دانشگاه علم و صنعت ایران
دانشکده مهندسی کامپیوتر

عنوان: پروژه چهارم درس داده‌کاوی
رتبه‌بندی ویژگی‌های مؤثر در دسته‌بندی مجموعه داده‌ها

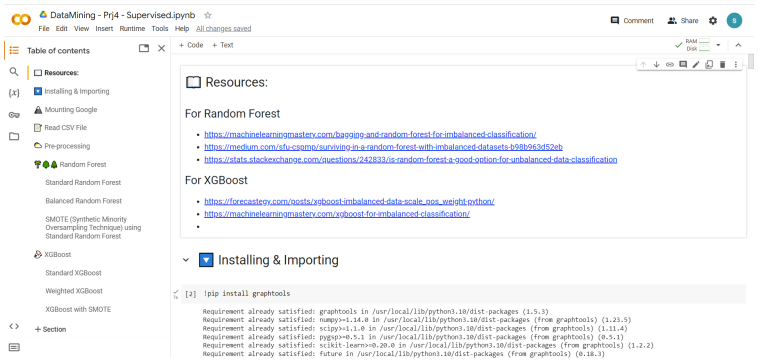
نام و نام خانوادگی: آیلین نائب‌زاده
شماره دانشجویی: ۹۹۵۲۲۱۸۵
نیم‌سال تحصیلی: پائیز ۱۴۰۲
مدرّس: دکتر حسین رحمانی

فهرست مطالب

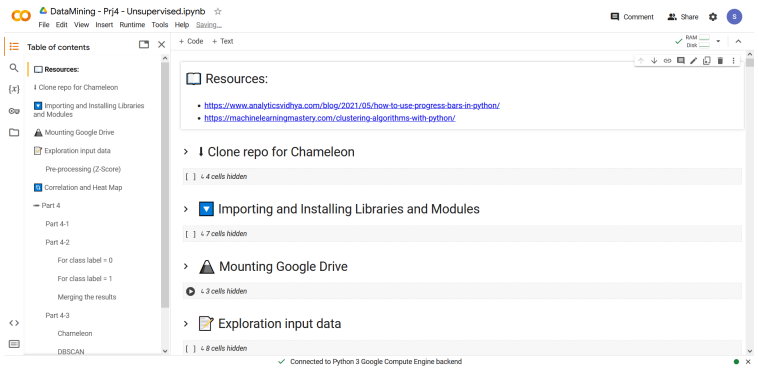
۱ گام اول

در این پروژه هدف این است تا باتوجه به مجموعه داده برچسب گذاری شده، ویژگی موجود را براساس میزان تاثیرگذاری رتبه‌بندی کنیم. فایل ورودی دارای ۸۳ ستون و حدود ۳۰۰۰۰۰ ردیف می‌باشد. همچنین یک ستون بیانگر شماره ردیف با شروع از ۱ و آخرین ستون نیز بیانگر برچسب نمونه‌ها می‌باشد که می‌تواند مقدار ۰ یا ۱ را داشته باشد.

بدلیل کمبود زمان و پیش بردن سریع‌تر مراحل خواسته‌شده به‌صورت موازی، گام‌های مورد نیاز برای حل پروژه، در دو فایل جداگانه به نام‌های DataMining - Prj4 - Supervised.ipyn و DataMining - Prj4 - Unsupervised.ipynb پیاده‌سازی شده‌اند.



شکل ۱: نمای کلی از پروژه با ناظر



شکل ۲: نمای کلی از پروژه بدون ناظر

در مرحله اول بااستفاده از کتابخانه pandas و تابع read_csv، فایل‌های ورودی را به متغیرهایی از نوع dataframe تبدیل می‌کنیم تا در قدم‌های بعدی راحت‌تر بتوانیم تحلیل‌های مورد نیاز را انجام بدهیم. همچنین بااستفاده از توابع info()، head() و describe() اطلاعات بیشتری نسبت به داده‌های موجود کسب می‌کنیم.

