

بسمه تعالی



**نام درس:**

**کاوش دادگان انبوه (Big data)**

**نام پروژه:**

**ساخت مدل پیش بینی بر روی یک مجموعه کلان داده (Big data) با استفاده از ابزار  
یادگیری ماشینی برای کلان داده**

**نام استاد درس:**

**دکتر الهام آخوندزاده**

**نام TA درس:**

**مهندس امیررضا نجفی**

**نام و شماره دانشجویی اعضای گروه:**

**سمیه حمیدی ۴۰۰۶۶۲۴۱۰۰۲**

**هاله خوش شانس ۴۰۰۶۶۲۴۱۶۰۴**

**جواد رفیعی فرد ۴۰۰۶۶۲۴۱۰۰۵**

**محمد رضا شاقوزی ۴۰۰۶۶۲۴۱۰۰۸**

بسمه تعالی

گزارش پروژه درس داده کاوی

ویرایش: ۱۴۰۱/۱۱/۲۲ - تدوین: ۱۴۰۱/۱۱/۰۶ - تهیه کننده: سمیه حمیدی، هاله خوش شانس، جواد

رفیعی فرد، محمدرضا شاقوزی

## فهرست

۴	۱ مقدمه .....
۵	۱-۱ مراحل نصب و راه اندازی spark .....
۷	۱-۲ مقایسه پلتفرم‌های Databricks و Zeppelin .....
۸	1-3 نصب Apache Zeppelin .....
۱۰	۲ معرفی مجموعه داده و توصیف آن .....
۱۰	۲-۱ معرفی کلی مجموعه داده و نحوه شکل گیری آن .....
۱۱	۳ تحلیل اکتشافی داده (Exploratory data analysis) .....
۱۱	۳-۱ معرفی ویژگی های ورودی .....
۱۷	3-2 ویژگی خروجی (هدف): .....
۱۸	۳-۳ مجموعه داده در یک نگاه .....
۱۹	4 فراخوانی کتابخانه های مورد نیاز و انجام تنظیمات pyspark .....
۱۹	۵ جمعیت چانک های مجموعه داده .....
۲۰	۶ مرحله پیش پردازش .....
۲۰	6-1 تبدیل به داده های عددی .....
۲۲	۶-۲ نرمال سازی .....
۲۲	6-3 تشخیص و حذف داده های پرت .....
۲۲	۷ داده های از دست رفته (Missing Values) .....
۲۲	7-1 بررسی داده های از دست رفته .....
۲۳	۷-۲ پر کردن داده های از دست رفته .....
۲۵	۸ مهندسی ویژگی (Feature Engineering) .....
۲۷	۹ مدل سازی .....
۲۷	۹-۱ روش رگرسیون منطقی (Logistic Regression) .....

۲۷	۹-۲ روش درخت تصمیم (Decision Tree)
۲۸	۹-۳ روش جنگل تصادفی (Random Forest)
۲۸	۹-۴ روش طبقه‌بند ماشین بردار پشتیبان خطی (Linear SVC)
۳۰	۱۰ ارزیابی
۳۰	10-1 سنجه Macro F1
۳۰	10-2 سنجه دقت (precision) یا positive predictive value (PPV)
	10-3 سنجه بازخوانی (Recall) یا sensitivity یا hit rate یا true positive rate (TPR)
۳۰	
۳۱	10-4 سنجه AUC score
۴۰	11 مقایسه عملکرد مدل‌ها

## ۱ مقدمه

در پروژه نهایی کاوش دادگان انبوه، قرار است با استفاده از ابزارهای big data به دنبال پردازش و ساخت طبقه‌بند برای مجموعه داده Epidemiologisches Krebsregister NRW باشیم. با توجه به اینکه مقدمات ابزارهایی مانند Apache Spark در طول ترم توسط جناب مهندس نجفی آموزش داده شد، در زمینه کدنویسی پروژه مشکلی پیش نیامد. اما بزرگترین چالش ما تأمین منابع برای اجرای عملیات‌های big data بود.

هدف ابتدایی گروه، این بود که یک پروژه در databricks ایجاد کنیم و به صورت یکپارچه بر روی امکانات apache spark کار کنیم. اما بدلیل خطای ۴۰۳ فیلترشکن در پلتفرم databricks اکثریت اعضا گروه نتوانستیم به این سایت دسترسی داشته باشیم. لذا تصمیم بر این شد که بصورت محلی (local) کار را پیش ببریم و کد را در github برای دسترسی همه اعضا به اشتراک بگذاریم.

The image shows two side-by-side screenshots. The left screenshot is the Databricks Community Edition login page. It features the Databricks logo, a 'Sign In to Databricks Community Edition' heading, and a login form with fields for email (sh.mohammad66@gmail.com) and password. A red error message box states: 'You entered an invalid email or password, or your workspace access is locked because of unusual activity. For help, see Community Edition Login Issues. Note: Emails/usernames are case-sensitive'. Below the form is a 'Sign In' button and a link for 'New to Databricks? Sign Up.' The right screenshot is a network monitoring tool interface. It shows a timeline at the top with a red vertical line at approximately 1000ms. Below the timeline is a table with two columns: 'Name' and 'Status'. The table contains two entries: 'pub-conf' with status '200' and 'j\_security\_check' with status '403'.

Name	Status
pub-conf	200
j_security_check	403

The image shows a screenshot of a web browser window. The address bar shows the URL 'http://localhost:4040/j\_security\_check'. The main content area displays a large red 'HTTP ERROR 403' message. Below the message, it says 'Problem accessing /j\_security\_check. Reason:'. The browser's developer tools are open, showing the 'Preview' tab selected. The 'Name' column on the left lists 'pub-conf' and 'j\_security\_check'. The 'Status' column shows '200' for 'pub-conf' and '403' for 'j\_security\_check'.

Name	Status
pub-conf	200
j_security_check	403

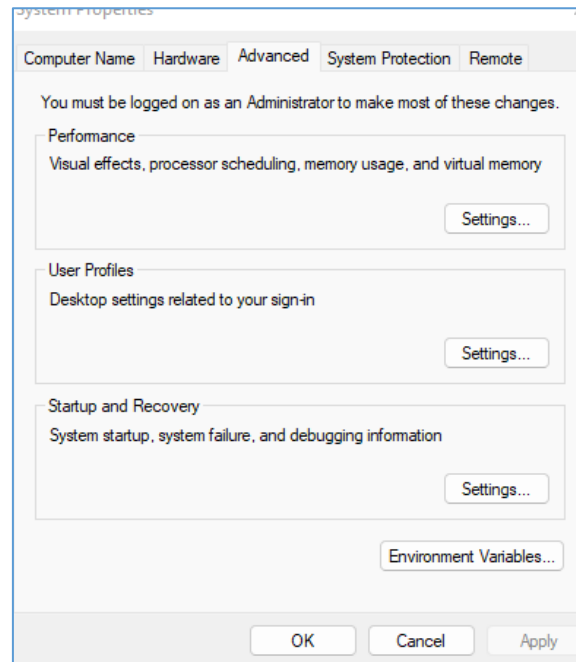
## ۱-۱ مراحل نصب و راه اندازی spark

ابتدا ابزارهای زیر را به ترتیب نصب کردیم:

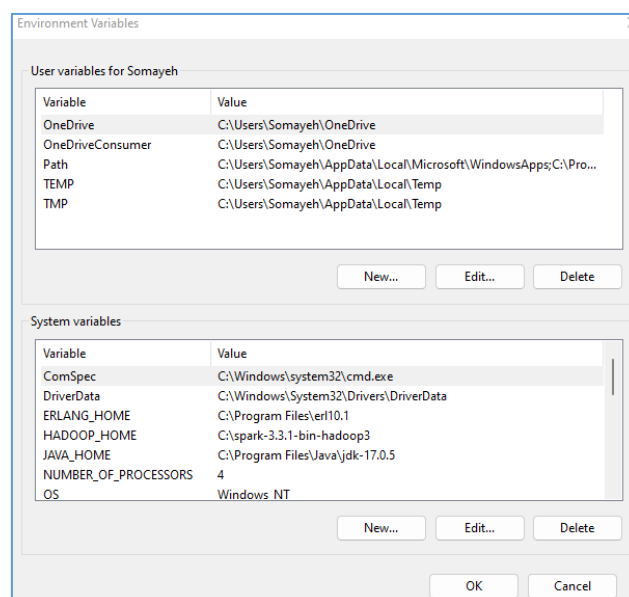
Java.SE.Development.Kit.17.0.5.x64  
Git-2.39.1-64-bit  
GitHubDesktopSetup-x64  
vscode

سپس فایل زیپ apache spark را از آدرس زیر دانلود و در درایو c کپی و استخراج کردیم.

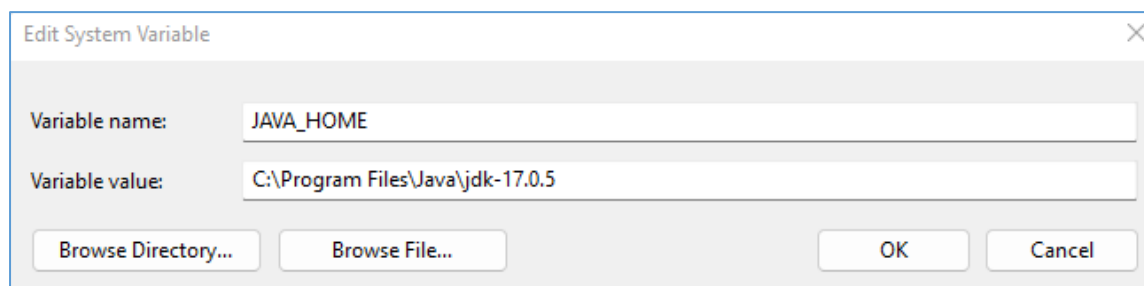
<https://spark.apache.org/downloads.html>



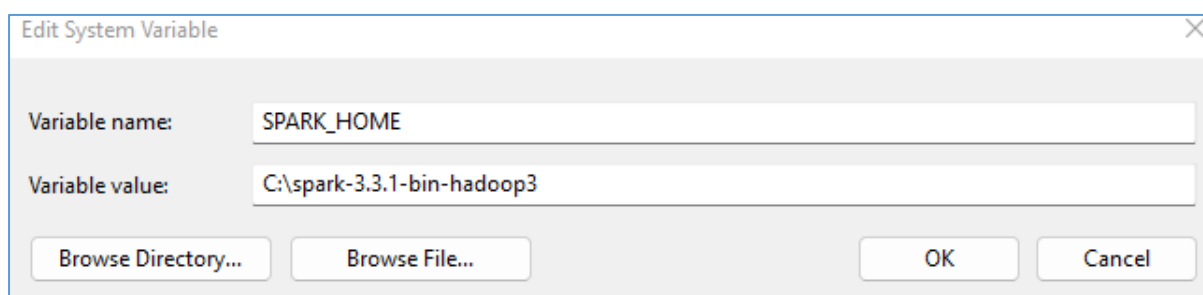
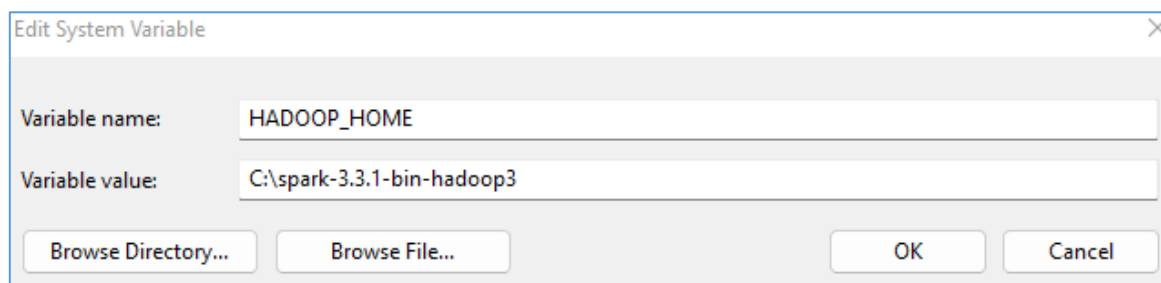
نسخه‌ای که در این پروژه استفاده کردیم، spark-3.3.1-bin-hadoop3.rar است. سپس مسیرهای پیش فرض را در EnvironMent Variables اضافه کردیم. برای این کار مراحل زیر را انجام شده است:



ابتدا system properties را باز و روی دکمه Environment Variables کلیک کرده در قسمت پایین دکمه new را انتخاب کردیم:



سپس مسیر ها به صورت زیر اضافه شدند:



نکته: variable value ها با توجه به مسیر نصب برای هر کس ممکن است متفاوت باشد. برای استفاده از پسته اسپارک در Command Prompt دستور spark-shell را اجرا و سپس دستور `pip install pyspark` را در همین محیط، برای نصب pyspark اجرا کردیم. سپس برای استفاده از کد موجود در github از آن یک شبیه سازی (Clone) در درایو c ایجاد شد. با اجرای دستور `cd c:/` در Command Prompt خط دستور (Command Line) را به درایو c منتقل و دستور زیر را در آن اجرا کردیم:

```
Git clone [ssh address in github for this project]
```

سپس برای استفاده از کد شبیه سازی شده در درایو c دستورات زیر به ترتیب در Command Prompt اجرا شد:

```
c:\>cd spark_project
```

```
c:\spark_project>code .
```

و برای استفاده از github desktop دستور زیر را اجرا کردیم:

```
c:\spark_project>github .
```

## ۱-۲ مقایسه پلتفرم‌های Databricks و Zeppelin

پس از بحث و مشورت اعضای گروه، اولین راه حلی که برای رفع مشکل databricks پیشنهاد شد، پیدا کردن پلتفرمی مشابه و در دسترس بود.

