استفاده می‌کنیم. کد نیز در ادامه آمده است.

```
0.795 accuracy with a standard deviation of 0.028
0.478 f1-score with a standard deviation of 0.046
0.800 auc with a standard deviation of 0.027
-0.467 neg_log_loss with a standard deviation of 0.041
```

```
from sklearn.ensemble import RandomForestClassifier
# define dataset
trainx = df.drop(['Revenue'], axis=1)
ty = df['Revenue']

svm_smote = SMOTE( random_state=101) #,sampling_strategy='minority', k_neighbors=5)
# X_svm_smote, y_svm_smote = svm_smote.fit_resample(trainx,ty)

# X_train_svm, X_test_svm, y_train, y_test = train_test_split(X_svm_smote,y_svm_smote, test_size=0.20, random_state=

sc = StandardScaler()
# X_train = sc.fit_transform(X_svm_smote)

pca = PCA(n_components=9)
# pca.fit(X_train)
# X_train_scaled_pca = pca.transform(X_train)

# define the model
model = RandomForestClassifier()
pipeline = imbpipeline(steps = [['smote', svm_smote],
                                ['scaler', sc],
                                ['pca', pca],
                                ['classifier', model]])

# fit the model
scores = cross_val_score(pipeline, trainx, ty, cv=5)
print("%0.3f accuracy with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx,ty, cv=5, scoring='f1')
print("%0.3f f1-score with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='roc_auc')
print("%0.3f auc with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='neg_log_loss')
print("%0.3f neg_log_loss with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
```

همانطور که دیده می‌شود، با کاهش بعد ۱۰ درصد مقدار auc کاهش می یابد که کاهش بسیار چشمگیری است. این در حالی است که تقریبا ۱۰۰ درصد اطلاعات را داریم و کمی اطلاعات از دست رفته است. این نشان می‌دهد که اطلاعات از دست رفته قابل توجه بوده است. یعنی کاهش بعد در این مجموعه داده کارا نیست و این نشان دهنده عدم وجود نفرین ابعاد است. برای اطمینان بیشتر روش دیگر کاهش بعد یعنی LDA را را مانند PCA انجام می‌دهیم. نتایج در زیر آمده است.

```
0.701 accuracy with a standard deviation of 0.025
0.402 f1-score with a standard deviation of 0.017
0.753 auc with a standard deviation of 0.017
-2.586 neg_log_loss with a standard deviation of 0.570
```

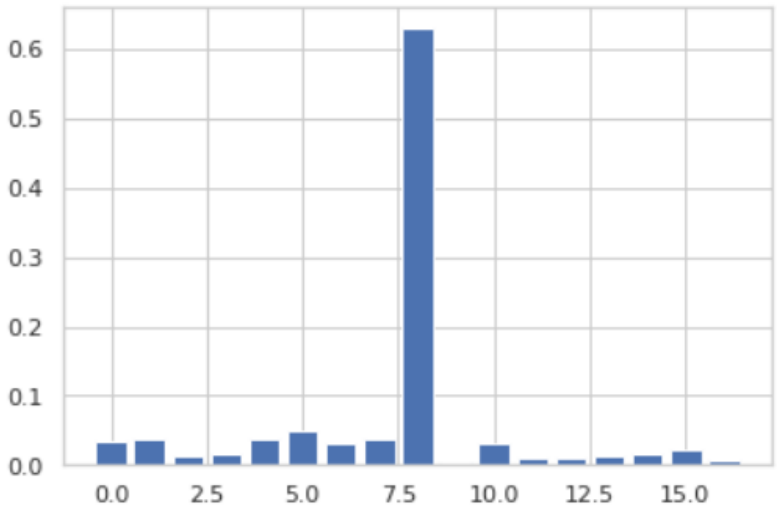
همانطور که دیده می شود، معیار auc به اندازه ۵ درصد از حالت استفاده از pca کمتر است. این در حالی است که در LDA سعی شده که جدا پذیری کلاس‌ها بالا باشد. ما نفرین ابعاد نداریم. این به این دلیل است که عملکرد مدل ثابت با روش‌های کاهش ابعادی بدتر می‌شود و تفاوت عملکرد قابل توجه است. یعنی کاهش تعداد زیاد ابعاد تاثیر زیادی در عملکرد مدل داشته است. در صورتی که داده ها دارای نفرین ابعاد باشند، با کاهش اندازه ابعاد داده ها، عملکرد مدل به طور قابل توجهی کاهش نمی یابد و حتی ممکن است عملکرد بهبود یابد.

