به نام خدا

دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

تکلیف سوم - تحلیل کلان داده ها

استادمربوطه: دکتر حقیر چهرقانی

نام: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

تابستان ۱۴۰۲

فهرست:

بخش اول Data Stream

۱-۱

الگوریتم Random Forest برای جلوگیری از overfit مدل با استفاده از bagging و انتخاب تصادفی ویژگی ها توسعه می یابد. برای استفاده از این الگوریتم در جریان داده باید به موارد زیر نیاز است.

۱- ایجاد تنوع برای re-sampling

۲- ایجاد تنوع در انتخاب نمونه ای از ویژگی ها برای تقسیم در گره ها

۳- آشکارسازی تغییر توزیع داده در هر درخت پایه و آموزش درخت‌های پس زمینه که در صورت شناسایی رانش شروع به آموزش می‌کنند و در صورت بیشتر شدن آنها نسبت به آستانه، درخت جدید را جایگزین میکند.

نیاز دوم با اصلاح الگوریتم القای درخت پایه، به طور موثر با محدود کردن مجموعه ویژگی‌های در نظر گرفته شده برای تقسیم‌های بیشتر به یک زیرمجموعه تصادفی با اندازه m، که $m < M$ و(M با تعداد کل ویژگی‌ها مطابقت دارد) به دست می‌آید.

برای اولین نیاز، در بسته بندی غیر جریانی هر یک از n مدل پایه در یک نمونه اندازه Z که با نمونه های تصادفی با جایگزینی از مجموعه آموزشی ایجاد می شود و آموزش داده می شود. برای مقادیر بزرگ Z، این توزیع دو جمله ای به توزیع پواسون ($\lambda = 1$) پایبند است. برای استفاده از bagging در جریان داده، که نمونه‌گیری تصادفی با جایگزینی نمونه‌های وزن‌دهی بر اساس توزیع پواسون ($\lambda = 1$) تقریبی می‌کند. در این الگوریتم به صورت دیفالت، به جای پواسون ($\lambda = 1$) از پواسون ($\lambda = 6$) استفاده شده است.

استراتژی‌های دیگری برای مقابله با رانش‌ها لحاظ شود. به طور مشخص، این استراتژی‌ها شامل روش‌های تشخیص رانش/هشدار، رای‌گیری وزنی و آموزش درختان در پس‌زمینه قبل از جایگزینی درختان موجود است.

در این الگوریتم برای کنترل تغییر توزیع داده ها، پس از تشخیص رانش، درخت پس زمینه ایجاد میشود و پس از افزایش رانش از حد آستانه درخت پس زمینه جایگزین میشود.

در ARF، آرا بر اساس دقت تست درختان به صورت وزن دار انجام میشود، به عنوان مثال، با فرض اینکه درخت l از آخرین بازنشانی خود nl نمونه را دیده و نمونه های cl را به درستی طبقه بندی کرده است، به طوری که $cl \leq nl$، سپس وزن آن cl/nl خواهد بود.

این الگوریتم چند تفاوت با درخت Hoeffding دارد ، شامل هرس اولیه درختان نمی شود. دوم، هر زمان که یک گره جدید ایجاد می شود یک زیر مجموعه تصادفی از ویژگی ها با اندازه m انتخاب می شود و تقسیم به این ویژگی ها برای گره داده شده محدود می شود.