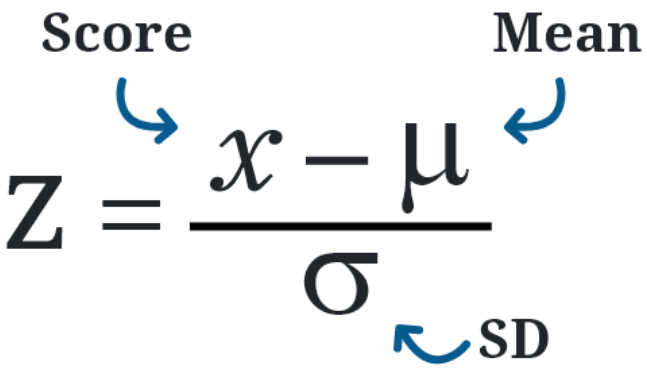
* خروجی تمامی مراحل ذکر شده بطور کامل در فایل‌های ipynb موجود می‌باشند.

۲ گام دوم

طبق توضیحات ذکر شده در فایل پروژه به‌منظور پیش پردازش داده‌ها تنها کافی است عملیات نرمالایز را بر روی داده‌ها انجام دهیم. البته همان‌گونه که در خروجی تابع info() می‌توانید مشاهده کنید، هیچ یک از ستون‌ها شامل مقادیر None نمی‌باشد و همگی دارای ۲۹۸۷۱۴ مقدار غیر تهی می‌باشند.

حال به‌منظور انجام عملیات normalization از تابع آماده StandardScaler() موجود در کتابخانه sklearn.preprocessing استفاده می‌کنیم. درواقع این تابع عملیات Z-Score را بر روی ویژگی‌های گوناگون به‌طور جداگانه انجام می‌دهد.

البته به این نکته دقت شود که نیازی نیست که برروی ستون‌های اول و آخر مجموعه داده‌های ورودی نیز این عملیات انجام شود.