Databricks یک شرکت نرم افزاری آمریکایی است که توسط سازندگان آپاچی اسپارک تأسیس شده است. Databricks یک پلت فرم مبتنی بر وب برای کار با Spark ایجاد می کند که مدیریت خودکار خوشه و نوت بوک های سبک IPython را ارائه می دهد.

این شرکت که در زمینه داده های بزرگ (BigData) فعالیت می کند، عمده ابزار هایی که توسعه می دهد را به صورت عمومی منتشر کرده است.

انبارهای داده (data warehouses) برای داده های ساختاریافته ایده آل هستند، اما برای رسیدگی به داده های بدون ساختار، داده های نیمه ساختار یافته و داده هایی با تنوع، سرعت و حجم بالا، دریاچه های داده (Data Lakes) معرفی شده اند.

پلتفرم Databricks Lakehouse بهترین ویژگی های دریاچه های داده و انبارهای داده را برای ارائه قابلیت اطمینان، حاکمیت قوی، انعطاف پذیری و پشتیبانی از یادگیری ماشینی، ترکیب می کند.

اما پلتفرم Apache Zeppelin یک web based notebook مشابه databricks است که ویژگی هایی مانند data exploration, visualization, sharing and collaboration را فراهم می کند. همچنین از زبان های python, scala, hive, sparksql, shell, markdown پشتیبانی می کند. مزیت اصلی آن منبع باز بودن است که می توان آن را بر روی سرور به صورت اختصاصی مستقر کرد.

Apache Spark یک سیستم محاسباتی خوشه ای سریع و همه منظوره است که API های سطح بالا را که از نمودارهای اجرای عمومی پشتیبانی می کند در جاوا، اسکالا، پایتون و R و یک موتور بهینه سازی شده ارائه می دهد. Apache Spark در Zeppelin پشتیبانی می شود.

### مقایسه zeppelin و Databricks:

	zeppelin	Databricks
setup	خود میزبان (Self-hosted)	خود میزبان یا کاملاً مدیریت شده
JUPYTER COMPATIBILITY	ندارد	سازگار با ژوپیتِر (JUPYTER)
PROGRAMMING LANGUAGES	Python, SQL, Spark	JUPYTER
DATA VISUALIZATION	ناشناخته	مصور سازی با UI
COLLABORATIVE EDITING	مبتنی بر فایل یا نا همزمان	بلادرنگ

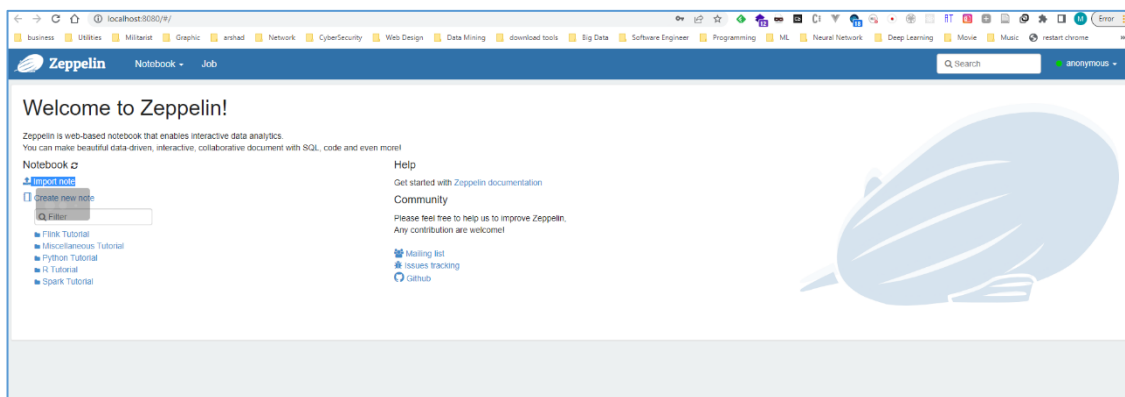
<b>PRICING</b>	رایگان	رایگان و دارای هزینه
<b>LICENSE</b>	منبع باز	اختصاصی

### ۳-۱ نصب Apache Zeppelin

در این پروژه ما از طریق docker، این پلتفرم را با دستور زیر بر روی سیستم محلی (local) نصب و مستقر کردیم:

```
docker run -p 8080:8080 --rm --name zeppelin apache/zeppelin:0.10.1
```

پس از استقرار با مراجعه به آدرس localhost:8080 با صفحه اصلی zeppelin مواجه شدیم:



می توان notebook ایجاد کرد و یا با گزینه import از notebook قبلا ساخته شده استفاده نمود. همچنین به دلیل اینکه از چندین زبان مختلف پشتیبانی می کند، در خط اول هر cell بهتر است نوع پردازش آن سلول با % مشخص شود.



```
%spark.pyspark
from pyspark.sql import SparkSession
from pyspark import SparkConf, SparkContext
from pyspark.sql.types import StructType,IntegerType,FloatType,BooleanType,StringType
from pyspark.sql.functions import rand

def load_data(files,schema):
    df=spark.read.csv(files,header=True
    ,schema=schema)
    return df

def load_record_linkage_data():
    schema = StructType() \
        .add("id_1",IntegerType(),True) \
        .add("id_2",IntegerType(),True) \
        .add("cmp_fname_c1",FloatType(),True) \
        .add("cmp_fname_c2",FloatType(),True) \
        .add("cmp_lname_c1",FloatType(),True) \
        .add("cmp_lname_c2",FloatType(),True) \
        .add("cmp_sex",IntegerType(),True) \
        .add("cmp_bd",IntegerType(),True) \
        .add("cmp_bm",IntegerType(),True) \
        .add("cmp_by",IntegerType(),True) \
        .add("cmp_plz",IntegerType(),True) \
        .add("is_match",BooleanType(),False)
    files=[f'/usr/share/data/block_{id}.csv' for id in range(1,11)]
    return load_data(files,schema=schema)
```

Took 11 sec. Last updated by anonymous at February 11 2023, 6:22:13 PM.

پس از کار با zeppelin، مشاهده شد که مصورسازی با محدودیت ۱۰۰۰ ردیف مواجه است که با تغییر تنظیمات این مورد برطرف شد. اما با توجه به منابع محدودی که سیستم داشت، خروجی نمودارها به صورت truncated براساس بایت نمایش داده می‌شد.



به دلیل محدودیت‌های مصورسازی، به دنبال رفع مشکل ۴۰۳ پلتفرم databricks اقدام کرده و به عنوان مرجعی برای مصورسازی از آن استفاده کردیم. کدهای عملیاتی تا قبل از این مرحله انجام شده و خروجی‌ها کامل بود.

## ۲ معرفی مجموعه داده و توصیف آن

### ۲-۱ معرفی کلی مجموعه داده و نحوه شکل گیری آن

در این تحقیق قصد داریم یک پروژه یادگیری ماشینی را بر روی یک مجموعه کلان داده (Big data) توسط روش‌های پردازش کلان داده انجام دهیم. این مجموعه داده که از سایت Machine Learning Repository استخراج شده، داده‌های مربوط به تعداد زیادی از بیماران در یک مرکز تحقیقات بیماران سرطانی در آلمان به نام Epidemiologisches Krebsregister NRW است و در آدرس زیر قابل دسترسی است:

<https://archive.ics.uci.edu/ml/machine-learning-databases/00210/donation.zip>

موضوع از این قرار است که اطلاعات ۱۰۰,۰۰۰ بیمار بین سال‌های ۲۰۰۵ تا ۲۰۰۸ توسط کارمندان مختلف مرکز تحقیقات ثبت شده است. مشکل اینجاست که اطلاعات یک فرد ممکن است دوبار توسط کارمندان ثبت شده باشد و مشکل اساسی تر اینکه در این بین اشتباهات تایپی نیز وجود دارد.

بنابراین مسأله این مجموعه داده پیدا کردن جفت رکوردهایی از اطلاعات است که متعلق به یک شخص است.

پس از آن در سال ۲۰۰۸ موسسه آمار زیستی پزشکی، اپیدمیولوژی و انفورماتیک (IMBEI) و مرکز پزشکی دانشگاه یوهانس گوتنبرگ، ماینس، آلمان بر اساس ۶ معیار که در ادامه می‌آید تمام جفت رکوردهای مربوط به این ۱۰۰,۰۰۰ رکورد (بیمار) را مورد بررسی قرار دادند. معیارهای مذکور به شرح زیر هستند:

- ۱- برابری آوایی نام و نام خانوادگی + برابری تاریخ تولد
- ۲- برابری آوایی نام + برابری روز تولد (ماه و سال می‌توانند متفاوت باشند)
- ۳- برابری آوایی نام + برابری ماه تولد (روز و سال می‌توانند متفاوت باشند)
- ۴- برابری آوایی نام + برابری سال تولد (روز و ماه می‌توانند متفاوت باشند)
- ۵- برابری تاریخ تولد
- ۶- برابری آوایی نام خانوادگی + برابری جنسیت

برقراری هر کدام از شروط فوق منجر به انتخاب جفت رکورد شد. در نتیجه ۵,۷۴۹,۱۳۲ جفت رکورد از میان تمام جفت رکوردها انتخاب شد که از این بین در ۲۰,۹۳۱ جفت رکورد، شخص اول و شخص دوم هر دو یک نفر بودند که برچسب match به آنها تعلق گرفت یعنی این جفت رکورد مربوط به یک شخص است و در مابقی جفت رکوردها شخص اول و شخص دوم دو فرد متفاوت بودند.

مشخص نیست برچسب گذاری به چه روشی انجام شده و چه سنجه‌هایی برای تعیین برچسب در نظر گرفته شده است اما به نظر می‌آید سنجه‌های زیادی در نظر گرفته شده و برچسب گذاری توسط روش‌های جمع‌سپاری انجام شده باشد.

این مجموعه داده به ۱۰ فایل با اندازه تقریباً برابر تقسیم شده که در هر فایل نسبت زوج نفرات match و non-match تقریباً برابر است.

لازم به ذکر است در مجموعه داده اولیه هر رکورد متعلق به یک بیمار بود که شامل ۱۰۰,۰۰۰ رکورد بود ولی پس از بررسی جفت رکوردها مجموعه داده جدیدی تشکیل شد که هر رکورد متعلق به رکورد مجموعه داده اولیه است که باهم مقایسه می شوند که تعداد آنها ۵,۷۴۹,۱۳۲ رکورد است.

از اینجا به بعد منظور از عبارت رکورد، هر سطر از مجموعه داده جدید (نهایی) است. هدف از مجموعه داده نهایی یافتن یک مدل یادگیری ماشینی بر اساس کلان داده برای پیش بینی وضعیت تطابق دو شخص بر اساس ویژگی های آنهاست و اینطور به نظر می آید که علت استفاده از این معیارها کم کردن تعداد زوج نفرات مورد استفاده است. با توجه به تعداد رکوردها، تعداد کل زوج رکوردهای ممکن تقریباً ۱۰ میلیارد خواهد شد که پردازش این تعداد بسیار زمان بر است و نیاز به منابع زیادی دارد. با وجود اینکه ۶ معیار برای کاهش تعداد زوج نفرات استفاده شده باز هم تعداد رکوردها بسیار زیاد است و نیاز است از روش های پردازش کلان داده استفاده شود. پس از ساخت مدل با این تعداد رکورد می توان آن را برای بقیه زوج نفرات استفاده نمود.

در ادامه به معرفی این ویژگی ها می پردازیم:

مجموعه داده نهایی شامل ۱۲ ویژگی است که ۲ ویژگی اول آن شماره شناسایی دو شخص مورد مقایسه است که برای پیش بینی مورد استفاده قرار نمی گیرد ۹ ویژگی، ویژگی های ورودی مورد استفاده جهت پیش بینی و ۱ ویژگی، ویژگی خروجی است که match بودن یا نبودن دو شخص را مشخص می کند.

### ۳ تحلیل اکتشافی داده (Exploratory data analysis)

از آنجا که کتابخانه pyspark متدی برای مصورسازی کلان داده ندارد و با توجه به اینکه databricks از ابزار پردازش کلان داده استفاده می کند با وجود اینکه چند کتابخانه را برای مصور سازی آزمایش کردیم ولی منابع سیستم محلی جوابگو نبودند لذا از databricks برای مصورسازی مجموعه داده استفاده کردیم.

#### ۳-۱ معرفی ویژگی های ورودی

۱- id\_1:

شماره شناسایی شخص اول که یک متغیر عددی و از نوع صحیح است.