۲-۱

mnist.data

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783	pixel784
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
69996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
69997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
69998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
69999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

70000 rows × 784 columns

۳-۱

```
from skmultiflow.data import DataStream
import numpy as np
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)


stream = DataStream(X, np.array(y))
```

۴-۱

بر روی ۲۰۰ نونه به ازای پارامتر های مختلف الگوریتم اجرا شده است و پارامتر های بهترین دقت به عنوان جواب بهینه در نظر گرفته شده است. تعریف پارامتر ها به صورت زیر میباشد:

```
max_samples = 200

n_estimators_values = [10, 50, 100]
grace_period_values = [50,100,150]
max_features_values = [0.1, 0.3, 0.5]
```

میزان دقت به ازای پارامتر های مختلف به صورت زیر است:

```
n_estimators=10 max_feature=0.1 grace_period=50  Accuracy: 0.6
n_estimators=10 max_feature=0.1 grace_period=100  Accuracy: 0.605
n_estimators=10 max_feature=0.1 grace_period=150  Accuracy: 0.52
n_estimators=10 max_feature=0.3 grace_period=50  Accuracy: 0.48
n_estimators=10 max_feature=0.3 grace_period=100  Accuracy: 0.42
n_estimators=10 max_feature=0.3 grace_period=150  Accuracy: 0.405
n_estimators=10 max_feature=0.5 grace_period=50  Accuracy: 0.12
n_estimators=10 max_feature=0.5 grace_period=100  Accuracy: 0.11
n_estimators=10 max_feature=0.5 grace_period=150  Accuracy: 0.195
n_estimators=50 max_feature=0.1 grace_period=50  Accuracy: 0.605
n_estimators=50 max_feature=0.1 grace_period=100  Accuracy: 0.625
n_estimators=50 max_feature=0.1 grace_period=150  Accuracy: 0.54
n_estimators=50 max_feature=0.3 grace_period=50  Accuracy: 0.5
n_estimators=50 max_feature=0.3 grace_period=100  Accuracy: 0.525
n_estimators=50 max_feature=0.3 grace_period=150  Accuracy: 0.515
n_estimators=50 max_feature=0.5 grace_period=50  Accuracy: 0.17
n_estimators=50 max_feature=0.5 grace_period=100  Accuracy: 0.14
n_estimators=50 max_feature=0.5 grace_period=150  Accuracy: 0.16
n_estimators=100 max_feature=0.1 grace_period=50  Accuracy: 0.645
n_estimators=100 max_feature=0.1 grace_period=100  Accuracy: 0.6
n_estimators=100 max_feature=0.1 grace_period=150  Accuracy: 0.55
n_estimators=100 max_feature=0.3 grace_period=50  Accuracy: 0.53
n_estimators=100 max_feature=0.3 grace_period=100  Accuracy: 0.525
n_estimators=100 max_feature=0.3 grace_period=150  Accuracy: 0.52
```

بالاترین دقت به دست آمده 0.645 میباشد.

۵-۱

اجرا روی پارامتر های قسمت قبل با ۶ ساعت اجرا روی ۵ هزار نمونه پایان نیافت، بنابراین پارامتر های زیر انتخاب شدند، پارامتر های انتخاب شده نزدیک به جواب بهینه بعدی یعنی ۰.۶۲۵ هستند:

```
arf=AdaptiveRandomForestClassifier(n_estimators=50,max_features='auto',
grace_period=100)
```

کد اجرا شده روی ۵ هزار نمونه به صورت زیر میباشد، داده ها به صورت incremental با استفاده از تابع next_sample خوانده میشود و با استفاده از تابع partial_fit درخت به صورت incremental آموزش داده میشود.

```
arf = AdaptiveRandomForestClassifier(n_estimators=50, max_features='auto',grace_period=100)

n_samples = 0
correct_cnt = 0
max_samples = 5000

stream = DataStream(X, np.array(y))

# Train the estimator with the samples provided by the data stream
while n_samples < max_samples and stream.has_more_samples():

    X, y = stream.next_sample()
    y_pred = arf.predict(X)

    if y[0] == y_pred[0]:
        correct_cnt += 1

    arf.partial_fit(X, y)
    n_samples += 1


    acc.append(correct_cnt / n_samples)
```

برای اینکه گفته شده دقت روی صد داده اخیر، پنجره ای به طول ۱۰۰ در نظر گرفته شده است.