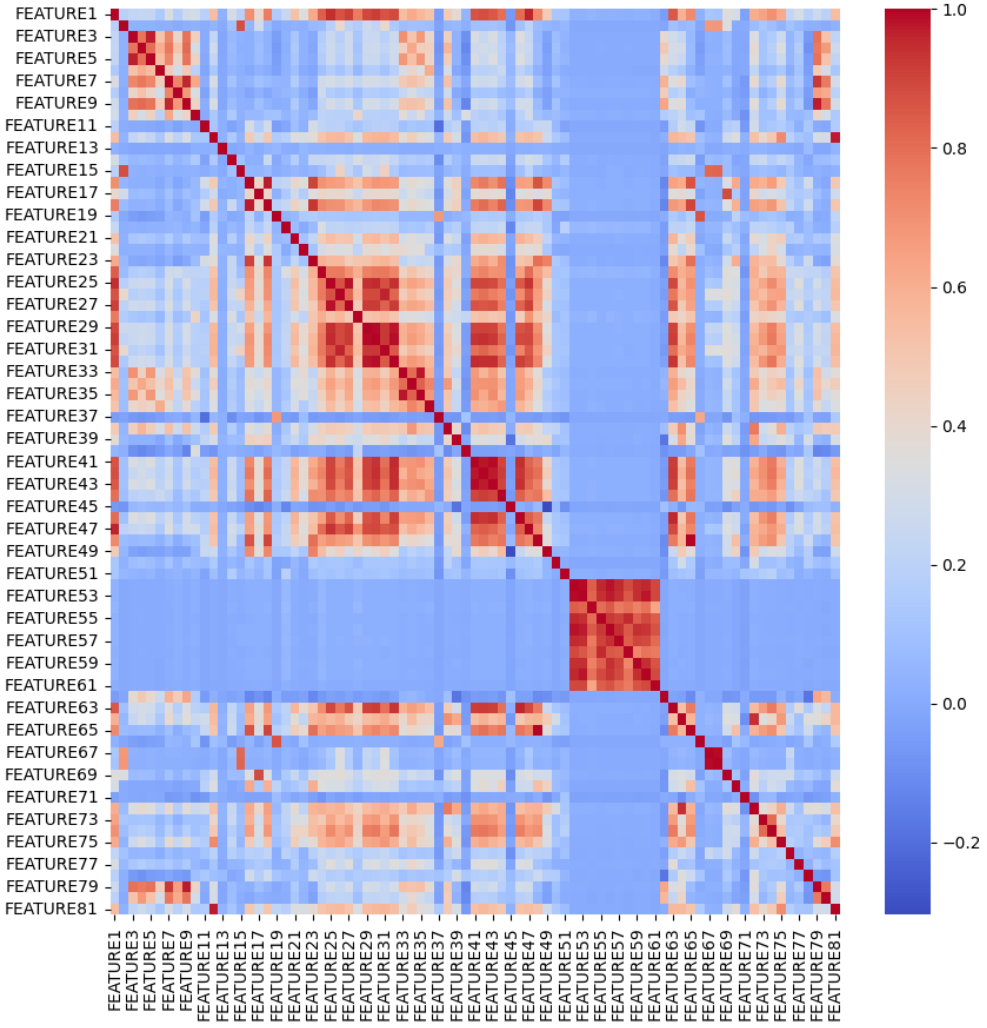


شکل ۳: رابطه Z-Score

۳ گام سوم

در این مرحله همان‌طور که خواسته شده‌است، ماتریس همبستگی میان ویژگی‌های موجود را بدست می‌آوریم. برای محاسبه این ماتریس می‌توانیم از تابع آماده corr(). که در کتابخانه pandas موجود می‌باشد استفاده کنیم.

تصویر ماتریس نهایی را در شکل زیر می‌توانید مشاهده کنید.



شکل ۴: ماتریس همبستگی میان ویژگی‌ها

۴ گام چهارم

این مرحله درواقع مربوط به استفاده از الگوریتم یادگیری بدون ناظر می‌باشد. در دو بخش اول از الگوریتم‌های K-Means و Mini Batch K-Means استفاده می‌کنیم. و در بخش سوم از الگوریتم BIRCH به‌همراه الگوریتم PCA به‌منظور کاهش ابعاد داده‌ها استفاده خواهیم کرد.

در این مرحله به‌دلیل حجم بالای داده‌ها با چالش‌های زیادی مواجه شدم. بطور مثال در زیر بخش سوم با ارور .Session crashed. All the RAM was used در پروژه Colab چندین بار مواجه شدم که مجبور به کاهش ابعاد مدل شدم. و همچنین بدلیل تعداد بالای نمونه‌ها، ارزیابی مدل‌های این مرحله بااستفاده از الگوریتم Silhoutte به زمان زیادی احتیاج داشتند.

۱.۴ بخش الف)

در این مرحله ابتدا بااستفاده از تابع ()find_optimal_k_batch مقدار بهینه K را برای الگوریتم K-Means که برروی تمام داده‌ها اجرا خواهد شد، پیدا خواهیم کرد. نحوه ارزیابی مقدار بهینه K نیز براساس بیشینه مقدار Silhoutte می‌باشد که همانطور که می‌دانیم هرچه به عدد ۱ نزدیک‌تر باشد یعنی خوشه‌های بهتری تشکیل شده‌اند. طبق نتایج بدست آمده، مقدار K = 3 بهترین خروجی را برای ما خواهد داشت. (برای این بخش مقادیر K از ۲ تا ۱۰ مورد بررسی قرار گرفته‌اند.) در نهایت الگوریتم K-Means با k = 3 برروی تمامی داده‌ها را مجددا اجرا می‌کنیم. در نهایت در یک حلقه با تعداد دفعات تکرار ۸۱(به تعداد ویژگی‌ها) و محاسبه نسبت بیشینه مسافت درون خوشه و خارج از آن، میزان اهمیت و تاثیرگذاری ویژگی‌ها را محاسبه می‌کنیم. بخشی از کد مربوط به این بخش را در زیر به‌همراه تعدادی از ویژگی‌ها به‌همراه اهمیتشان می‌توانید مشاهده کنید.