در اینجا به بررسی اهمیت ویژگی‌ها در پیش بینی لیبل مورد نظر می‌پردازیم. برای پی بردن به میزان اهمیت ویژگی‌ها از RandomForestRegressor استفاده می‌کنیم. ابتدا روی کل داده SMOTE و سپس اسکیل روی ویژگی‌ها انجام می‌دهیم. سپس RandomForestRegressor را روی کل داده fir می‌کنیم و برای ارزیابی عملکرد مدل باز از کل داده استفاده می‌کنیم. برای ارزیابی معیار R^2 را محاسبه می‌کنیم. این معیار می گوید که چقدر از اطلاعات مربوط به لیبل توسط مدل را توانسته ایم توضیح دهیم. یعنی مدل چقدر اطلاعات لیبل را بر اساس دیگر ویژگی ها پوشش می‌دهد. در زیر کد آمده است.

```
from sklearn.ensemble import RandomForestRegressor
# define dataset
trainx = df.drop(['Revenue'], axis=1)
ty = df['Revenue']
svm_smote = SMOTE( random_state=101) #,sampling_strategy='minority', k_neighbors=5)
X_svm_smote, y_svm_smote = svm_smote.fit_resample(trainx,ty)

# X_train_svm, X_test_svm, y_train, y_test = train_test_split(X_svm_smote,y_svm_smote, test_size=0.20, random_stat

sc = StandardScaler()
X_train = sc.fit_transform(X_svm_smote)
# define the model
model = RandomForestRegressor()
# fit the model
model.fit(X_train, y_svm_smote)
print('R^2:',model.score(X_train, y_svm_smote))
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (trainx.columns[i],v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

خود مدل میزان اهمیت ویژگی ها را محاسبه می‌کند و در اختیار ما می‌گذارد. اهمیت ویژگی‌ها و مقدار معیار ارزیابی در زیر آمده است.

```
R^2: 0.9718687583957014
Feature: Administrative, Score: 0.03373
Feature: Administrative_Duration, Score: 0.03877
Feature: Informational, Score: 0.01336
Feature: Informational_Duration, Score: 0.01682
Feature: ProductRelated, Score: 0.03670
Feature: ProductRelated_Duration, Score: 0.04856
Feature: BounceRates, Score: 0.03203
Feature: ExitRates, Score: 0.03844
Feature: PageValues, Score: 0.63013
Feature: SpecialDay, Score: 0.00416
Feature: Month, Score: 0.03136
Feature: OperatingSystems, Score: 0.00917
Feature: Browser, Score: 0.01019
Feature: Region, Score: 0.01453
Feature: TrafficType, Score: 0.01466
Feature: VisitorType, Score: 0.02186
Feature: Weekend, Score: 0.00552
```



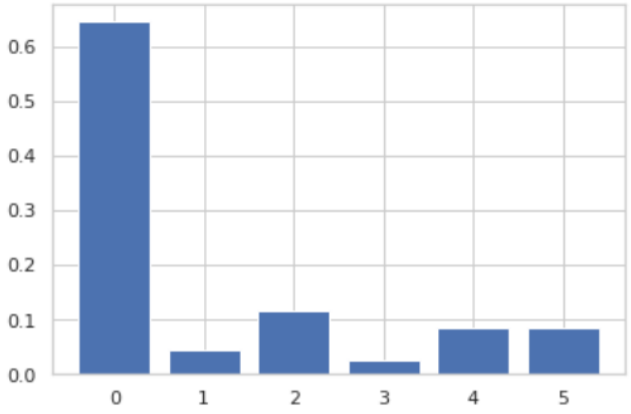
نام ۶ ویژگی برتر در زیر آمده است. مقدار معیار R^2 نزدیک یک است و این یعنی یادگیری به خوبی انجام شده است.

top 6 features from most important to least important

- PageValues
- Month
- ProductRelated_Duration
- VisitorType
- Administrative_Duration
- ProductRelated

حال بررسی می‌کنیم که اگر فقط از این ویژگی‌ها استفاده کنیم، چه میزان از اطلاعات درون لیبل را می توان با دیگر ویژگی‌ها پوشش داد. برای این کار مانند قبل عمل می‌کنیم و فقط یک تفاوت دارد. فقط از این ۶ ویژگی استفاده می‌کنیم. نتایج در زیر آمده است.