۲- id\_2:

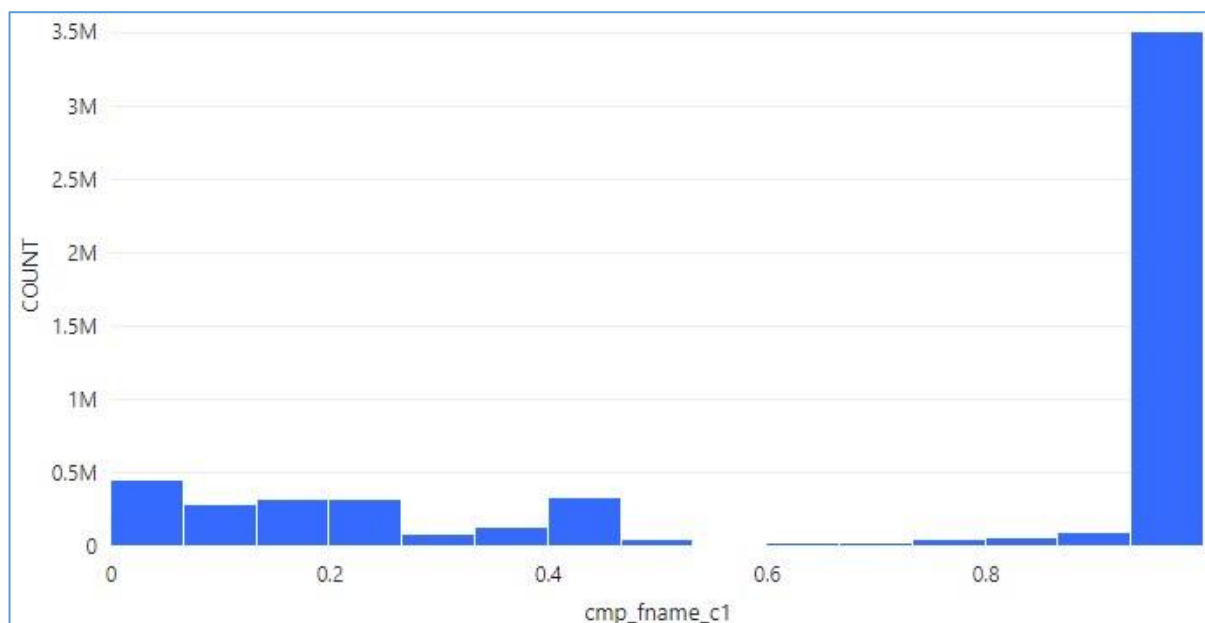
شماره شناسایی شخص دوم که یک متغیر عددی و از نوع صحیح است.

این دو ویژگی در ساخت مدل شرکت نمی کنند و در پیش پردازش از مجموعه داده حذف خواهند شد.

۳- cmp\_fname\_c1:

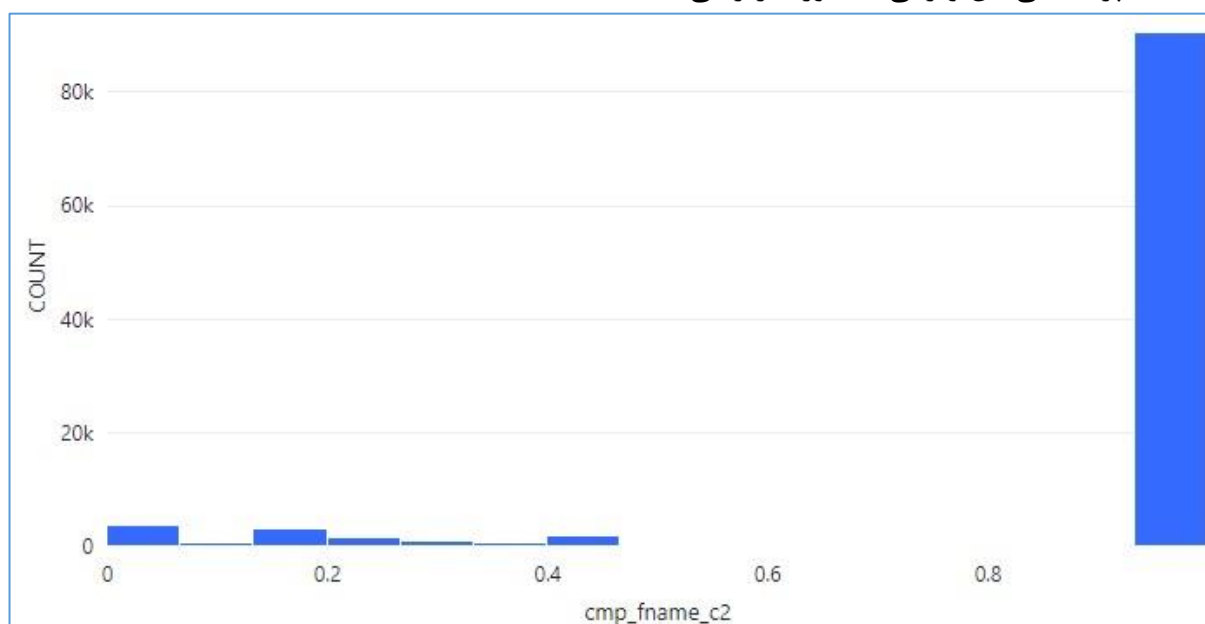
میزان مشابهت جزء اول نام دو نفر که یک متغیر عددی پیوسته و از نوع اعشاری است. نمودار

پراکندگی این ویژگی به صورت زیر می باشد:



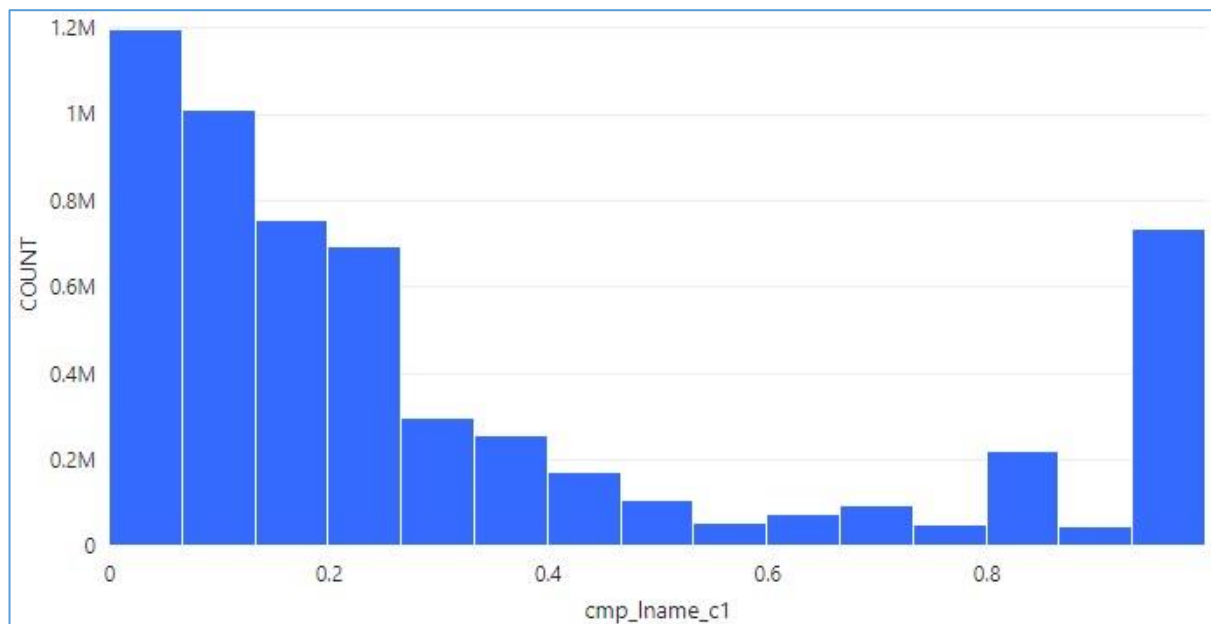
#### ۴- `cmp_fname_c2`:

میزان مشابهت جزء دوم نام دو نفر که یک متغیر عددی پیوسته و از نوع اعشاری است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



#### ۵- `cmp_lname_c1`:

میزان مشابهت جزء اول نام خانوادگی دو نفر که یک متغیر عددی پیوسته و از نوع اعشاری است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



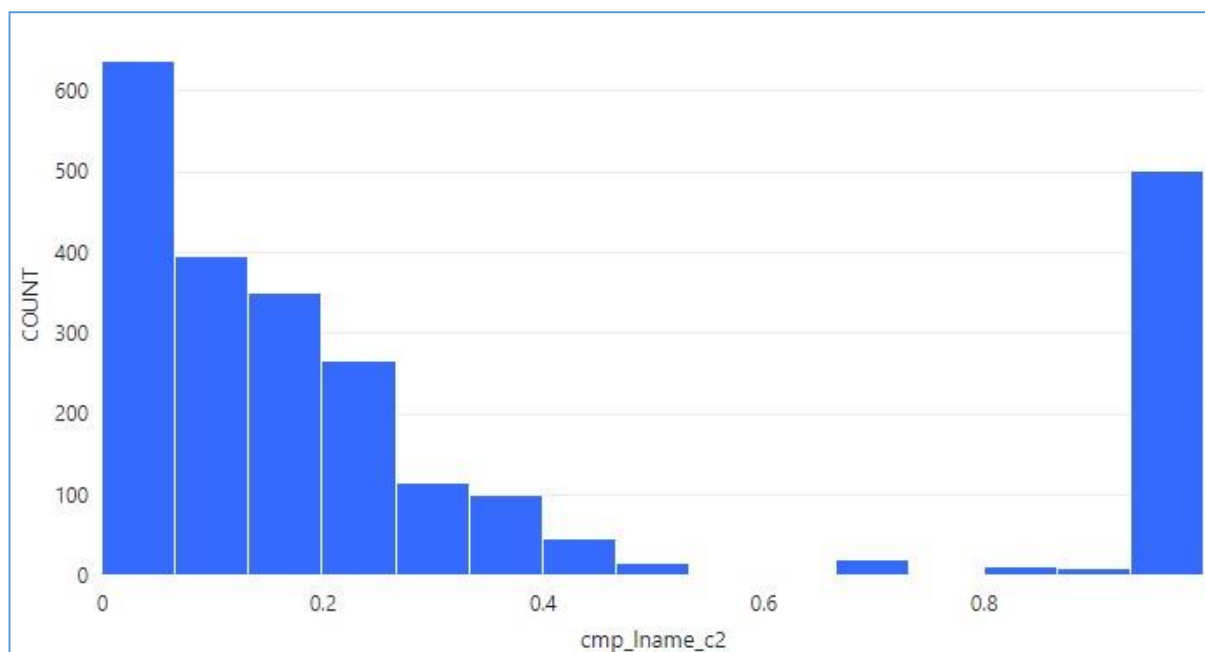
#### ۶- cmp\_lname\_c2:

میزان مشابهت جزء دوم نام خانوادگی دو نفر که یک متغیر عددی پیوسته و از نوع اعشاری است. (Sariyar *et al.*, ۲۰۱۲) که تحقیق خود را بر روی همین مجموعه داده انجام داده اند اشاره می کنند که نام و نام خانوادگی در این مجموعه داده هر کدام دارای دو جزء هستند که پسوند c2 نشانه جزء دوم نام یا نام خانوادگی است.

An example of a realistic and problematic record linkage task regarding personal data is given below with the following attributes: first name and last name (two **components** each), sex, date of birth (comprised of day, month and year) and postal code.