* در فایل اصلی تمامی ویژگی‌ها همراه با میزان اهمیت‌شان آورده‌شده‌اند.

```
# calculate the feature importance for each cluster
importance = []
for i in range(81):
  imp = 1 - np.std(centroids[:, i]) / (np.max(centroids[:, i]) - np.min(
     centroids[:, i]))
  importance.append(imp)

# sort the features by importance in descending order
features = X_scaled_df.columns
sorted_features = sorted(zip(features, importance), key=lambda x: x[1],
    reverse=True)

# print the feature importance
print("Feature importance for k-means clustering for all the samples:")
for f, i in sorted_features:
  print(f"{f}: {i:.4f}")
```

```
Feature importance for k-means clustering for all the samples:
FEATURE64: 0.5918
FEATURE45: 0.5917
FEATURE38: 0.5917
FEATURE72: 0.5917
FEATURE11: 0.5917
FEATURE7: 0.5917
FEATURE76: 0.5915
FEATURE9: 0.5914
FEATURE4: 0.5909
FEATURE79: 0.5904
FEATURE50: 0.5903
FEATURE8: 0.5901
FEATURE71: 0.5898
FEATURE62: 0.5895
FEATURE78: 0.5873
FEATURE10: 0.5860
FEATURE6: 0.5858
FEATURE39: 0.5855
FEATURE5: 0.5847
FEATURE37: 0.5843
FEATURE3: 0.5837
FEATURE51: 0.5819
FEATURE61: 0.5819
FEATURE69: 0.5768
FEATURE80: 0.5763
FEATURE17: 0.5751
FEATURE75: 0.5739
FEATURE81: 0.5730
FEATURE12: 0.5725
FEATURE49: 0.5711
```

شکل ۵: خروجی بخش ۴-الف)

۲.۴ بخش ب)

در این بخش نیز مراحل بخش الف) را تکرار می‌کنیم ولی با این تفاوت که نیاز است الگوریتم K-Means را یک بار برروی نمونه‌هایی با برچسب ۱ و بار دیگر برروی نمونه‌هایی با برچسب ۰ جداگانه اجرا کنیم. در هر بار مقدار بهینه K را بااستفاده از معیار Silhoutte جداگانه محاسبه کردم. همچنین در نهایت بااستفاده از یک میانگین وزن‌دار نتایج اهمیت ویژگی‌ها را با یکدیگر ادغام کردم.

بخشی از کد مربوط به این بخش را در زیر به‌همراه تعدادی از ویژگی‌ها به‌همراه اهمیتشان می‌توانید مشاهده کنید.

* در فایل اصلی تمامی ویژگی‌ها همراه با میزان اهمیت‌شان آورده‌شده‌اند.

```
Feature importance for k-means clustering based on the weighted average:
FEATURE18: 0.5916
FEATURE67: 0.5916
FEATURE36: 0.5916
FEATURE27: 0.5916
FEATURE31: 0.5916
FEATURE29: 0.5916
FEATURE26: 0.5915
FEATURE16: 0.5915
FEATURE30: 0.5915
FEATURE41: 0.5915
FEATURE43: 0.5914
FEATURE32: 0.5914
FEATURE25: 0.5914
FEATURE44: 0.5914
FEATURE39: 0.5913
FEATURE40: 0.5912
FEATURE69: 0.5912
FEATURE42: 0.5911
FEATURE33: 0.5911
FEATURE20: 0.5910
FEATURE35: 0.5905
FEATURE7: 0.5900
FEATURE21: 0.5899
FEATURE34: 0.5899
FEATURE46: 0.5898
FEATURE47: 0.5897
FEATURE68: 0.5896
FEATURE73: 0.5894
FEATURE72: 0.5889
FEATURE63: 0.5888
FEATURE76: 0.5875
FEATURE38: 0.5869
FEATURE22: 0.5869
```