```
R^2: 0.9657752125721781
Feature: PageValues, Score: 0.64516
Feature: Month, Score: 0.04376
Feature: ProductRelated_Duration, Score: 0.11619
Feature: VisitorType, Score: 0.02490
Feature: Administrative_Duration, Score: 0.08527
Feature: ProductRelated, Score: 0.08472
```



مقدار معیار ارزیابی کمی کمتر از قبل است. یعنی درصد کمتری از اطلاعات لیبل را با این ویژگی‌های برتر می توان پوشش داد. سایر ویژگی ها درصد کمی از کل اطلاعات را توضیح می دهند. ما از معیار R^2 برای محاسبه میزان پوشش واریانس و اطلاعات موجود در داده ها استفاده می کنیم. با این کاهش ابعاد، اطلاعات قابل توجهی از بین نمی رود. فقدان این ویژگی ها اطلاعات پیش بینی ما را اندکی کاهش می دهد و بر عملکرد مدل تأثیر می گذارد. باید بررسی کنیم که آیا این تاثیر منفی، قابل توجه است یا خیر. برای این کار باید از طبقه بندی با یک مدل رایج استفاده کنیم. باید ببینیم که کاهش ویژگی چقدر معیارهای ارزیابی را کاهش می دهد. مانند مقایسه انجام شده برای کاهش بعد با PCA عمل می‌کنیم. یک بار داده‌ها بدون کاهش بعد را استفاده می کنیم و یک بار داده‌ها را با تنها این ۶ ویژگی استفاده می‌کنیم. نتایج به ترتیب در زیر آمده است.

```
0.880 accuracy with a standard deviation of 0.028
0.650 f1-score with a standard deviation of 0.087
0.901 auc with a standard deviation of 0.044
-0.317 neg_log_loss with a standard deviation of 0.033
```

```
0.830 accuracy with a standard deviation of 0.037
0.616 f1-score with a standard deviation of 0.051
0.884 auc with a standard deviation of 0.040
-0.465 neg_log_loss with a standard deviation of 0.062
```

مقدار auc از 0.901 به 0.884 کاهش یافته است. استفاده از مهمترین ویژگی ها باعث کاهش عملکرد مدل می شود که این کاهش قابل توجه است. یعنی اطلاعات مفیدی در ویژگی‌های دیگر وجود دارد که به ما در طبقه بندی کمک می کند. یعنی نمی توان با این ویژگی های برتر به کار ادامه داد. با توجه به این قسمت ها می توانیم بگوییم که بهترین راه برای کاهش ابعاد، کم نکردن ابعاد است.

در اینجا به بررسی مدل های مختلف و مقایسه آنها می‌پردازیم. برای آموزش مدل و ارزیابی مانند قبل از پایپ لاین استفاده می کنیم و برای ارزیابی مانند قبل از 5-fold برای cross validation استفاده می‌کنیم. کد برای آموزش و

ارزیابی هر مدل در زیر آمده است. جای خالی را با بهترین روش پر می شود. با بهترین روش هم کاهش بعد انجام می‌دهیم. یعنی کاهش بعد انجام نمی دهیم.