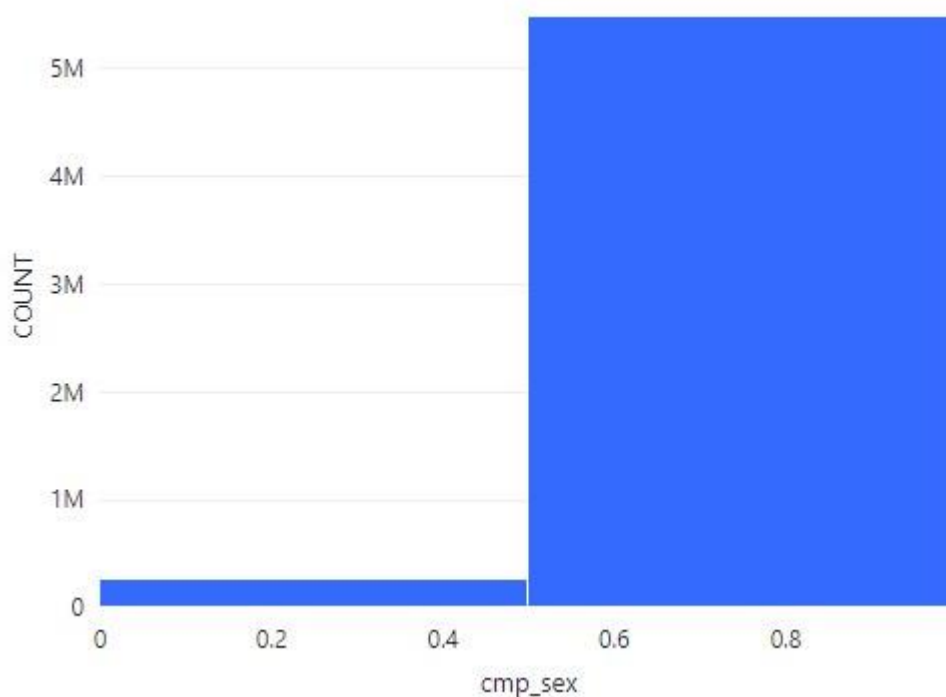
```
(( 'Peter', 'John', 'Branket', , 'm', '11', '10', '1971', '100098' )
( 'Peter', , 'Blanket', , 'm', '01', '10', '1971', '10098' ))
```

نمودار پراکندگی این ویژگی به صورت زیر می باشد:



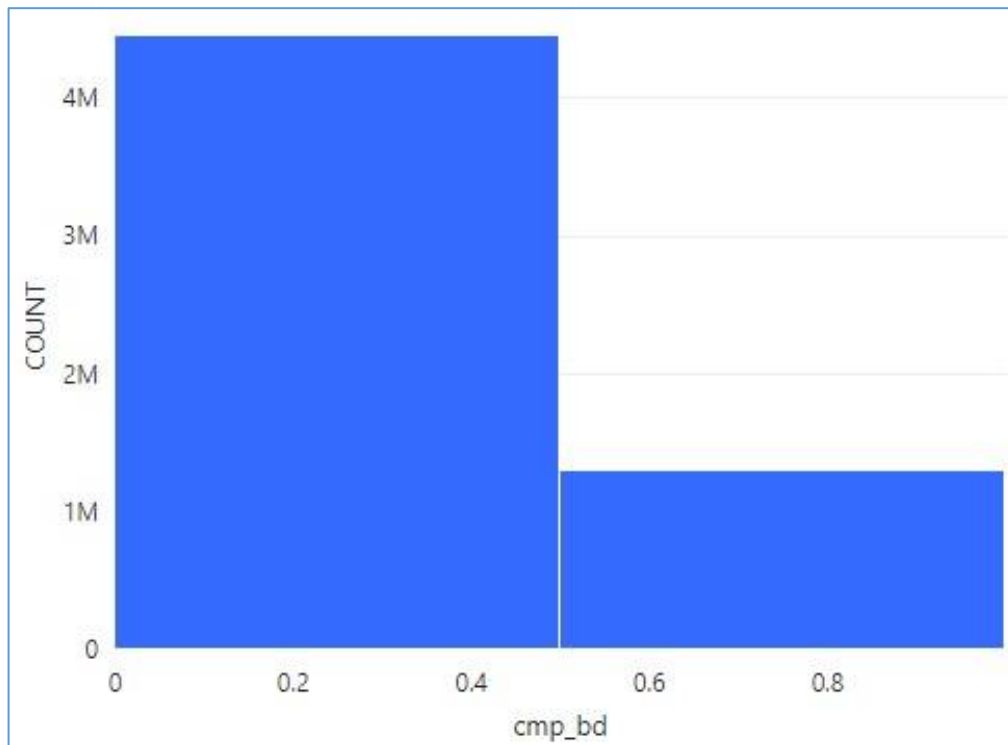
#### ۷- cmp\_sex:

تشابه جنسیت دو نفر که یک متغیر باینری است و مقدار آن 0 یا 1 است که 0 به معنی عدم یکسانی جنسیت و 1 به معنی یکسان بودن جنسیت دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



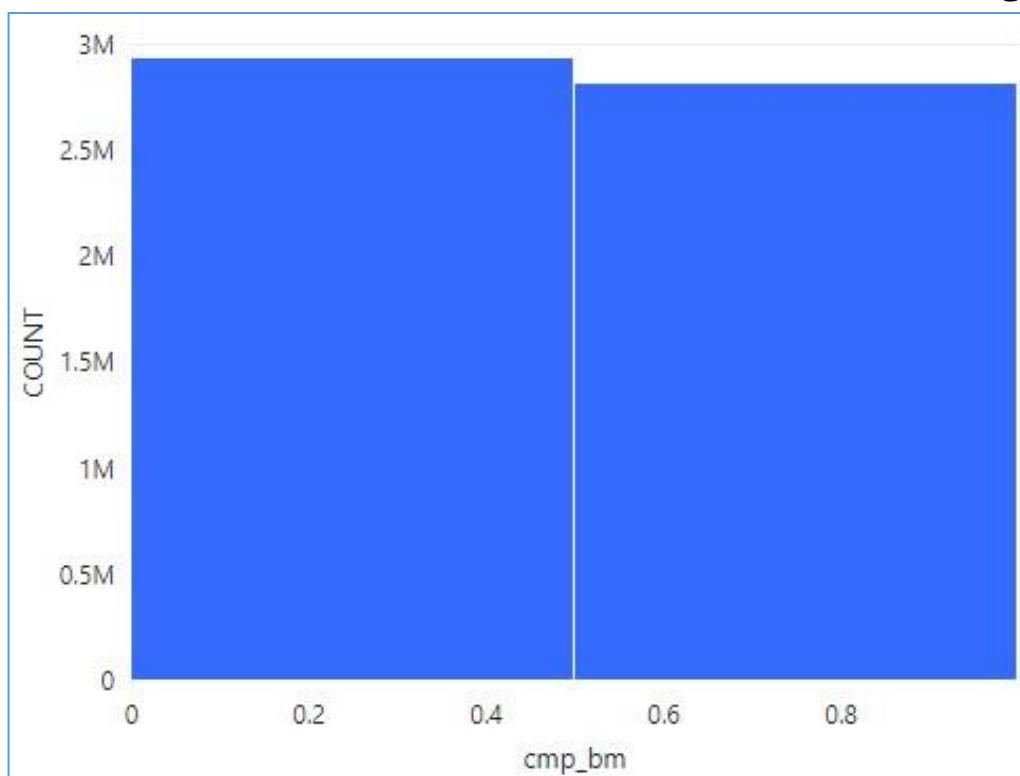
#### ۸- cmp\_bd:

تشابه روز تولد دو نفر که یک متغیر باینری است و مقدار آن 0 یا 1 است که 0 به معنی عدم یکسانی روز تولد و 1 به معنی یکسان بودن روز تولد دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



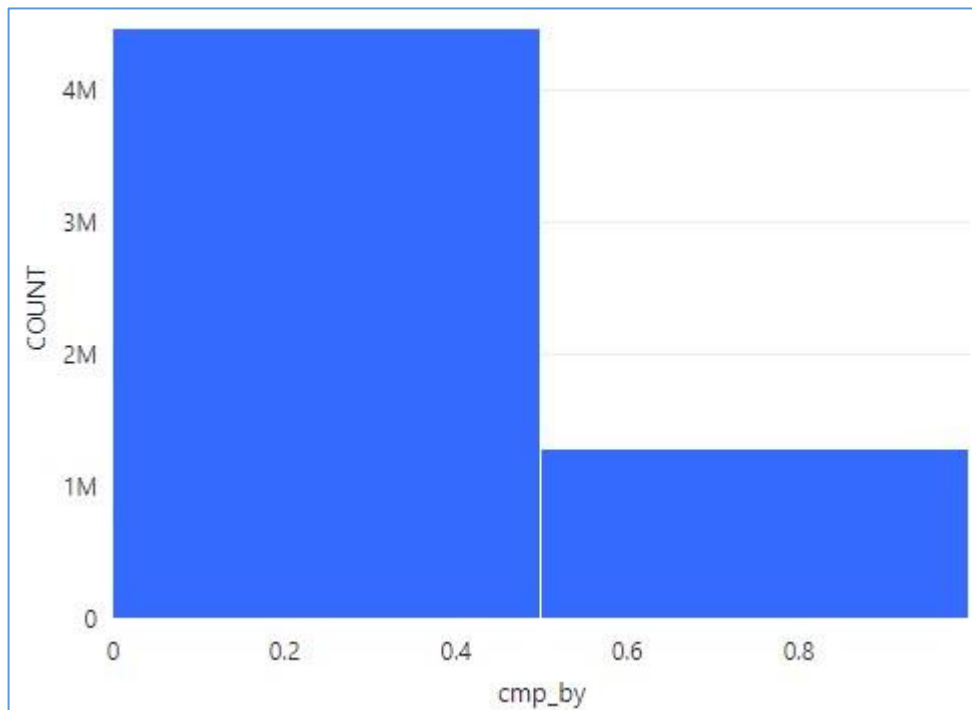
۹- cmp\_bm :

تشابه ماه تولد دو نفر که یک متغیر باینری است و مقدار آن 0 یا 1 است که 0 به معنی عدم یکسانی ماه تولد و 1 به معنی یکسان بودن ماه تولد دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



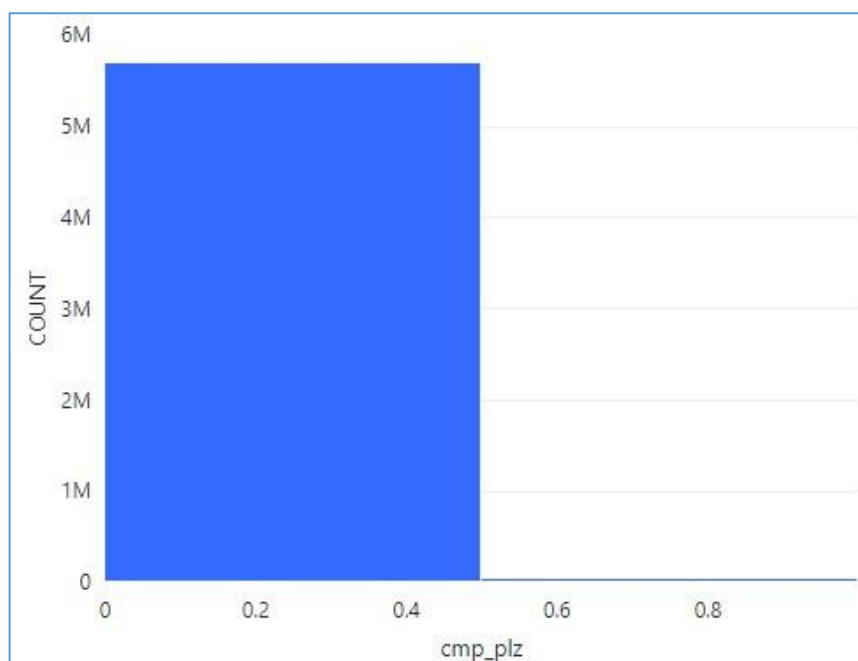
#### ۱۰ - cmp\_by :

تشابه سال تولد دو نفر که یک متغیر باینری است و مقدار آن 0 یا 1 است که 0 به معنی عدم یکسانی سال تولد و 1 به معنی یکسان بودن سال تولد دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



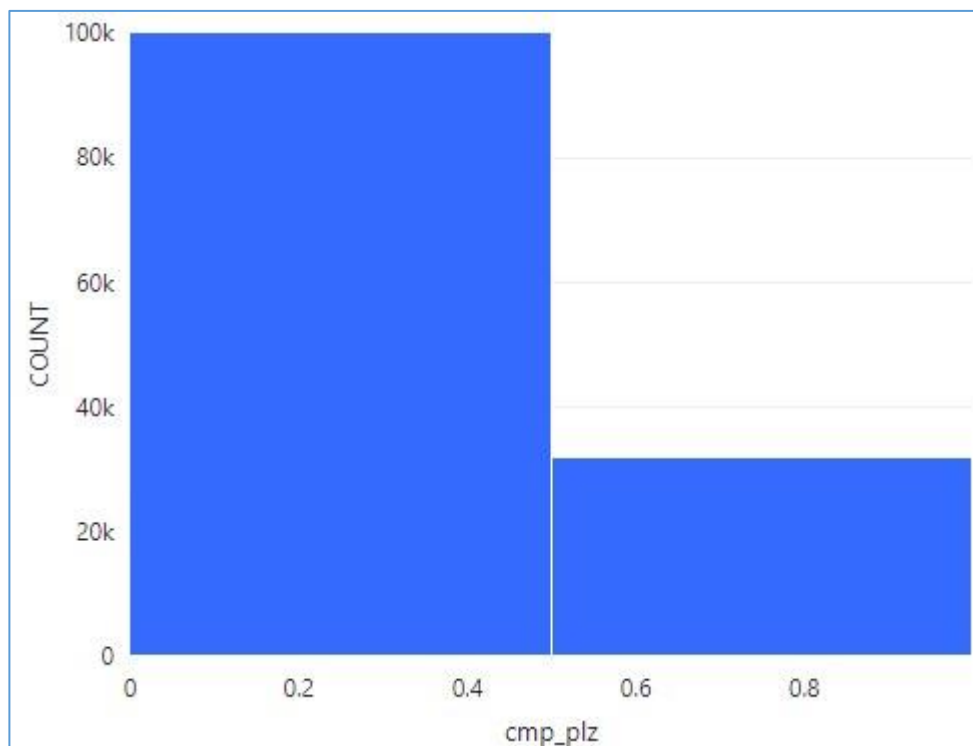
#### ۱۱ - cmp\_plz :

تشابه کدپستی دو نفر که یک متغیر باینری است و مقدار آن 0 یا 1 است که 0 به معنی عدم یکسانی کدپستی و 1 به معنی یکسان بودن سال تولد دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:





با توجه به اینکه تعداد کلاس 1 یعنی رکوردهای دارای تطابق خیلی کم است در نمودار پایین با مقیاس بزرگتر مشخص شده است:

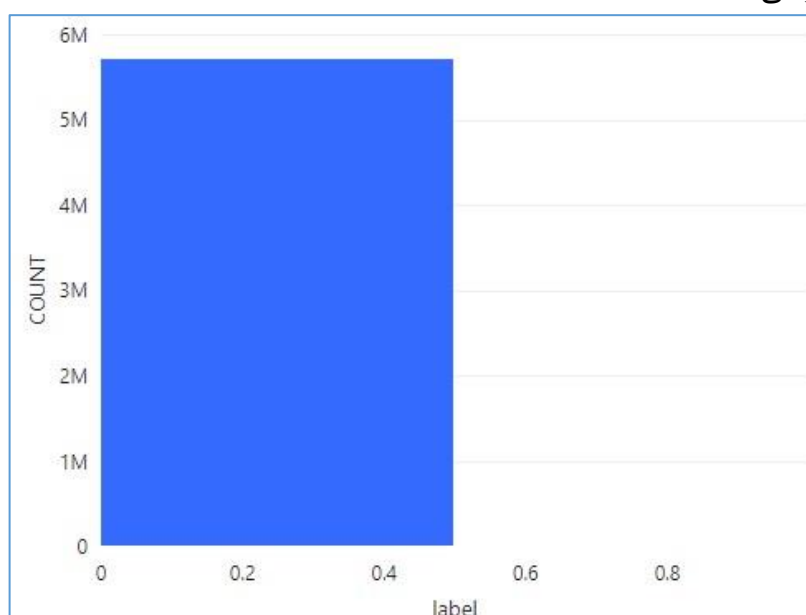


۹ ویژگی فوق، در ساخت مدل شرکت می کنند.