شکل ۶: خروجی بخش ۴-ب)

```
1  importance_0 = []
2  importance_1 = []
3  features = X.columns
4  for i in range(len(features)):
5    imp_0 = 1 - np.std(centroids_0[:, i]) / (np.max(centroids_0[:, i]) - np.min(
       centroids_0[:, i]))
6    importance_0.append(imp_0)
7    imp_1 = 1 - np.std(centroids_1[:, i]) / (np.max(centroids_1[:, i]) - np.min(
       centroids_1[:, i]))
8    importance_1.append(imp_1)
9
10 sorted_features_0 = sorted(zip(features, importance_0), key=lambda x: x[1],
     reverse=True)
```

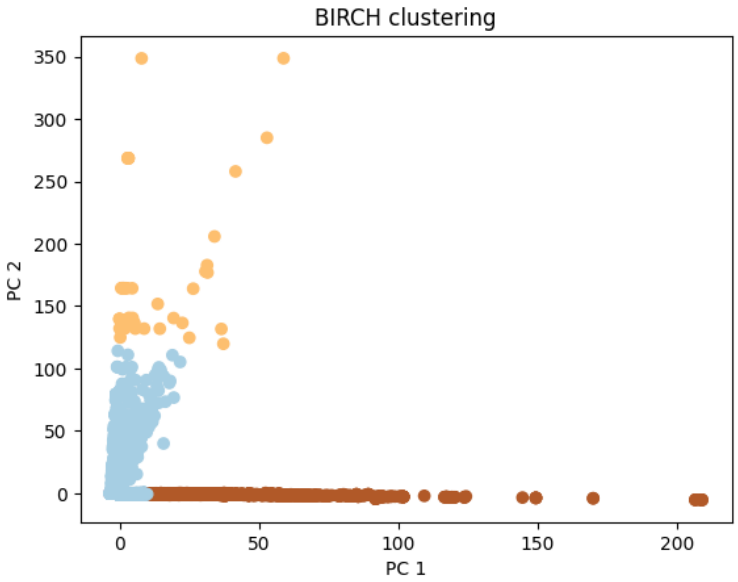
```
sorted_features_1 = sorted(zip(features, importance_1), key=lambda x: x[1],
    reverse=True)
def weighted_average(x, y, w):
  return [w * a + (1 - w) * b for a, b in zip(x, y)]
w_0 = len(X_0) / len(X)
w_1 = len(X_1) / len(X)
importance_avg = weighted_average(importance_0, importance_1, w_0)
sorted_features_avg = sorted(zip(features, importance_avg), key=lambda x: x
    [1], reverse=True)
print("Feature importance for k-means clustering based on the weighted average
    :")
for f, i in sorted_features_avg:
  print(f"{f}: {i:.4f}")
```

۳.۴ بخش ج)

در این مرحله نیز از ما خواسته شده‌است تا با یک الگوریتم خوشه‌بندی دیگر میزان تاثیرگذاری ویژگی‌ها را محاسبه کنیم. ابتدا الگوریتم Chameleon را تست کردم. برای این کار طبق توضیحات فرستاده‌شده، سعی کردم از کتابخانه تعریف‌شده یکی از دوستان استفاده کنم. در هنگام استفاده از این الگوریتم به چالش‌هایی همانند نصب کتابخانه‌های مورد نیاز و هماهنگی آن‌ها با سیستم‌عامل مواجه شدم. همچنین پس از رفع مشکلات گفته‌شده به زمانی در حدود ۲۰۰ ساعت نیاز بود تااینکه داده‌های ورودی را خوشه‌بندی کند.