دقت کلی به صورت زیر میباشد:

```
Adaptive Random Forest ensemble classifier example
5000 samples analyzed.
Accuracy: 0.6712
```

ماکزیمم دقت به دست آمده روی پنجره و آخرین آن به صورت زیر است:

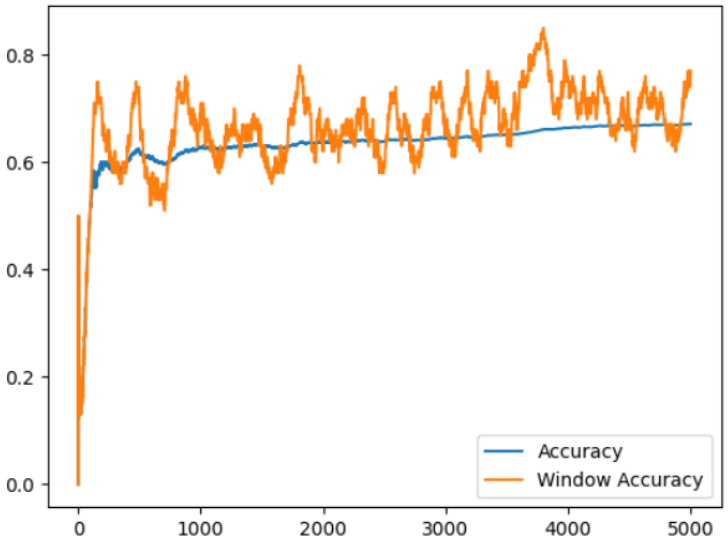
```
sum(window)/len(window)
```

```
0.77
```

```
max(window_acc)
```

```
0.85
```

نمودار صحت روی کل مجموعه داده و ۱۰۰ داده اخیر:



در نمودار بالا دقت روی ۱۰۰ داده اخیر دارای نوسان می باشد، ولی دقت کلی منحنی دارای نوسان نیست.

بخش دوم Singular Value Decomposition

۲-۱ الف

برای نشان دادن ارتباط از دو راه حل استفاده شده که هر دو نتایج یکسانی دارند.

میدانیم که عناصر ماتریس X نمونه های تصادفی از توزیع گوسی با میانگین صفر و کواریانس ماتریس همانی هستند.

راه حل اول:

Y=MX پس روابط زیر برقرار است:

mean_Y = M * mean_X

Cov_Y = M * Cov_X * M^T

با توجه به اینکه Cov_X = I و ضرب هر ماتریس در ماتریس همانی برابر با خود آن ماتریس است، داریم:

$$Covariance(Y) = M * M^T$$

$$U_c S_c V_c^t = U_m S_m V_m^t V_m S_m U_c^t$$

می دانیم $I = V_m^t V_m$ بنابراین روابط زیر برقرار است:

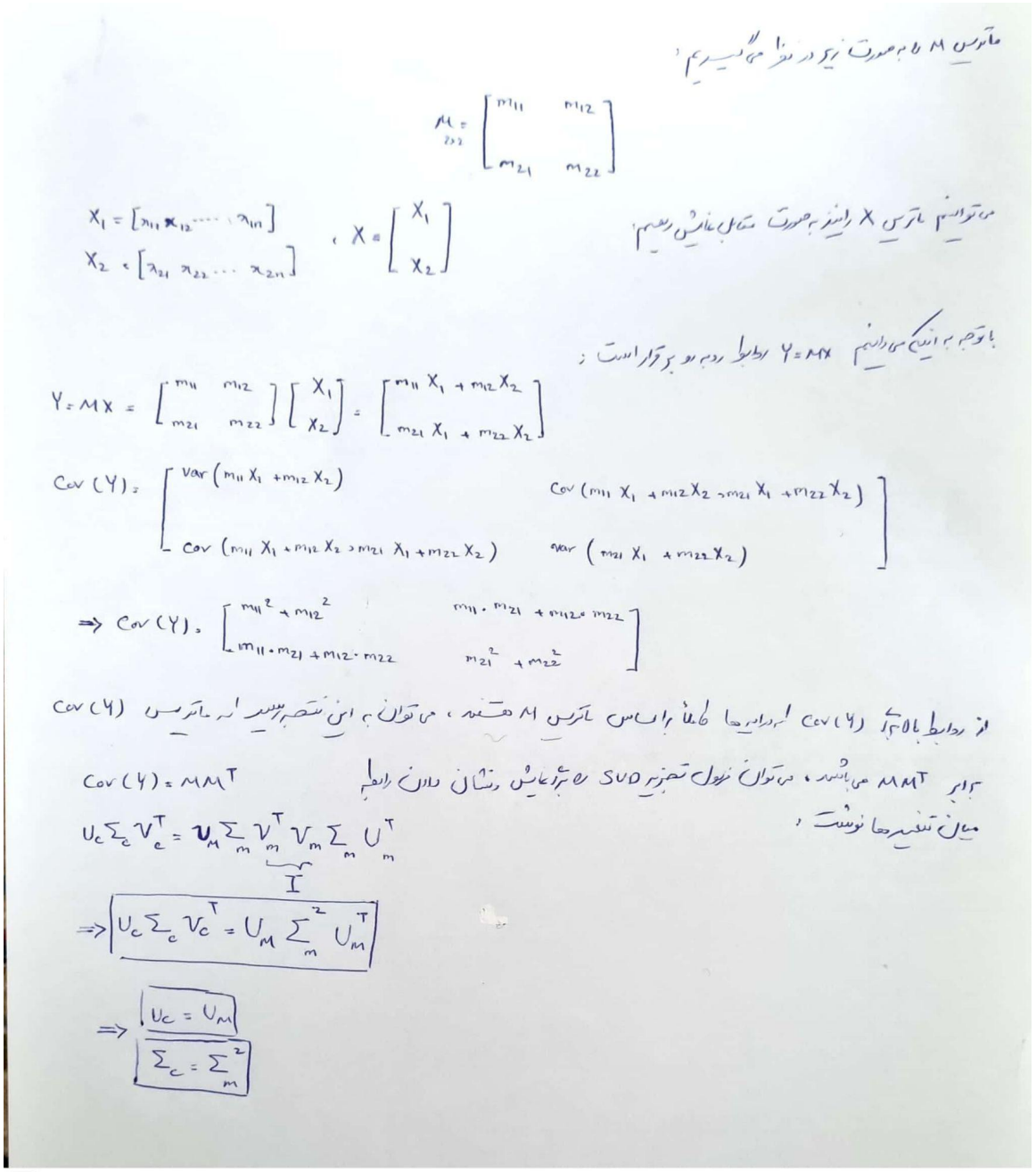
$$U_c S_c V_c^t = U_m S_m^2 U_c^t$$

در نتیجه روابط زیر برقرار است:

$$S_c = S_m^2$$

$$U_c = U_m$$

راه حل دوم:



هر دو حالت نتایج یکسانی دارند

۲-۲ ب

راه حل اول: اثبات با استفاده از روابط ریاضی

برای حل این قسمت، دو طرف تساوی را در Transpose آن ضرب می‌کنیم، داریم:

$$YY^T = MX(MX)^T = MXX^TM$$

ابتدا، XX^T را بررسی می‌کنیم. برای محاسبه‌ی XX^T از امید ریاضی استفاده می‌کنیم.