```
pipeline = imbpipeline(steps = [['smote', svm_smote],
                                ['scaler', sc],
                                ['classifier', knn_tunned]])
knn_ac = cross_val_score(pipeline, trainx, ty, cv=5)
print("%0.3f accuracy with a standard deviation of %0.3f" % (knn_ac.mean(), knn_ac.std()))
scores = cross_val_score(pipeline, trainx,ty, cv=5, scoring='f1')
print("%0.3f f1-score with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='roc_auc')
print("%0.3f auc with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
scores = cross_val_score(pipeline, trainx, ty, cv=5, scoring='neg_log_loss')
print("%0.3f neg_log_loss with a standard deviation of %0.3f" % (scores.mean(), scores.std()))
```

کافی است مدل را با بهترین هایپر پارامترها به جای knn_tunned قرار دهیم و به کمک این ساختار معیار های ارزیابی را برای عملکرد مدل در طبقه بندی محاسبه کنیم. برای بدست آورن بهترین هایپر پارامترها ما نیاز داریم بین مقادیر ممکن جستجو انجام دهیم. برای این کار ما برای هر مدل مهمترین هایپر پارامتر ها را باید تعیین کنیم. سپس مقادیری برای هر پارامتر در نظر بگیریم و به جستجو بپردازیم. برای جستجو از GridSearchCV استفاده می کنیم. تمام حالات ممکن برای مقادیر تعیین شده برای پارامترها را بررسی می‌کند. برای مقایسه هر چندتایی ممکن مقادیر برای پارامتر ها، از 3-fold برای cross validation استفاده می‌کنیم. برای ارزیابی نیز از معیار roc auc استفاده می کنیم که می دانیم به عدم توازن حساس نیست و معیاری برای مرتب کردن مدل ها از نظر عملکرد است. در زیر استفاده از این روش جستجو برای KNN آمده است.

مانند قبل پایپ لاین مناسب برای طبقه بندی را با این مدل می سازیم و به GridSearchCV می‌دهیم تا جستجو انجام دهد. پارامتر های مورد نظرمان را انتخاب می‌کنیم و مقادیر مورد نظر برای آنها را نیز تعیین می‌کنیم. با سعی و خطا مقادیر مناسب برای جستجو و بازه‌های مقادیر بدست می آید. در اینجا ما فقط دو هایپر پارامتر n_neighbors یا تعداد همسایه ها و p را بررسی می‌کنیم. اینها هایپر پارامتر های مهم در این مدل هستند. در این مدل بر اساس همسایه‌های نزدیک یک داده، کلاس آن تعیین می‌شود. پس دو پارامتر تعداد همسایه نزدیک و معیار نزدیکی مهم

است. P تعیین می کند که معیار فاصله به شکل باشد. مقدار ۲ برای مثال باعث محاسبه فاصله اقلیدسی دو نقطه و مقدار یک باعث محاسبه فاصله منهتن دو نقطه می‌شود.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
from scipy.stats import randint as sp_randint

pca_knn_tunned=KNeighborsClassifier()

pipeline = imbpipeline(steps = [['smote', svm_smote],
                                ['scaler', sc],
                                ['classifier', pca_knn_tunned]])

params={'classifier__n_neighbors':[(i+1)*20 for i in range(4)],
        'classifier__p':[i+1 for i in range(3)]}

rsearch_knn_pca=GridSearchCV(pipeline,params,cv=3,scoring='roc_auc',n_jobs=-1)
# rsearch_knn_pca.get_params().keys()
rsearch_knn_pca.fit(trainx, ty)
```

بعد از fit کردن جستجوگر روی داده‌ها، جستجوگر بهترین چندتایی را پیدا می‌کند. بهترین چندتایی، چندتایی است که باعث بهترین عملکرد در طبقه بندی از نظر معیار مورد نظر شود. در زیر بهترین چندتایی برای این مدل آمده است.

```
rsearch_knn_pca.best_params_
```

```
{'classifier__n_neighbors': 80, 'classifier__p': 1}
```

برای مقایسه مدل‌ها ما نیاز داریم مقادیر معیارها را مانند آنچه گفته شد بدست آوریم. پس باید مدل را با بهترین مقادیر هایپر پارامترهای بدست آمده بسازیم و مقادیر معیارها را محاسبه کنیم. مقادیر معیار ها برای مدل KNN با بهترین هایپر پارامتر ها در زیر آمده است.

```
0.827 accuracy with a standard deviation of 0.023
0.516 f1-score with a standard deviation of 0.041
0.834 auc with a standard deviation of 0.026
-0.403 neg_log_loss with a standard deviation of 0.038
```

برای مدل svm نیز به همین شکل عمل می‌کنیم. مهمترین هایپر پارامتر این مدل نوع کرنل است. اگر نوع کرنل خطی باشد، مدل هایپر پارامتر دیگری ندارد. در حالت چند جمله‌ای ما یک هایپر پارامتر توان چند جمله‌ای داریم. اگر از rbf استفاده کنیم ما یک پارامتر گاما داریم. در تمام این موارد اگر بخواهیم از soft margin استفاده کنیم، یک پارامتر C داریم که مشخص می کند که چقدر در برابر عبور نمونه ها از خط جدا کننده منعطف هستیم. در حالت چند جمله‌ای، پارامتر گاما نیز مهم است. تعریف کرنل‌ها در زیر آمده است.