به‌عنوان جایگزین به‌سراغ استفاده از الگوریتم DBSCAN می‌روم. در هنگام استفاده از این الگوریتم حتی با وجود کاهش ابعاد داده‌های ورودی، ولی خطای .Session is crashed را در محیط پروژه Colab دریافت کردم.

پس از چندین تلاش، در نهایت به‌سراغ استفاده از الگوریتم BIRCH رفتم. برای استفاده از این الگوریتم نیز در صورتی که همه داده‌های ورودی را به‌عنوان ورودی استفاده کنیم، خطایی مشابه حالت DBSCAN مشاهده خواهیم کرد. در نتیجه در ابتدا بااستفاده از الگوریتم PCA، ابعاد داده‌ها را کاهش می‌دهیم. خروجی نهایی مربوط به این الگوریتم و تائیر دو ویژگی اصلی را در تصاویر زیر می‌توانید مشاهده کنید.



شکل ۷: خوشه‌های حاصل از بخش ۴-ج)

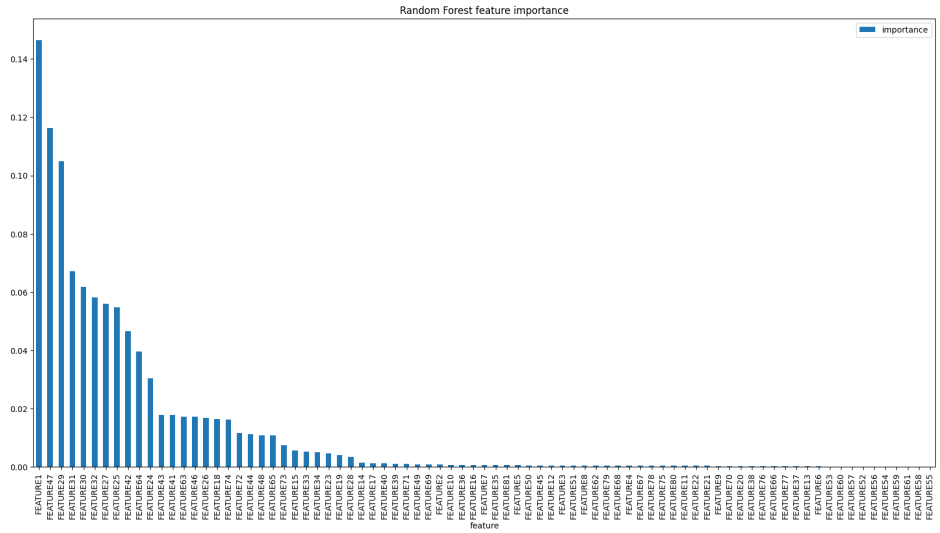
```
Feature importance for BIRCH clustering:
PC1: 0.1663
PC2: 0.0002
```

شکل ۸: خروجی بخش ۴-ج)

