

به نام خدا

**موضوع :**

گزارش پایانی پروژه داده کاوی

(**(Type 2 Diabetes Mellitus Prediction Model Based on Data Mining**

**تهیه کننده :**

فریبا محمدی خواه

**استاد درس :**

دکتر بهشید بهکمال

* **شرح روش پیاده سازی :**

در این مقاله یک مدل جدید مبتنی بر تکنیک های داده کاوی برای پیش بینی دیابت نوع2 پیشنهاد شده است که هدف آن بهبود دقت نسبت به کارهای پیشین و سازگاری مدل با بیش از یک مجموعه داده است. این مدل به 3.04٪ دقت پیش بینی بیشتری نسبت به سایر محققان رسیده است.

در این مقاله از ابزار WEKA برای انجام مراحل استفاده شده اما روش پیاده سازی بنده با استفاده از زبان پایتون و کتابخانه های کامل آن انجام شده است.

1. Data Preprocessing :

کیفیت داده ها تا حد زیادی بر نتیجه پیش بینی تاثیر می گذارد بدین منظور ابتدا ، پیامدهای پزشکی هر یک از ویژگی ها و ارتباط آن با DM را تجزیه و تحلیل شده در نتیجه تشخیص داده شده که تعداد حاملگی ها ارتباط کمی با DM دارند. بنابراین ، این ویژگی عددی را به یک صفت اسمی تبدیل کردیم. مقدار 0 نشانگر باردار نبودن و 1 باردار بودن است. با این فرآیند از پیچیدگی مجموعه داده کاسته شد.

for k in df1['Pregnancies']:

if k != 0:

df1['Pregnancies'] = df1['Pregnancies'].replace(k,1)

برخی از مقادیر گمشده و نادرست در مجموعه داده وجود داشت. به عنوان مثال ، در مجموعه داده اصلی ، مقادیر 'Glucose','BloodPressure','SkinThickness','Insulin','BMI'نمی تواند 0 باشد ، که نشان می دهد مقدار واقعی از دست رفته است. برای کاهش تأثیر مقادیر بی معنی ، از میانگین برای جایگزینی تمام مقادیر از دست رفته استفاده شده است.

for l in ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']:

df1[l] = df1[l].replace(0,df1[l].mean())

1. Normalization:

با استفاده از فرمول Z-score مقادیر در بازه ی [0,1] نرمال شده اند.

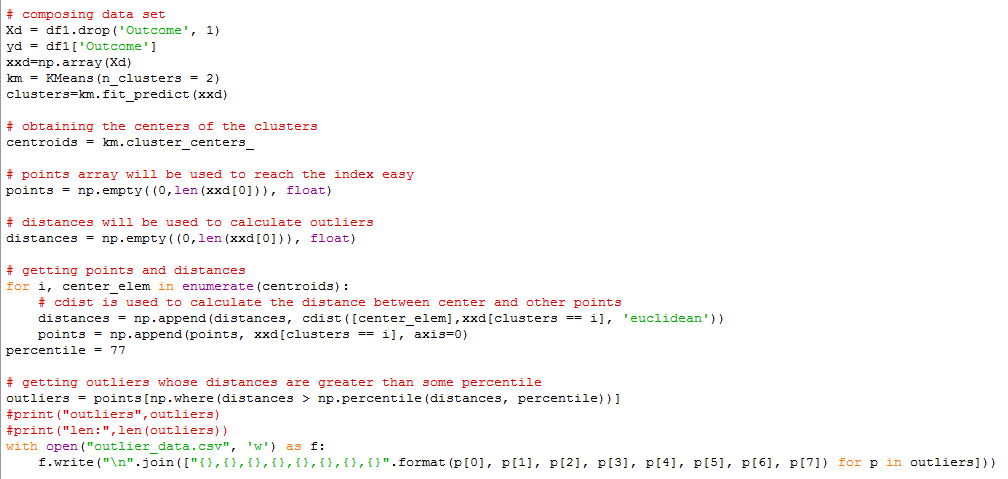
for col in df1.columns[1:-1] :

df1[col] = (df1[col] - df1[col].mean())/df1[col].std(ddof=0)

1. K-Means Algorithm :

از الگوریتم k-means برای حذف داده های outlier استفاده شده است. ابتدا داده ها و لیبل کلاس ها جدا شده اند. سپس با استفاده از کتابخانه from sklearn.cluster import KMeans الگوریتم k-means با دو خوشه ( زیرا دو کلاس 0 و 1 داریم؛ دیابتی بودن یا نبودن) فراخوانی شده است. سپس داده ها خوشه بندی شده و centroid ها تعیین شده است. در نهایت فاصله نقاط با استفاده از یک Threshold که مقدار آن را 77 در نظرگرفته ام سنجیده شده اگر بزرگتر از این مقدار بوده نقطه به عنوان یک نقطه outlier در نظر گرفته شده است و در فایل outlier\_data.csv ذخیره شده است.