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

کرنل سیگموئید به دلیل کارایی پایین مورد بررسی قرار نگرفت. بهترین هایپر پارامترها زمانی است که همه مقدار پیش فرض را داشته باشند و کرنل خطی باشد. در زیر عملکرد مدل با بهترین هایپر پارامترها در طبقه بندی آمده است.

```
0.877 accuracy with a standard deviation of 0.032
0.646 f1-score with a standard deviation of 0.084
0.881 auc with a standard deviation of 0.046
nan neg_log_loss with a standard deviation of nan
```

مقدار auc مدل svm از KNN بیشتر است، پس svm بهتر از KNN است. برای بدست آوردن بهترین هایپر پارامتر های Decision Tree به صورت زیر عمل می‌کنیم.

```
params={
        'classifier__max_features':[1,5,10,15,20],
        'classifier__max_depth': [2,6,10,14],
        'classifier__min_samples_split':[1,5,9],
        'classifier__min_samples_leaf':[1,3,5, 6],
        'classifier__criterion':['gini','entropy']}
```

پارامتر های بالا مهمترین پارامترهای مدل هستند که در جستجو در نظر می گیریم. مقادیر منیب نیز در مقابل شان آمده است. Max_depth بیشترین عمق درخت است. Criterion اندازه گیری کیفیت یک تقسیم در درخت را انجام می دهد. دو حالت ممکن آن در نظر گرفته شده است. Min_samples_split حداقل تعداد نمونه مورد نیاز برای تقسیم یک گره داخلی است. Min_samples_leaf حداقل تعداد نمونه مورد نیاز برای قرار گرفتن در یک گره برگ است. یک نقطه تقسیم در هر عمق تنها در صورتی در نظر گرفته می شود که حداقل نمونه های آموزشی min_samples_leaf در هر یک از شاخه های چپ و راست باقی بماند. Max_features تعداد ویژگی هایی است که هنگام جستجوی بهترین تقسیم باید در نظر بگیرید. در زیر بهترین مقادیر این هایپر پارامتر ها آمده است.

```
{'classifier__criterion': 'entropy',
 'classifier__max_depth': 6,
 'classifier__max_features': 1,
 'classifier__min_samples_leaf': 3,
 'classifier__min_samples_split': 5}
```

مقدار معیار های ارزیابی برای این مدل با بهترین هایپر پارامترها در زیر آمده است.

```
0.729 accuracy with a standard deviation of 0.096
0.427 f1-score with a standard deviation of 0.143
0.673 auc with a standard deviation of 0.056
-0.603 neg_log_loss with a standard deviation of 0.134
```

همانطور که دیده می شود، مقدار auc از دو مدل قبل کمتر است. پس همچنان svm بهترین عملکرد را دارد. برای بررسی هایپر پارامتر های مدل linear regression ما فقط دو هایپر پارامتر داریم که در زیر آمده اند.

```
params={
        'classifier__C':[0.01,0.1,1,10,100],
        'classifier__penalty': ['l1', 'l2', 'elasticnet']
        }
```

هایپر پارامتر C، اهمیت رگولاریزیشن را تعیین می‌مند. Penalty نوع رگولاریزیشن را برای ما مشخص می‌کند. برای penalty تمام حالات بررسی شده است. بهترین مقادیر این پارامتر ها بعد از جست و جو در زیر آمده است.

```
{'classifier__C': 0.01, 'classifier__penalty': 'l2'}
```

مقدار معیار های ارزیابی برای این مدل با بهترین هایپر پارامترها در زیر آمده است.

```
0.837 accuracy with a standard deviation of 0.031
0.570 f1-score with a standard deviation of 0.056
0.854 auc with a standard deviation of 0.034
-0.451 neg_log_loss with a standard deviation of 0.060
```

همانطور که دیده می شود، مقدار auc مدل لجستیک رگرسیون از مدل svm کمتر است. یعنی بهترین مدل تا الان svm است.

مدل بعدی mlp است. مهم ترین پارامترهای آن تعداد لایه های مخفی و تعداد نرون‌های در هر لایه آن و تابع فعال ساز نرون‌ها است. در زیر این پارامترها و مقادیر مورد نظر آمده است. Mlp با دو لایه مخفی توانایی دارد که تقریبا هر پترنی را یاد بگیرد. پس کافی است یک و دو لایه مخفی را بررسی کنیم. تعداد نرون‌های لایه‌ها را برابر عدد بزرگی در نظر می گیریم. خود مدل توانایی دارد با وزن‌دهی، نرون‌های بی اهمیت را کمرنگ کند. می تواند تعداد نرون‌های لایه را با این روش کاهش دهد.