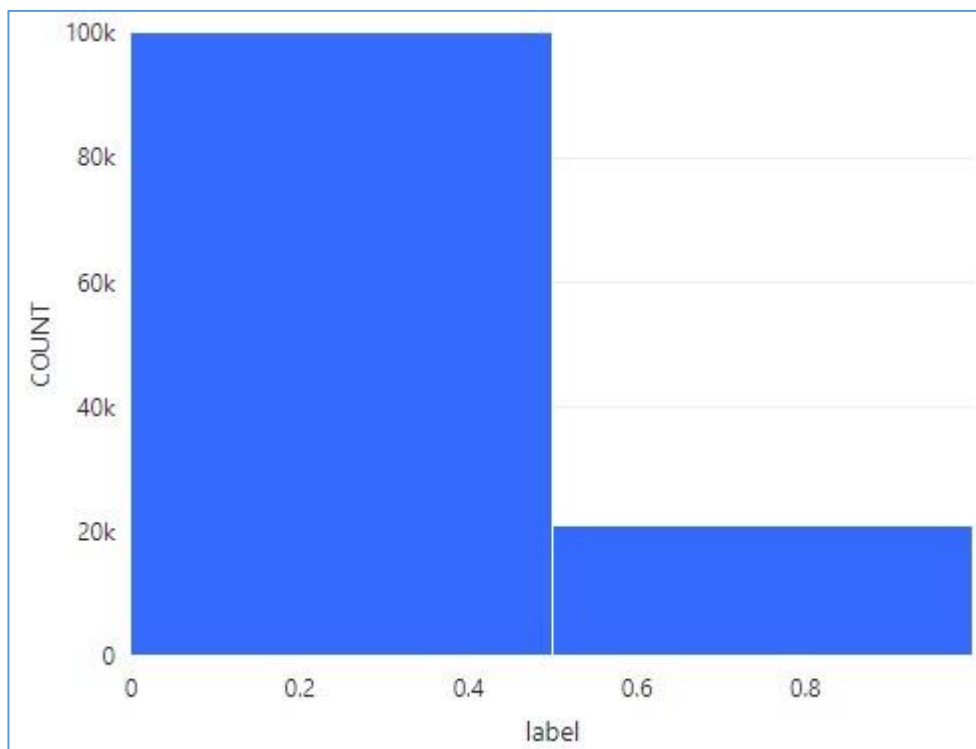
## ۳-۲ ویژگی خروجی (هدف):

۱۲ - is\_match:

وضعیت تطابق دو نفر است که یک متغیر باینری است و مقدار آن True یا False است که True به معنی تطابق دو نفر و False به معنی عدم تطابق دو نفر است. نمودار پراکندگی این ویژگی به صورت زیر می باشد:



با توجه به اینکه تعداد کلاس 1 یعنی رکوردهای دارای تطابق خیلی کم است در نمودار پایین با مقیاس بزرگتر مشخص شده است:

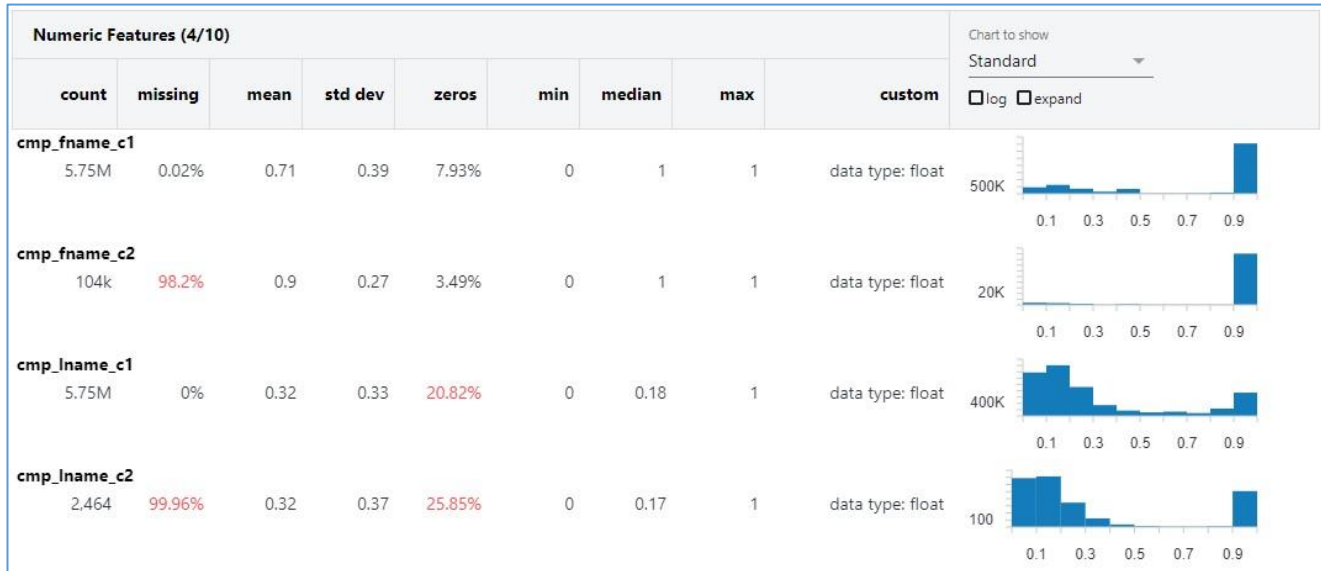


### ۳-۳ مجموعه داده در یک نگاه

مشخصات آماری ۶ ویژگی صحیح (int) به صورت زیر می باشد:

Numeric Features (6/10)									Chart to show
count	missing	mean	std dev	zeros	min	median	max	custom	Standard
									<input type="checkbox"/> log <input type="checkbox"/> expand
<b>cmp_sex</b>									
5.75M	0%	0.96	0.21	4.5%	0	1	1	data type: int	
<b>cmp_bd</b>									
5.75M	0.01%	0.22	0.42	77.55%	0	0	1	data type: int	
<b>cmp_bm</b>									
5.75M	0.01%	0.49	0.5	51.11%	0	0	1	data type: int	
<b>cmp_by</b>									
5.75M	0.01%	0.22	0.42	77.73%	0	0	1	data type: int	
<b>cmp_plz</b>									
5.74M	0.22%	0.01	0.07	99.45%	0	0	1	data type: int	
<b>label</b>									
5.75M	0%	0	0.06	99.64%	0	0	1	data type: int	

همچنین مشخصات آماری ۴ ویژگی اعشاری (float) به صورت زیر می باشد:













## ۴ فراخوانی کتابخانه های مورد نیاز و انجام تنظیمات pyspark

ابتدا توسط کد زیر متدها و کتابخانه های مورد نیاز را فراخوانی نمودیم. همچنین تنظیمات اولیه مورد نیاز برای کتابخانه pyspark را انجام دادیم:

```
from pyspark.sql import SparkSession
from pyspark import SparkConf, SparkContext
from pyspark.sql.types import StructType, IntegerType, FloatType, BooleanType, StringType
from pyspark.sql.functions import rand, count, isnull, when, col
conf = SparkConf().setMaster("local[*]").setAppName("My App")
sc = SparkContext.getOrCreate(conf = conf)
sc._conf.set('spark.executor.memory', '15g')\
    .set('spark.driver.memory', '15g')\
    .set('spark.driver.maxResultsSize', '0')
spark=SparkSession.builder\
    .appName('myApp')\
    .config("spark.driver.memory", "15g")\
    .getOrCreate()
```

## ۵ جمع چانک های مجموعه داده

مجموعه داده شامل ده فایل با مشخصات زیر می باشد:

 block_1.csv	2011/03/09 3:31 PM	CSV File	25,634 KB
 block_2.csv	2011/03/09 3:31 PM	CSV File	25,627 KB
 block_3.csv	2011/03/09 3:31 PM	CSV File	25,638 KB
 block_4.csv	2011/03/09 3:31 PM	CSV File	25,633 KB
 block_5.csv	2011/03/09 3:32 PM	CSV File	25,635 KB
 block_6.csv	2011/03/09 3:32 PM	CSV File	25,641 KB
 block_7.csv	2011/03/09 3:32 PM	CSV File	25,647 KB
 block_8.csv	2011/03/09 3:32 PM	CSV File	25,639 KB
 block_9.csv	2011/03/09 3:33 PM	CSV File	25,639 KB
 block_10.csv	2011/03/09 3:33 PM	CSV File	25,641 KB

این چانک ها را توسط کد زیر تجمیع کرده و در یک متغیر دیتافریم ذخیره کردیم:

```
def load_data(files,schema):
    df=spark.read.csv(files,header=True
                      ,schema=schema)
    return df

def load_record_linkage_data():
    schema = StructType() \
        .add("id_1",IntegerType(),True) \
        .add("id_2",IntegerType(),True) \
        .add("cmp_fname_c1",FloatType(),True) \
        .add("cmp_fname_c2",FloatType(),True) \
        .add("cmp_lname_c1",FloatType(),True) \
        .add("cmp_lname_c2",FloatType(),True) \
        .add("cmp_sex",IntegerType(),True) \
        .add("cmp_bd",IntegerType(),True) \
        .add("cmp_bm",IntegerType(),True) \
        .add("cmp_by",IntegerType(),True) \
        .add("cmp_plz",IntegerType(),True) \
        .add("is_match",BooleanType(),False)
    files=[f'./data/block_{id}.csv' for id in range(1,11)]
    return load_data(files,schema=schema)
df=load_record_linkage_data()
```

## ۶ مرحله پیش پردازش

### ۶-۱ تبدیل به داده‌های عددی

تمام ویژگی‌های این مجموعه داده از قبل عددی شده اند بجز خروجی که به صورت منطقی و باینری است:

```
df.printSchema()
```

```

root
|-- id_1: integer (nullable = true)
|-- id_2: integer (nullable = true)
|-- cmp_fname_c1: float (nullable = true)
|-- cmp_fname_c2: float (nullable = true)
|-- cmp_lname_c1: float (nullable = true)
|-- cmp_lname_c2: float (nullable = true)
|-- cmp_sex: integer (nullable = true)
|-- cmp_bd: integer (nullable = true)
|-- cmp_bm: integer (nullable = true)
|-- cmp_by: integer (nullable = true)
|-- cmp_plz: integer (nullable = true)
|-- is_match: boolean (nullable = true)

```

```
df.show(10)
```

id_1	id_2	cmp_fname_c1	cmp_fname_c2	cmp_lname_c1	cmp_lname_c2	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz	is_match
3148	8326	1.0	null	1.0	null	1	1	1	1	1	true
14055	94934	1.0	null	1.0	null	1	1	1	1	1	true
33948	34740	1.0	null	1.0	null	1	1	1	1	1	true
946	71870	1.0	null	1.0	null	1	1	1	1	1	true
64880	71676	1.0	null	1.0	null	1	1	1	1	1	true
25739	45991	1.0	null	1.0	null	1	1	1	1	1	true
62415	93584	1.0	null	1.0	null	1	1	1	1	0	true
27995	31399	1.0	null	1.0	null	1	1	1	1	1	true
4909	12238	1.0	null	1.0	null	1	1	1	1	1	true
15161	16743	1.0	null	1.0	null	1	1	1	1	1	true

only showing top 10 rows

و آن را توسط تابع زیر عددی کردیم:

```

from pyspark.sql.functions import when, lit

def convert_label_binary(input_df):
    temp = input_df.withColumn('label',
                               when(input_df['is_match']==True,
                                    lit(1)).otherwise(0)
                               )
    return temp
numerical_df = convert_label_binary(df).drop('is_match')

```

که نتیجه آن به صورت زیر می باشد:

```
numerical_df.printSchema()
```

```

root
|-- id_1: integer (nullable = true)
|-- id_2: integer (nullable = true)
|-- cmp_fname_c1: float (nullable = true)
|-- cmp_fname_c2: float (nullable = true)
|-- cmp_lname_c1: float (nullable = true)
|-- cmp_lname_c2: float (nullable = true)
|-- cmp_sex: integer (nullable = true)
|-- cmp_bd: integer (nullable = true)
|-- cmp_bm: integer (nullable = true)
|-- cmp_by: integer (nullable = true)
|-- cmp_plz: integer (nullable = true)
|-- label: integer (nullable = false)

```

```
numerical_df.show(10)
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id_1 | id_2 | cmp_fname_c1 | cmp_fname_c2 | cmp_lname_c1 | cmp_lname_c2 | cmp_sex | cmp_bd | cmp_bm | cmp_by | cmp_plz | label |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 3148 | 8326 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 14055 | 94934 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 33948 | 34740 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 946 | 71870 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 64880 | 71676 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 25739 | 45991 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 62415 | 93584 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 0 | 1 |
| 27995 | 31399 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 4909 | 12238 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
| 15161 | 16743 | 1.0 | null | 1.0 | null | 1 | 1 | 1 | 1 | 1 | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 10 rows

```

## ۶-۲ نرمال سازی

ویژگی ها همه نرمال شده و بین 0 و 1 قرار دارند.