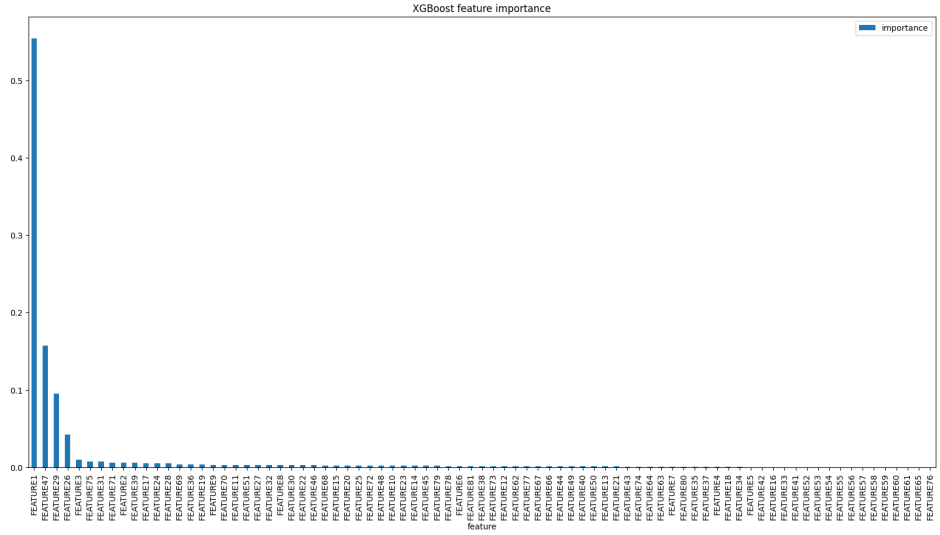
۵ گام پنجم

در این مرحله باتوجه به خواسته موجود در صورت پروژه الگوریتم Random Forest و XGBoost جهت یادگیری با نظارت استفاده می‌کنیم. همچنین باتوجه به توزیع نامتقارن داده‌ها با برچسب‌های ۰ و ۱ در هنگام استفاده از هر یک از الگوریتم‌های ذکر شده، از دو روش برای بهبود نتایج استفاده می‌کنیم. باتوجه به اینکه بهترین نتایج در حالت Random Forest with Oversampling و Standard XGBoost بدست می‌آیند، تنها برای این دو حالت تاثیرگذاری و میزان اهمیت هر یک از ویژگی‌ها را محاسبه می‌کنیم.

نتایج گفته‌شده را در تصاویر زیر می‌توانید مشاهده کنید.



شکل ۹: میزان اهمیت و تاثیرگذاری ویژگی‌ها برروی الگوریتم Random Forest with SMOTE

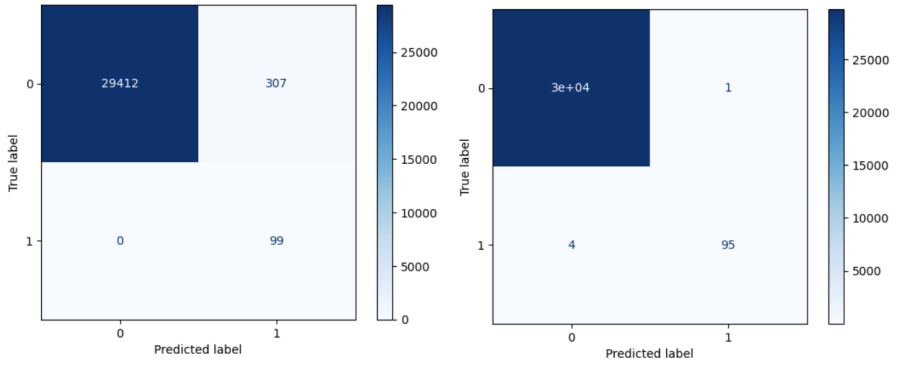


شکل ۱۰: میزان اهمیت و تاثیرگذاری ویژگی‌ها برروی الگوریتم Standard XGBoost

۶ گام ششم

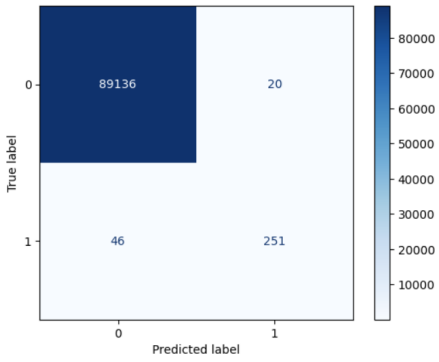
نتایج حاصل از ارزیابی معیارهای گوناگون برروی مدل‌های استفاده‌شده را در فایل اکسل ارسال‌شده می‌توانید مشاهده کنید.

همچنین در تصاویر زیر نیز می‌توانید Confusion Matrix الگوریتم‌های یادگیری با نظارت استفاده‌شده در حالت‌های مختلف را مشاهده کنید.