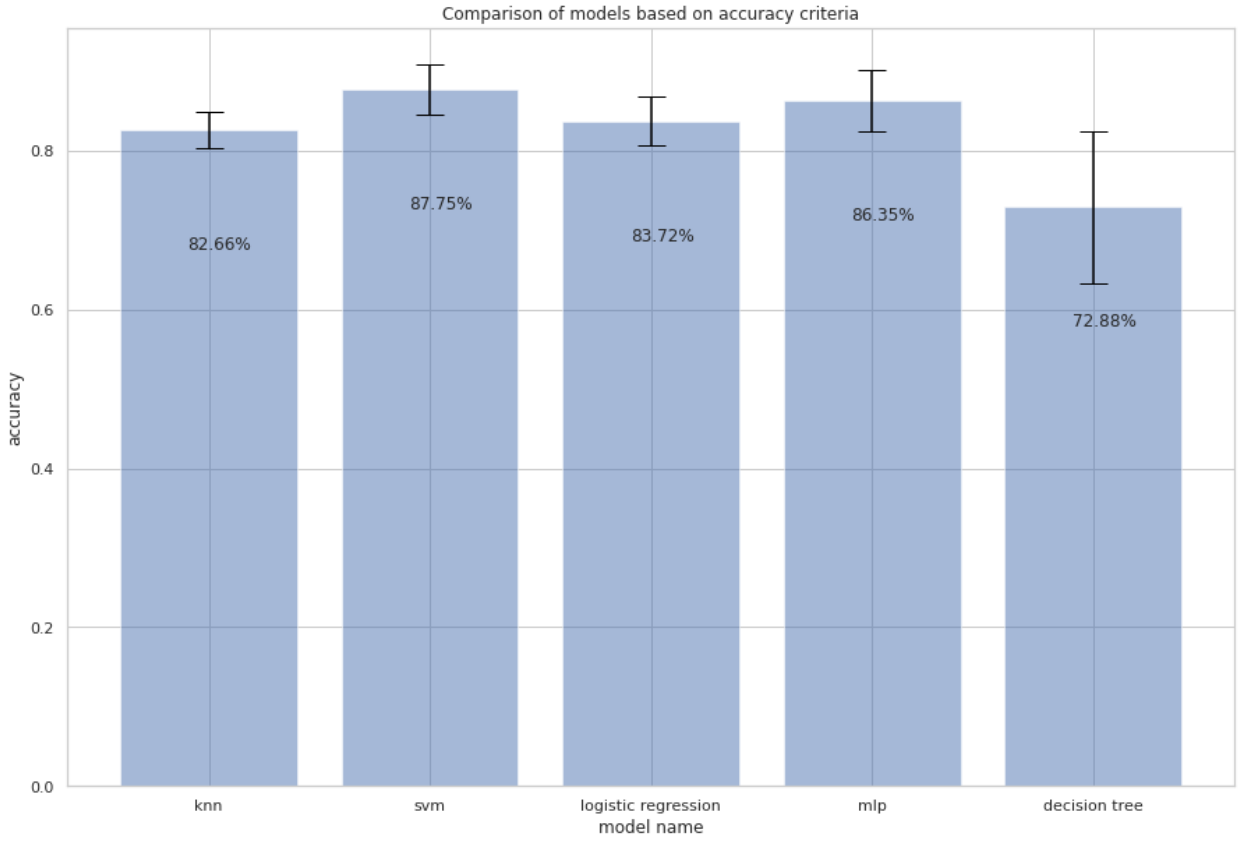
```
params={
        'classifier__activation':['logistic', 'tanh', 'relu'],
        'classifier__hidden_layer_sizes': [(100),(100,100)]
        }
```

در زیر بهترین مقادیر این پارامترها و مقادیر معیار های ارزیابی مدل mlp با بهترین هایپر پارامترها آمده است.

```
{'classifier__activation': 'logistic',
 'classifier__hidden_layer_sizes': (100, 100)}
```

```
0.864 accuracy with a standard deviation of 0.038
0.636 f1-score with a standard deviation of 0.086
0.880 auc with a standard deviation of 0.054
-0.373 neg_log_loss with a standard deviation of 0.086
```

همانطور که دیده می شود مقدار تمام سه متریک اول مدل mlp از مدل svm کمتر است. این یعنی بهترین مدل ما مدل svm است. نمودار Accuaracy بدست آمده از طبقه‌بند های بالا در زیر آمده است.