## ۶-۳ تشخیص و حذف داده های پرت

با توجه به اینکه همه ویژگی ها از جنس شباهت و بر اساس مقایسه بوده و پس از نرمال سازی در بازه معنی دار و معین 0 و 1 قرار دارند، لذا داده پرتی وجود ندارد.

## ۷ داده های از دست رفته (Missing Values)

### ۷-۱ بررسی داده های از دست رفته

از آنجا که ویژگی id برای هر فرد و ترکیب id\_1 و id\_2 برای هر رکورد منحصر به فرد است همانگونه که در بخش های قبلی به آن اشاره شد در ساخت مدل ارزشی ایجاد نمی کنند و مشارکت ندارند. لذا برای ایجاد مدل باید حذف شوند که این کار را با کد زیر انجام دادیم:

```
no_id_numerical_df = numerical_df.drop('id_1','id_2')
```

نتیجه آن به صورت زیر است:

```
no_id_numerical_df.show(10)
```

cmp_fname_c1	cmp_fname_c2	cmp_lname_c1	cmp_lname_c2	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz	label
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	0	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1

توسط کد زیر وضعیت داده‌های از دست رفته را در مجموعه داده بررسی می‌کنیم:

```
numerical_df.select([count(when(isnull(column), column)).alias(column) for  
column in numerical_df.columns]).show()
```

id_1	id_2	cmp_fname_c1	cmp_fname_c2	cmp_lname_c1	cmp_lname_c2	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz	label
0	0	1007	5645434	0	5746668	0	795	795	795	12843	0

## ۷-۲ پر کردن داده‌های از دست رفته

همانگونه که در قسمت قبل ذکر شد مجموعه داده دارای داده‌های از دست رفته زیادی مخصوصاً در ویژگی‌های cmp\_fname\_c2 و cmp\_lname\_c2 است:

```
no_id_numerical_df.show(10)
```

cmp_fname_c1	cmp_fname_c2	cmp_lname_c1	cmp_lname_c2	cmp_sex	cmp_bd	cmp_bm	cmp_by	cmp_plz	label
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	0	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1
1.0	null	1.0	null	1	1	1	1	1	1

نمونه داده قبل از پر کردن داده‌های از دست رفته

با استفاده از روش Imputer داده‌های از دست رفته را پر کردیم که کد آن به صورت زیر می‌باشد:

```

from pyspark.ml.feature import Imputer

def fill_missing_values(input_df):
    # for float variables
    miss_df=input_df.drop('id_1','id_2')
    miss_df=miss_df.replace('?',None)
    float_cols=[
        'cmp_fname_c1',
        'cmp_fname_c2',
        'cmp_lname_c1',
        'cmp_lname_c2',
    ]
    float_imputer = Imputer(
        inputCols=float_cols,
        outputCols=[f"{col}_imp" for col in float_cols]
    ).setStrategy('mean')

    # for binary variables
    binary_cols=[
        'cmp_sex',
        'cmp_bd',
        'cmp_bm',
        'cmp_by',
        'cmp_plz',
    ]
    binary_imputer = Imputer(
        inputCols=binary_cols,
        outputCols=[f"{col}_imp" for col in binary_cols]
    ).setStrategy('mode')
    imputed_df=float_imputer.fit(miss_df).transform(miss_df)
    output_df=binary_imputer.fit(imputed_df).transform(imputed_df)
    output_df=output_df.select([x for x in output_df.columns if '_imp' in x or
x=='is_match'])
    return output_df

def preprocessing_df(input_df):
    output_df = convert_label_binary(fill_missing_values(input_df))
    return output_df.drop('is_match')
prep_df=preprocessing_df(df)

```

نتیجه اجرای کد فوق به صورت زیر است:

```
prep_df.show(10)
```



cmp_fname_c1_imp	cmp_fname_c2_imp	cmp_lname_c1_imp	cmp_lname_c2_imp	cmp_sex_imp	cmp_bd_imp	cmp_bm_imp	cmp_by_imp	cmp_plz_imp	label
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	0	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1
1.0	0.9000177	1.0	0.31841284	1	1	1	1	1	1

نمونه داده پس از پر کردن داده‌های از دست رفته

توسط کد زیر چک می‌کنیم که تمام داده‌های از دست رفته پر شده باشد:

```
prep_df.select([count(when(isnull(column), column)).alias(column) for column
in prep_df.columns]).show()
```

cmp_fname_c1_imp	cmp_fname_c2_imp	cmp_lname_c1_imp	cmp_lname_c2_imp	cmp_sex_imp	cmp_bd_imp	cmp_bm_imp	cmp_by_imp	cmp_plz_imp	label
0	0	0	0	0	0	0	0	0	0

همانطور که ملاحظه می‌شود هیچ داده از دست رفته ای برای هیچ کدام از ویژگی‌ها وجود ندارد.

## ۸ مهندسی ویژگی (Feature Engineering)

در این مرحله جهت آماده سازی داده برای ورود به مرحله مدل سازی لازم است تمام ویژگی های ورودی به صورت بردار در قالب یک ویژگی جمع شوند. در واقع پس از این مرحله مجموعه داده دارای دو ستون خواهد بود: ستون اول تمام ویژگی های ورودی که به برداری از ویژگی ها تبدیل شده اند و ستون دوم خروجی یا برچسب هر رکورد. کد این مرحله به صورت زیر می باشد:

```
from pyspark.ml.feature import VectorAssembler
def feature_engineering(input_df, feature_list, label_name):
    assembler = VectorAssembler(inputCols=feature_list,
                                outputCol='features')
    assembled_df = assembler.transform(input_df)
    output_df=assembled_df.select('features', label_name)
    return output_df
```

```
input_features=list(set(prepare_df.columns) - set(['label','is_match']))
assembled_df = feature_engineering(prepare_df,input_features,'label')
```

خروجی مجموعه داده پس از انجام مهندسی ویژگی به صورت زیر است:

```
assembled_df.show(10, truncate=False)
```

features	label
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,0.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1
[1.0,1.0,1.0,0.3184128403663635,1.0,1.0,1.0,0.9000176787376404,1.0]	1

## ۹ مدل سازی

در این پروژه چهار الگوریتم زیر برای ساخت مدل یادگیری ماشینی در نظر گرفته شده است:

۱- رگرسیون منطقی (Logistic Regression)

۲- درخت تصمیم (Decision Tree)

۳- جنگل تصادفی (Random Forest)

۴- ماشین بردار پشتیبان خطی (Linear Support Vector Machine)

قالب اصلی آموزش و پیش بینی مدل ها به صورت زیر است:

```
estim=[Estimator](featuresCol='features', labelCol='label')
pipeline = Pipeline(stages=[estim])
model = pipeline.fit(train)
result = model.transform(test)
```

در مراحل قبل، تمام ویژگی های مورد نیاز را در ستون features به صورت بردار تجمیع کرده ایم. همچنین خروجی مورد انتظار پس از تبدیل به باینری در ستون label نگهداری می شود. برای استفاده از حداکثر امکانات pyspark از pipeline استفاده کردیم تا بعداً در صورتی که نیاز به cross\_validation یا پیش پردازش های خاصی بود، به صورت pipeline انجام شود. برای آموزش از متد fit و برای پیش بینی روی داده های آزمون از متد transform استفاده کردیم.

### ۹-۱ روش رگرسیون منطقی (Logistic Regression)

در این روش که برای طبقه بندی باینری به کار می رود، به دنبال پیش بینی طبقه براساس متغیرهای وابسته آن است. مفروض اصلی آن، خطی بودن متغیرهای مستقل است. مزیت این روش، استفاده ساده آن است و عملکرد بالایی که برای داده های تفکیک پذیر خطی دارد. برای پیاده سازی با pyspark از کد زیر استفاده کردیم:

```
lr=LogisticRegression(featuresCol='features', labelCol='label')
pipeline = Pipeline(stages=[lr])
model = pipeline.fit(train)
lr_result = model.transform(test)
```

### ۹-۲ روش درخت تصمیم (Decision Tree)

درخت تصمیم نیز مانند رگرسیون منطقی برای داده های با تفکیک پذیر خطی مناسب است. یکی از مفروضات اصلی درخت تصمیم، در نظر گرفتن ویژگی های طبقه ای است و در صورتی که همه یا بعضی از ویژگی ها پیوسته باشند به طبقه ای تبدیل می کند. ایده این روش، تولید دستورالعمل های شرطی برای تقسیم داده ها براساس خلوص است تا به جایی برسیم که خلوص برگ ها بیشینه شود؛ یعنی هر برگ این درخت نماینده یک طبقه باشد. کد زیر برای پیاده سازی درخت تصمیم به کار رفته است:

```
tree = DecisionTreeClassifier()
tree_pipeline = Pipeline(stages=[tree])
```

```
tree_model = tree_pipeline.fit(train)
tree_result = tree_model.transform(test)
```

### ۹-۳ روش جنگل تصادفی (Random Forest)

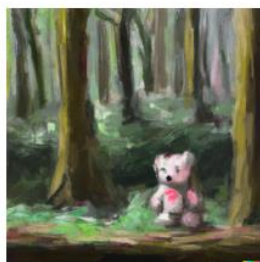
در این روش، از چندین درخت تصمیم استفاده کرده، جواب آن‌ها را با هم تجمیع کردیم. در این روش برخلاف روش‌های قبلی هیچ مفروضی بر روی داده نداریم. اهمیت و عملکرد این روش از دیرباز قابل تحسین بوده و دسته‌بندی آن به عنوان روش آماری یا هوشمند محل بحث است؛ به طوری که در دوره آموزش یادگیری عمیق آقای جرمی هاوارد، روش جنگل تصادفی نیز تدریس می‌شود.

Practical Deep Learning

Lessons

- 1: Getting started
- 2: Deployment
- 3: Neural net foundations
- 4: Natural Language (NLP)
- 5: From-scratch model
- 6: Random forests
- 7: Collaborative filtering
- 8: Convolutions (CNNs)
- 9: Data ethics
- Summaries

## 6: Random forests



Random forests started a revolution in machine learning 20 years ago. For the first time, there was a fast and reliable algorithm which made almost no assumptions about the form of the data, and required almost no preprocessing. In today's lesson, you'll learn how a random forest really works, and how to build one from scratch. And, just as importantly, you'll learn how to interpret random forests to better understand your data.