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \quad , \quad X_1 = [x_{11}\ x_{12} \cdots x_{1n}] \quad X_2 = [x_{21}\ x_{22} \cdots x_{2n}]$$

$$XX^T = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix}_{2\times n} [X_1 \quad X_2] = \begin{bmatrix} \sum_{i=1}^{n} x_{1i}^2 & \sum_{i=1}^{n} x_{1i}\, x_{2i} \\ \sum_{i=1}^{n} x_{1i} \cdot x_{2i} & \sum_{i=1}^{n} x_{2i}^2 \end{bmatrix}$$

$$E(XX^T) = \begin{bmatrix} \sum_{i=1}^{n} E(x_{1i})^2 & \sum_{i=1}^{n} E(x_{1i} \cdot x_{2i}) \\ \sum_{i=1}^{n} E(x_{1i} \cdot x_{2i}) & \sum_{i=1}^{n} E(x_{2i}^2) \end{bmatrix}$$

$$Var(x) = E(x^2) - \underbrace{E(x)^2}_{\text{میانگین داده‌ها برابر با صفر است}} = E(x^2) \qquad Cov(x,y) = E(xy) - E(x)E(y) = E(xy)$$

با استفاده از روابط فوق، می‌توانیم $E(XX^T)$ را به صورت زیر پیاده‌سازی کنیم:

$$E(XX^T) = \begin{bmatrix} \sum_{i=1}^{n} Var(x_{1i}) & \sum_{i=1}^{n} cov(x_{1i}, x_{2i}) \\ \sum_{i=1}^{n} cov(x_{1i}, x_{2i}) & \sum_{i=1}^{n} var(x_{2i}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} 0 \\ \sum_{i=1}^{n} 0 & \sum_{i=1}^{n} 1 \end{bmatrix}$$

$$= n \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = nI$$

با توجه به روابط بالا $E(XX^T)$ را می‌توان به صورت n برابر $cov(X)$ در نظر گرفت و چون $cov(X)$ ماتریس همانی است روابط روبه‌رو برقرار است.

پس داریم:

$$YY^T = MnIM^T = nMM^T$$

تجزیه‌ی بردارهای روبه‌رو با SVD داریم:

$$U_Y \Sigma_Y V_Y^T V_Y \Sigma_Y U_Y^T = n \cdot U_m \Sigma_m V_m^T V_m \Sigma_m U_m^T$$

$$\Rightarrow U_Y \Sigma_Y^2 U_Y^T = n \cdot U_m \Sigma_m^2 U_m^T$$

$$\Rightarrow \Sigma_Y^2 = n \Sigma_m^2 \quad \Rightarrow \quad \boxed{\Sigma_Y = \sqrt{n \Sigma_m^2} = \sqrt{n}\, \Sigma_m}$$

راه حل دوم: استفاده از شبیه سازی

```
[2] import numpy as np

    size_list= [10,30,50,100,200,300,400]
    M1 = np.array([[2, 1], [10, 2.5]])
    M2 = np.array([[8, 9], [7.5, 3]])

    _, Sm1, _ = np.linalg.svd(M1, full_matrices=True)
    _, Sm2, _ = np.linalg.svd(M2, full_matrices=True)

    print(f"M1: {Sm1}  M2:{Sm2} \n********************")


    for s in size_list:
       # Generate the matrix
       matrix = np.random.multivariate_normal(mean, covariance, size=s).T
       y1 = M1 @ matrix
       y2 = M2 @ matrix
       U2, S2, Vh2 = np.linalg.svd(y2, full_matrices=True)
       U1, S1, Vh1 = np.linalg.svd(y1, full_matrices=False)

       print(f"size:{s}  y1: {S1}  y2:{S2}")
```

```
M1: [10.53683184  0.47452594]  M2:[14.17137065  3.069569  ]
********************
size:10  y1: [37.00092246  1.02150184]  y2:[48.45974979  6.78562397]
size:30  y1: [51.76370625  3.03460443]  y2:[70.87919139 19.28092876]
size:50  y1: [72.01368566  3.12306921]  y2:[101.9176756   19.19845985]
size:100  y1: [99.34959596  4.25676612]  y2:[126.49659467  29.08615488]
size:200  y1: [136.6462212    7.04317041]  y2:[190.14194682  44.03592656]
size:300  y1: [163.74637693   8.51101872]  y2:[219.91807453  55.13299346]
size:400  y1: [203.15985581   9.53492661]  y2:[277.39691264  60.75372069]
```

نتیجه به دست آمده در راه حل اول، در ارتباط میان ماتریس های بالا نیز برقرار است.