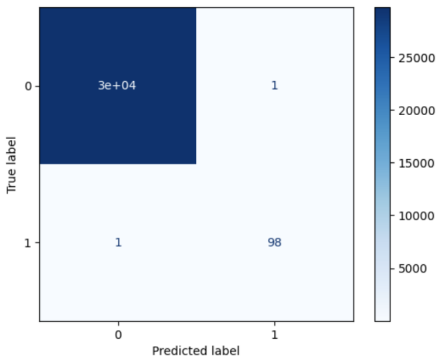


Balanced Random Forest (ب)

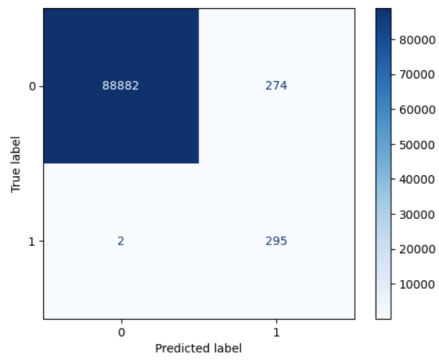
Standard Random Forest (آ)



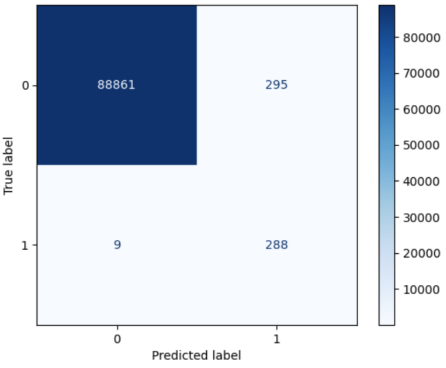
Standard XGBoost (د)



SMOTE Random Forest (ج)



SMOTE XGBoost (و)



Weighted XGBoost (ه)

۷ نتیجه‌گیری

۱.۷ آیا ویژگی‌هایی وجود دارند که در هر دو نوع یادگیری **Supervised** و **Unsupervised** از نظر تاثیرگذاری هم‌پوشانی داشته باشند؟

در حالت یادگیری بدون ناظر هم‌پوشانی مشاهده نمی‌شود. ولی در یادگیری باناظر همان‌گونه که در تصاویر گام پنجم توضیح داده شده‌است، اولین ویژگی بیشترین تاثیر در حالت‌های محاسبه‌شده را داشته‌است و بین سایر ویژگی‌ها نیز هم‌پوشانی و شباهت دیده می‌شود.

۲.۷ به نظر شما کدام یک از رتبه‌بندی‌ها منطقی‌تر است؟

باتوجه به نتایج خروجی و همچنین چالش‌های حل مسئله، الگوریتم با ناظر عملکرد بهتری داشته‌اند. باتوجه به حجم بالای داده‌ها در هنگام استفاده از الگوریتم‌های بدون ناظر ممکن است با چالش‌های سخت‌افزاری مواجه شویم. ولی با تغییرات جزئی در الگوریتم‌های با ناظر می‌توانیم به دقت و صحت بالایی دست پیدا کنیم.

همچنین یکی از چالش‌های دیگر در هنگام استفاده از الگوریتم بدون ناظر تعیین مقادیر برای پارامترهای مورد نیاز آن‌ها می‌باشد که نتایج نهایی را تحت تاثیر قرار می‌دهند.

۳.۷ اگر اختلاف زیادی در معیارهای ارزیابی روش‌ها مشاهده شد، دلیل آن را بنویسید.

در الگوریتم‌های با ناظر مقادیر accuracy، precision، recall و F1-score شباهت زیادی با یکدیگر دارند. در حالت استفاده از الگوریتم‌های بی‌ناظر همانطور که در بخش چهار اشاره شده‌است، مقادیر Silhoutte شباهت زیادی به یک‌دیگر دارند.