تعداد داده های پرت شناسایی شده 177 بود و تعداد کل داده های دیتاست 768 در نتیجه داده های صحیح پیاده سازی اینجانب 591 شد در حالیکه تعداد داده های صحیح مقاله 589 است.



پس از شناسایی داده های پرت این داده ها از دیتاست اصلی با قطعه کد زیر حذف شده اند و خروجی این فاز به عنوان ورودی به Regression Logistic داده می شود.

for indexx in outliers:

num.append((df1[(df1.Pregnancies == indexx[0]) & (df1.Glucose == indexx[1]) & (df1.BloodPressure == indexx[2]) & (df1.SkinThickness == indexx[3]) & (df1.Insulin == indexx[4]) & (df1.BMI == indexx[5]) & (df1.DiabetesPedigreeFunction == indexx[6]) & (df1.Age == indexx[7])].index)[0])

df1 = df1.drop(num, 0)

1. Logistic Regression Algorithm:

در این مرحله داده ها را طبقه بندی می کنیم برای اینکار ابتدا داده ها را به دو بخش داده ی Train و Test تقسیم می کنیم. 20درصد داده Test و 80درصد داده Train در نظر گرفته ام یعنی 119 داده ی Test و 472 داده ی train. Features ستون های دیتاست و lbl لیبل کلاسهاست.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, lbl, test\_size=0.2,random\_state=29)

سپس با استفاده از کتابخانه ی from sklearn.linear\_model import LogisticRegression داده های X\_train, y\_train به عنوان ورودی به تابع LogisticRegression().fit می دهیم و پس از یادگیری مدل با دادن داده ی تست به logreg.predict مقدار کلاس را برای داده های تست پیش بینی می کنیم. دقت حاصل از این طبقه بندی 89 درصد شد.

logreg = LogisticRegression().fit(X\_train, y\_train)

y\_pred = logreg.predict(X\_test)

print('model accuracy: {:.2f}'.format(logreg.score(X\_test, y\_test)))

* **مقایسه نتایج با مقاله :**

ابتدا ماتریس در هم ریختگی برای داده های Test رسم شده

confusion\_matrix = confusion\_matrix(y\_test, y\_pred)

در این قسمت مقاله کل داده ها را پوشش داده بود ینی تمام داده ها بعد از حذف داده های پرت در ماتریس مجاورت قرار داده شده بود که با مشورت با شما تصمیم بر این شد داده ها به دو قسمت داده ی test و train تبدیل شوند در نتیجه ماتریس مجاورت بنده با ماتریس مجاورت مقاله متفاوت است.

10-fold cross :

در این روش داده ها به 10 بخش تقسیم شده اند و هر بار یادگیری بر روی یک بخش انجام شده یعنی مدل 10 بار آموزش و تست شده است و میانگین این 10 بار به عنوان دقت در نظر گرفته شده است.

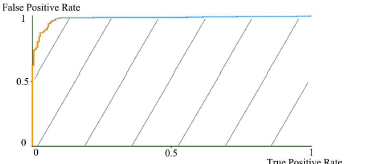
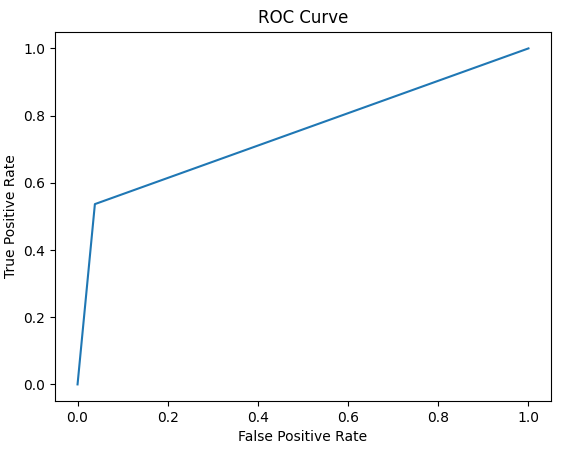
جدول زیر دقت های مقاله می باشد:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction Accuracy | Kappa | ROC | MCC | Recall | Precision |
| 95.42% | 0.897 | 0.979 | 0.899 | 0.954 | 0.954 |

جدول زیر دقت های بدست آمده از پیاده سازی پروژه می باشد:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Prediction Accuracy | Kappa | ROC | MCC | Recall | Precision | 10-fold cross |
| 82% | 55% | 75% | 58% | 54% | 88% | 80.527% |

نمودار Roc Area در پیاده سازی بنده به صورت شکل سمت چپ و در پیاده سازی مقاله به شکل سمت راست شد :



دقت پیاده سازی روش بنده 80.527% شد که در مقایسه با دقت مقاله که 95% بود 15 درصد کمتر شد.