کد پیاده‌سازی جنگل تصادفی به صورت زیر است:

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="label",
                           featuresCol="features")
rf_pipeline = Pipeline(stages=[ rf])
rf_model=rf_pipeline.fit(train)
rf_result=rf_model.transform(test)
```

### ۹-۴ روش ماشین بردار پشتیبان خطی (Linear SVC)

در این روش، به دنبال تفکیک داده‌های هر کلاس با خط یا ابرصفحه هستیم. یعنی در نهایت، معادله خط یا ابرصفحه بدست می‌آید و فاصله برداری نقاط نسبت به آن‌ها محاسبه می‌شود. با کد زیر می‌توانیم از پیاده‌سازی LinearSVC در pyspark استفاده کنیم:

```
from pyspark.ml.classification import LinearSVC
svc = LinearSVC(labelCol="label", featuresCol="features")
svc_pipeline = Pipeline(stages=[ svc])
svc_model=svc_pipeline.fit(train)
svc_result=svc_model.transform(test)
```

```
evaluate(svc_result,  
        model_name='SVC',  
        rawPredictionCol='rawPrediction',  
        require_auc=False)
```

## ۱۰ ارزیابی

اولین قدم در ارزیابی یک مدل تعیین سنجه مناسب بر اساس اهمیت داده های FP و FN است. اگر شرایط مسأله طوری باشد که تشخیص درست داده های کلاس مثبت اهمیت بیشتری نسبت به داده های کلاس منفی داشته باشد معنی آن این است که ارزیابی مدل باید بر اساس کم شدن مقدار FP انجام شود که با توجه به نحوه محاسبه سنجه های مختلف، سنجه دقت (precision) اهمیت بیشتری پیدا می کند زیرا مقدار FP در کسر قرار دارد که کمتر شدن مقدار آن باعث بیشتر شدن مقدار این سنجه می شود. مشابه همین تحلیل برای FN وجود دارد که منجر به اهمیت سنجه بازخوانی (Recall) می شود.

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

اینکه در یک مجموعه داده کدام یک از مقادیر FP یا FN اهمیت بیشتری دارد بستگی به شرایط و نوع مجموعه داده دارد. در مورد مجموعه داده این مسأله FP یعنی داده هایی که توسط مدل به اشتباه در کلاس مثبت قرار گرفته اند و FN یعنی داده هایی که توسط مدل به اشتباه در کلاس منفی قرار گرفته اند. داده FP یعنی دو شخص متفاوت به عنوان یک شخص توسط مدل شناسایی شده و برچسب match خورده است و داده FN یک شخص است که توسط مدل به عنوان دو شخص متفاوت شناسایی شده و برچسب non-match گرفته است. در مورد FP از دست دادن داده (data loss) اتفاق می افتد و در مورد FN تکرار داده (data duplication) اتفاق می افتد. واضح است که از دست دادن داده بحرانی تر است و باید سعی کنیم حتی الامکان از وقوع این اتفاق جلوگیری کرده و FP را تا جای ممکن کاهش دهیم. لذا با توجه به مطالب فوق سنجه اصلی برای ارزیابی مدل باید دقت (precision) باشد.

اولین چالش در ارزیابی مدل، نامتوازن بودن مجموعه داده است. برای مواجهه به داده های نامتوازن روش های متداول زیر توصیه می شود

۱- استفاده از سنجه هایی مثل AUC score, Macro F1, Weighted F1

۲- نمونه گیری متوازن و استفاده از متریک های معمولی مثل accuracy

۳- وزن دهی متناسب به loss که در هنگام ساخت مدل وزن دهی را تعریف کنیم

### ۱۰-۱ سنجه Macro F1

در حالت کلی سنجه F1 میانگین هارمونیک دو سنجه دقت (precision) و بازخوانی (recall) است. برای رفع ابهام، نام های دیگر این سنجه همراه محاسبه آن آورده شده است:

### ۱۰-۲ سنجه دقت (precision) یا positive predictive value (PPV)

$$TNR = \frac{TP}{TP + FP}$$

### ۱۰-۳ سنجه بازخوانی

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

به زبان ساده، سنجه دقت نشان می‌دهد که از بین پیش‌بینی‌های مدل برای کلاس True، چه نسبتی صحیح پیش‌بینی شده است. سنجه بازخوانی هم نشان می‌دهد از بین تمام داده‌های کلاس True، چه نسبتی درست پیش‌بینی شده است. با توجه به اینکه مقایسه دو سنجه از یک مدل با دو سنجه از یک مدل دیگر کار دشواری است، یک سنجه F1 که میانگین هارمونیک این دو است معرفی شده است که به صورت زیر است:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

تمام این سنجه‌ها در حالت عالی نمایانگر قدرت پیش‌بینی مدل است. اما وقتی داده نامتوازن باشد، باید از حالت خاصی از F1 یعنی macro F1 استفاده کرد. سنجه Macro F1 میانگین غیر وزن‌دار F1 برای تمام کلاس‌ها را اندازه می‌گیرد (N تعداد کلاس‌ها است).

$$F1(\text{macro}) = \frac{\sum f1_i}{N}$$

حالت خاصی از این سنجه به نام Weighted F1 وجود دارد که معیار support هر طبقه را نیز به f1 آن طبقه ضرب کرده و سپس میانگین را حساب می‌کند.

$$f1(\text{weighted}) = \frac{1}{N_L} \sum_{i \in L} (f1_i * \text{support}(i))$$

#### ۱۰-۴ سنجه AUC score

قبل از اینکه به امتیاز AUC بپردازیم باید با منحنی ROC آشنا شویم. منحنی ROC تغییرات نرخ مثبت‌های واقعی به نرخ مثبت‌های کاذب را در طول آستانه‌های مختلف نشان می‌دهد.

$$AUC_{score} = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$$

نمودار ROC براساس تغییرات نرخ‌ها می‌تواند حالت‌های مختلفی داشته باشد. هرچقدر نمودار بالاتر باشد نشان‌دهنده عملکرد بهتر طبقه‌بند است که داده‌های طبقات را بهتر تفکیک کرده است.

برای محاسبه نمودار بهتر از معیار امتیاز AUC یا مساحت زیر منحنی استفاده کردیم تا شهودی که مطرح شد را به صورت معیار کمیتی تبدیل کنیم.

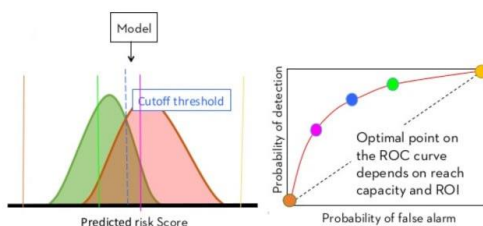


Fig. 19. Showcase of ROC/AUC curves for the given classification model.

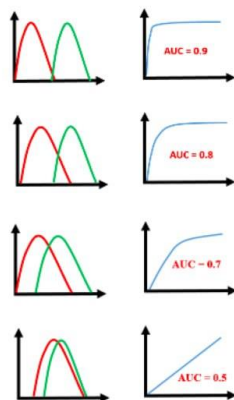
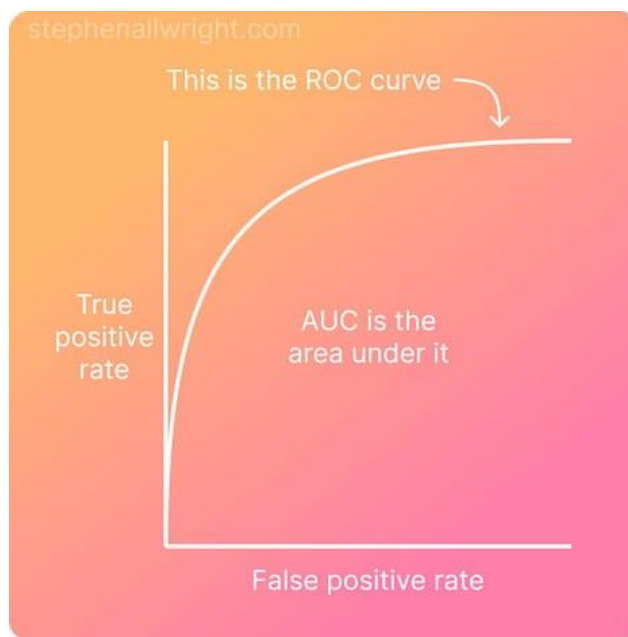


Fig. 20. Better AUC gives the idea of classifier's performance.



برای ارزیابی دو رویکرد مختلف پیاده‌سازی شده است. در رویکرد اول، ابتدا به صورت دستی سنج‌ها را از پیش‌بینی‌ها محاسبه کردیم. تابع `evaluate_from_scratch` برای محاسبه دستی سنج‌های ارزیابی استفاده می‌شود:

```
def evaluate_from_scratch(pred,
                          model_name='Logistic Regression'):
    pred.groupBy('label', 'prediction').count().show()

    # Calculate the elements of the confusion matrix
    TN = pred.filter('prediction = 0 AND label = prediction').count()
    TP = pred.filter('prediction = 1 AND label = prediction').count()
    FN = pred.filter('prediction = 0 AND label = 1').count()
    FP = pred.filter('prediction = 1 AND label = 0').count()

    # Accuracy measures the proportion of correct predictions
    accuracy = (TN + TP) / (TN + TP + FN + FP)
    recall = (TP) / (TP+FN)
    precision= (TP) / (TP+FP)
    f1=2*(precision*recall)/(precision+recall)
    print(f'EVALUATION SUMMARY for {model_name}:')
    print(f" accuracy:{accuracy}")
```



```
print(f" precision:{precision}")
print(f" recall:{recall}")
print(f" f1-score:{f1}")
```

با استفاده کد بالا، سنجه‌های مورد نیاز را به صورت دستی با استفاده از ماتریس آشفستگی بدست می‌آوریم. برای رویکرد دوم، از متدهای pyspark استفاده کردیم. تابع `evaluate_from_spark`، تمام سنجه‌ها را با استفاده از متدهای pyspark محاسبه می‌کند و منحنی ROC را نیز نمایش می‌دهد.

```
def evaluate_from_spark(predictions,
                        model_name='Logistic Regression',
                        rawPredictionCol="probability",
                        labelCol="label",
                        predictionCol="prediction",
                        require_auc=True
                        ):
    print(rawPredictionCol)

    eval2= MulticlassClassificationEvaluator(predictionCol=predictionCol,
                                             labelCol=labelCol)

    ACC = eval2.evaluate(predictions,
                          {eval2.metricName:"accuracy"})

    PREC = eval2.evaluate(predictions,
                           {eval2.metricName:"weightedPrecision"})

    REC = eval2.evaluate(predictions,
                           {eval2.metricName:"weightedRecall"})

    F1 = eval2.evaluate(predictions,
                         {eval2.metricName:"f1"})

    WeightedFMeasure=eval2.evaluate(predictions,
```

```

        {eval2.metricName:"weightedFMeasure"})

print(f"{model_name} Performance Measure")

print(" Accuracy = %0.8f" % ACC)

print(" Weighted Precision = %0.8f" % PREC)

print(" Weighted Recall = %0.8f" % REC)

print(" F1 = %0.8f" % F1)

print(" Weighted F Measure = %0.8f" % WeightedFMeasure)

if require_auc:

    eval = BinaryClassificationEvaluator(rawPredictionCol=rawPredictionCol,

                                       labelCol=labelCol)

    AUC = eval.evaluate(predictions)

    print(" AUC = %.8f" % AUC)

    print(" ROC curve:")

    PredAndLabels = predictions.select(rawPredictionCol,

                                       labelCol)

    PredAndLabels_collect = PredAndLabels.collect()

    PredAndLabels_list = [(float(i[0][0]), 1.0-float(i[1])) for i in
PredAndLabels_collect]

    PredAndLabels = sc.parallelize(PredAndLabels_list)

    fpr = dict()

# FPR: False Positive Rate

    tpr = dict()

# TPR: True Positive Rate

    roc_auc = dict()

    y_test = [i[1] for i in PredAndLabels_list]

```

```

y_score = [i[0] for i in PredAndLabels_list]

fpr, tpr, _ = roc_curve(y_test, y_score)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(5,4))

plt.plot(fpr, tpr, label='ROC curve (area = %0.8f)' % roc_auc)

plt.plot([0, 1], [0, 1], 'r--')

# plt.xlim([0.0, 1.0])

# plt.ylim([0.0, 1.05])

plt.yticks(np.arange(0,1.03,0.1))

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(f'ROC Curve - {model_name}')

plt.legend(loc="lower right")

plt.show()

```

برای تجميع توابع بالا از تابع evaluate استفاده كرديم تا خروجي به صورت يکپارچه چاپ شود.

```

def evaluate(predictions,

              model_name=None,

              rawPredictionCol="probability",

              labelCol="label",

              predictionCol="prediction",

              require_auc=True

              ):

    print('Evaluate From Scratch:')

    evaluate_from_scratch(predictions,model_name)

```

```
print('\nEvaluate From Spark Library:')

evaluate_from_spark(predictions,

                    model_name,

                    rawPredictionCol=rawPredictionCol,

                    labelCol=labelCol,

                    predictionCol=predictionCol,

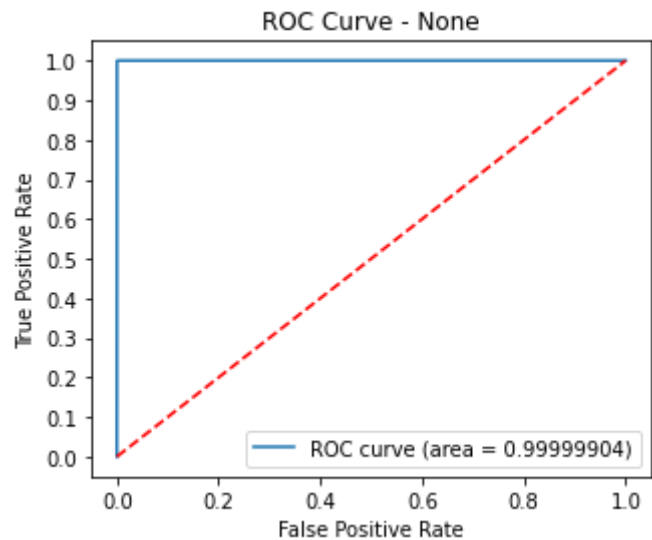
                    require_auc=require_auc)
```

نمونه خروجی ارزیابی یک مدل به صورت زیر است:

```
Evaluate From Scratch:
+-----+-----+-----+
|label|prediction|count|
+-----+-----+-----+
|1|0.0|27|
|0|0.0|1717224|
|1|1.0|6258|
|0|1.0|106|
+-----+-----+-----+