عملکرد هر مدل از نظر این معیار به شکل یک bar در bar plot رسم شده است. ارتفاع هر ستون نشان دهنده میانگین accuracy برای cross validation در حالت fold-5 است. روی هر ستون این مقدار نوشته شده است. مقدار انحراف از معیار برای نتیجه هر مدل از نظر این معیار نیز به صورت error bar روی هر ستون آمده است. هر چه این error bar طول بیشتری داشته باشد، یعنی مقادیر accuracy برای cross validation در حالت fold-5 از میانگین فاصله‌های بیشتری داشته اند. پس ستونی که بیشترین ارتفاع را دارد، بهترین مدل را نشان می دهد. اگر دو

ستون هم ارتفاع باشند، کوچکتر بودن error bar مهم است. همانطور که دیده می‌شود، ستون مدل svm از همه بلندتر است. یعنی بهترین مدل از این منظر svm است.

دو مدل دیگر را بررسی می‌کنیم. اول مدل xgboost را بررسی می‌کنیم. XGBoost یک پیاده‌سازی کارآمد از الگوریتم درختان تقویت‌شده گرادیان (gradient boosted trees) است. تقویت گرادیان (Gradient boosting) یک الگوریتم یادگیری نظارت شده است که تلاش می‌کند تا با ترکیب تخمین‌های مجموعه‌ای از مدل‌های ساده‌تر و ضعیف‌تر، یک متغیر هدف را پیش‌بینی کند. یک روش ensemble تعدادی مدل ضغیف برای ساخت مدل قوی تر است. برای جل یک تابع هزینه دارد با گرادیان کاهشی به بهینه سازی تابع هزینه می‌پردازد. ما از این مدل با هایپر پارامتر های پیش فرض استفاده می‌کنیم. مقدار معیارهای ارزیابی برای این مدل در زیر آمده است.

```
0.873 accuracy with a standard deviation of 0.034
0.650 f1-score with a standard deviation of 0.091
0.897 auc with a standard deviation of 0.040
-0.308 neg_log_loss with a standard deviation of 0.054
```

همانطور که دیده می شود، مقدار auc آن از مدل svm که بهترین مدل ما بود، بیشتر است. یعنی این مدل از ۵ مدل عنوان شده عملکرد بهتری داشته است. در ادامه از مدل random forest استفاده می‌کنیم. این مدل از تعدادی decision tree استفاده می کند. این مدل پیش بینی این درختان را با هم ترکیب می کند تا پیش بینی نهایی را انجام دهد. در این مدل باز ensemble تعدادی مدل ضغیف برای ساخت مدل قوی تر را می بینیم. تعداد پیش بینی کننده، هایپر پارامتر اضافه آن نسبت به درخت تصمیم است. ما از این مدل با مقدار پیش فرض هایپر پارامترها استفاده می کنیم. مقادیر معیارهای ارزیابی در زیر آمده است.

```
0.883 accuracy with a standard deviation of 0.028
0.657 f1-score with a standard deviation of 0.084
0.902 auc with a standard deviation of 0.043
-0.316 neg_log_loss with a standard deviation of 0.032
```

همانطور که دیده می‌شود، مقدار auc از مقدار auc تمام مدل‌ها بیشتر است. یعنی این مدل از تمام مدل ها بهتر است.

دیدیم که مدل‌های با روش ensemble که تعدادی مدل را ترکیب می کنند، عملکرد بهتری داشتند. نکته این است که ترکیب کردن یک مدل با مدل دیگر می تواند باعث شود که نقاط ضعف مدل ها کمتر شود. با اجتماع تعدادی مدل می توان نقاط ضعف را کمتر کرد و به مدلی بهتر در پیش بینی از مدل های پایه رسید. نکته مهم دیگر این است که ترکیب به این شکل تاثیر عدم توازن در پیش بینی را هم کمتر می کند و باعث می‌شود که پیش بینی های مطمئن تری داشته باشیم.