EVALUATION SUMMARY for Decision Tree:
accuracy:0.9999228365963396
precision:0.9833438089252042
recall:0.9957040572792363
f1-score:0.9894853348090757

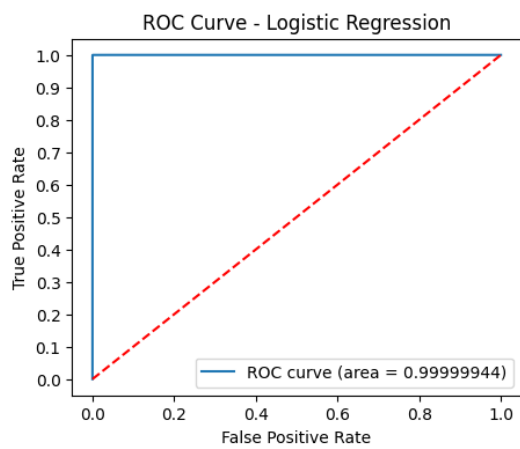
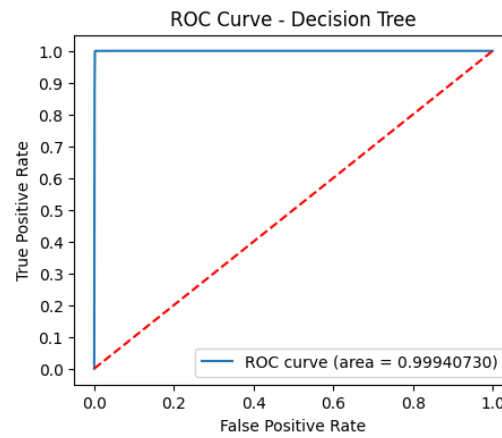
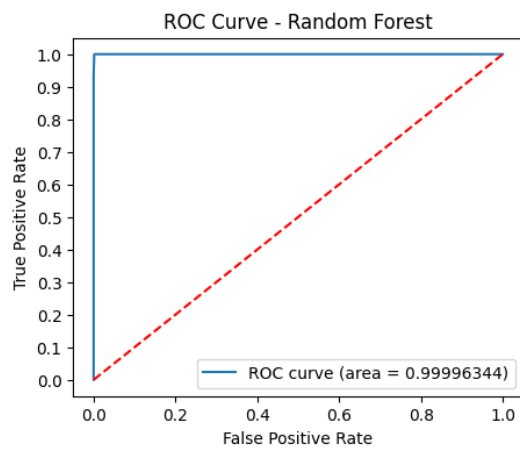
Evaluate From Spark Library:
Decision Tree Performance Measure
Accuracy = 0.99992284
Weighted Precision = 0.99992360
Weighted Recall = 0.99992284
F1 = 0.99992308
Weighted F Measure = 0.99992308
AUC = 0.99940730
ROC curve:
```



در جدول زیر سنجه‌های ارزیابی برای مدل‌های مختلف آورده شده است:

خطی SVC	جنگل تصادفی	درخت تصمیم	رگرسیون منطقی	سنجه
۰.۹۹۹۸۵۳۲۲	۰.۹۹۹۸۲۳۰۵	۰.۹۹۹۹۲۲۸۴	۰.۹۹۹۹۸۳۷۶	Accuracy
۰.۹۹۸۴۸۸۴۱	۰.۹۹۹۸۳۲۸۳	۰.۹۸۳۳۴۳۸۱	۰.۹۹۸۴۰۶۸۸	Precision
۰.۹۶۰۵۷۵۲۱	۰.۹۵۱۶۳۰۸۷	۰.۹۹۵۷۰۴۰۶	۰.۹۹۷۱۳۶۰۴	Recall
۰.۹۹۹۸۵۱۸۰	۰.۹۹۹۸۲۰۸۷	۰.۹۹۹۹۲۳۰۸	۰.۹۹۹۹۸۳۷۵	Weighted F1
	۰.۹۹۹۹۶۳۴۴	۰.۹۹۹۴۰۷۳۰	۰.۹۹۹۹۹۹۴۳	AUC score

همچنین منحنی ROC به تفکیک مدل‌ها را نیز به صورت زیر رسم شده است:



## ۱۱ ارزیابی متقابل (Cross Validation)

برای بهبود پارامترهای مدل و انتخاب بهترین پارامتر، از روش ارزیابی cross validation استفاده کردیم؛ یعنی یک قسمتی از داده آموزش را به عنوان مجموعه داده ارزیابی (Validation set) جدا کرده و هربار با تغییر پارامترها، سنجه مورد نظر را بر روی مجموعه ارزیابی سنجیدیم. سپس پارامترهایی که بالاترین مقدار سنجه را بدست آوردند انتخاب کردیم. در این قسمت، با مجموعه داده آزمون کاری نداریم و به اصطلاح مدل نباید داده‌های آزمون را ببیند.

برای بهبود پارامترها از روش GridSearch و برای مرحله ساخت مدل، از pipeline استفاده کردیم. به عنوان نمونه برای روش رگرسیون منطقی، کد بهبود پارامتر به صورت زیر است:

```
model_cs = current_model()
pipeline = Pipeline(stages=[model_cs])
grid = ParamGridBuilder().addGrid(model_cs.regParam, [0.0,0.1]) \
    .addGrid(model_cs.elasticNetParam, [0.0, 1.0])\
    .build()
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(estimator=pipeline,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)
cvModel = cv.fit(train)
```

همانطور که در کد بالا مشاهده می‌شود، برای رگرسیون منطقی دو پارامتر regParam و elasticNetParam با مقادیر مختلف تعریف شده است. وظیفه CrossValidator، آزمایش تمام حالت‌های مختلف این پارامترها در کنار یکدیگر و ارزیابی آن‌هاست. آرگومان numFolds به معنای این است که هر بار مجموعه آموزش را به ۳ قسمت تقسیم و یک قسمت آن را برای ارزیابی در نظر می‌گیرد. با توجه به منابع محدود local، هنگام پردازش این قسمت از کد، با خطای حافظه مواجه شدیم.

```

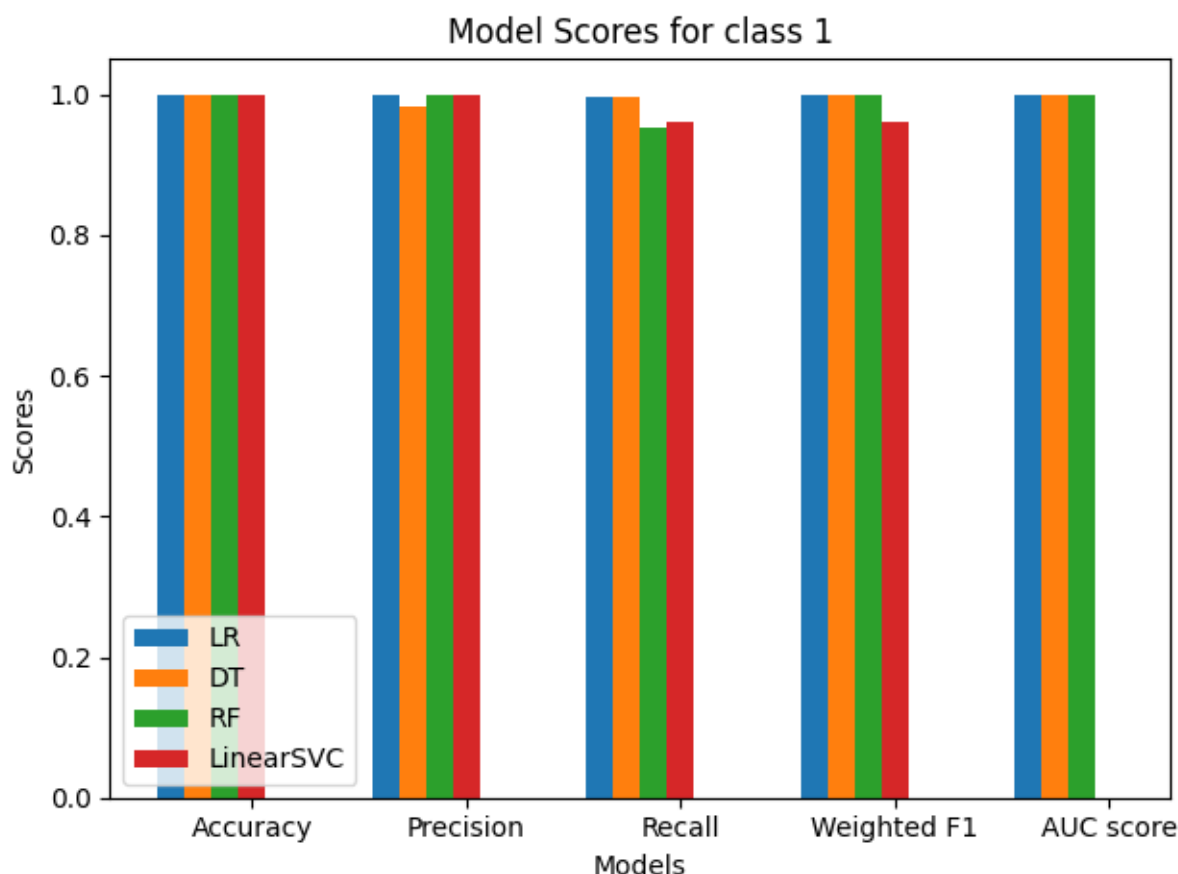
java.lang.OutOfMemoryError: Java heap space
23/02/11 12:44:09 ERROR SparkUncaughtExceptionHandler: Uncaught exception in thread Thread[Executor task launch worker for task 3.0 in stage 614.0
ID 7111],5,main]
java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.reflect.Array.newInstance(Array.java:78)
    at java.base/java.io.ObjectInputStream.readArray(ObjectInputStream.java:2132)
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1732)
    at java.base/java.io.ObjectInputStream.readArray(ObjectInputStream.java:2168)
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1732)
    at java.base/java.io.ObjectInputStream$FieldValues.<init>(ObjectInputStream.java:2617)
    at java.base/java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2468)
    at java.base/java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2268)
    at java.base/java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1744)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:514)
    at java.base/java.io.ObjectInputStream.readObject(ObjectInputStream.java:472)
    at org.apache.spark.serializer.JavaDeserializationStream.readObject(JavaSerializer.scala:87)
    at org.apache.spark.serializer.DeserializationStream$$anon$1.getNext(Serializer.scala:168)
    at org.apache.spark.util.NextIterator.hasNext(NextIterator.scala:73)
    at org.apache.spark.storage.memory.MemoryStore.putIterator(MemoryStore.scala:223)
    at org.apache.spark.storage.memory.MemoryStore.putIteratorAsValues(MemoryStore.scala:302)
    at org.apache.spark.storage.BlockManager.maybeCacheDiskValuesInMemory(BlockManager.scala:1664)
    at org.apache.spark.storage.BlockManager.getLocalValues(BlockManager.scala:953)
    at org.apache.spark.storage.BlockManager.get(BlockManager.scala:1258)
    at org.apache.spark.storage.BlockManager.getOrElseUpdate(BlockManager.scala:1325)
    at org.apache.spark.rdd.RDD.getOrCompute(RDD.scala:376)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:327)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:365)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:329)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:365)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:329)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:365)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:329)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)

```

کد این بخش در فایل cross\_valiation\_test.py موجود است.

## ۱۲ مقایسه عملکرد مدل‌ها

برای نشان دادن تفاوت عملکرد مدل‌ها به صورت ملموس‌تر، نمودار میله‌ای عملکرد آن‌ها را به صورت زیر رسم نمودیم. در این نمودار برای هر سنج عملکرد روش‌های رگرسیون منطقی (LR)، درخت تصمیم (DT)، جنگل تصادفی (RF) و ماشین بردار پشتیبان خطی (Linear SVC) با یکدیگر مقایسه شده‌اند.



مشاهده می‌شود که مدل رگرسیون منطقی نسبت به مدل‌های دیگر با اختلاف کمتری از صحت و  $f1$  بالاتری برخوردار است. همچنین امتیاز AUC آن نیز بیشتر بوده که به معنی طبقه‌بندی قابل اعتمادتر است. همچنین در سنج بازخوانی (recall) مدل‌های رگرسیون منطقی و درخت تصمیم اختلاف قابل توجهی با دیگر مدل‌ها دارند. عمده اختلاف بین مدل‌ها در سنج Recall است که باعث تأثیرگذاری در  $f1$  نیز می‌شود. البته دقت درخت تصمیم نسبت به دیگر مدل‌ها با اختلاف ناچیزی کمتر است.

با توجه به عملکرد مشابه مدل‌ها، دو راهکار برای استقرار مدل یادگیری ماشینی برای پروژه پیشنهاد می‌شود:

- ۱- در صورتی که نیاز است یادگیری به صورت مجموعه‌ای از قوانین پیاده شده و خاصیت بصری نیز داشته باشد، از درخت تصمیم استفاده شود.

- ۲- در صورتی که کمینه بودن FP اهمیت بیشتری دارد، توصیه می‌شود از مدل‌های دیگر که دقت بالاتری دارند استفاده شود. به عنوان مثال، جنگل تصادفی می‌تواند به عنوان مدلی استفاده شود که از هیچ مفروضی استفاده نمی‌کند که برای محیط واقعی ایده‌آل است.