۲-۳ ج

از نتایج به دست آمده در قسمت الف میتوان برای به دست آوردن M استفاده کرد.

با استفاده از کتابخانه numpy کواریانس Y2 و Y1 را محاسبه میکنیم، با استفاده از روابط زیر می توان ماتریس M را پیاده سازی کرد.

Y1 = M * X Y2 = M^T * X

کواریانس Y1 را C1 , کواریانس Y2 را C2 مینامیم.

$$U_{c1} S_{c1} V_{c1}^{t} = U_m S_m V_m^{t} V_m S_m U_c^{t}$$

$$U_{c1} S_{c1} V_{c1}^{t} = U_m S_m^2 U_c^{t}$$

داریم:

$$S_m = \sqrt{S_{c1}} \qquad U_m = U_{c1}$$

تا اینجا U , S در تجزیه SVD ماتریس M را به دست آورده ایم و نیاز به V داریم:

$$C2 = M^T M = V_m S_m U^T{}_m U_m S V^T{}_m$$
$$U_{c2} S_{c2} V_{c2}{}^t = V_m S^2{}_m V_c{}^t$$

روابط روبه رو برقرار است: $V_m = V_{c2}$

در نتیجه ماتریس M را میتوان به صورت رو به رو به دست آورد: $M = U_{c1}\sqrt{S_{c1}} V_{c2}{}^t$

۲-۳ د

در این بخش با استفاده از میانگین و کواریانس داده شده، ماتریس X را با ۵۰۰۰۰ نمونه به صورت زیر پیاده سازی میکنیم

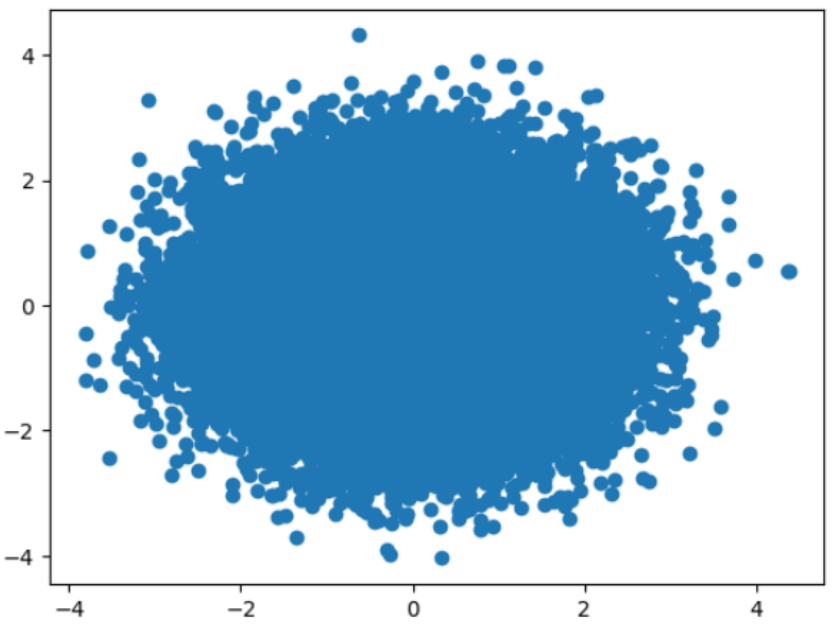
```
X = np.random.multivariate_normal(mean, covariance, size=50000).T
```

ماتریس X به صورت زیرمیباشد:

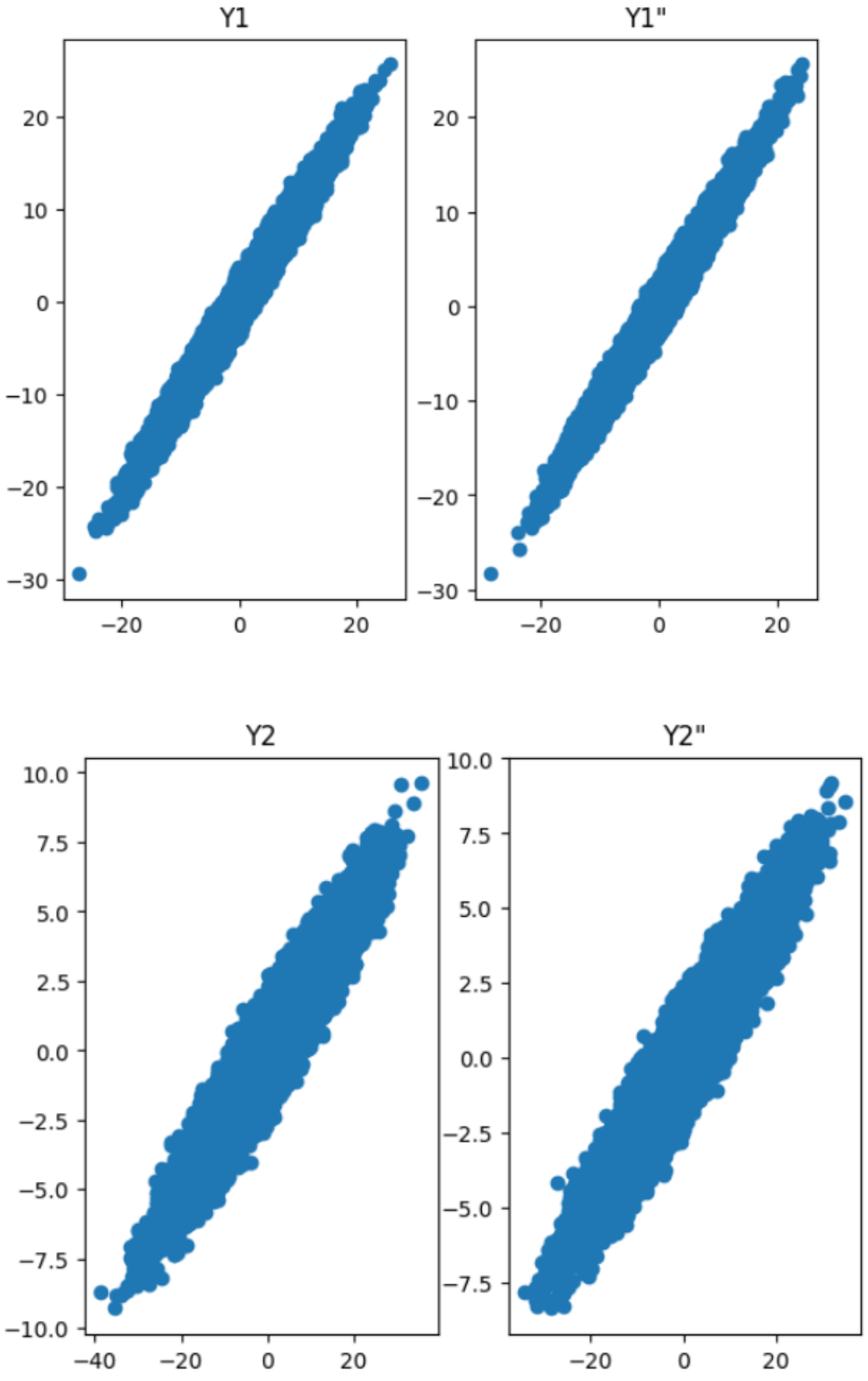
```
X = np.random.multivariate_normal(mean, covariance, size=50000).T
X.shape
```

```
(2, 50000)
```

```
plt.scatter(X[0],X[1])
plt.show()
```



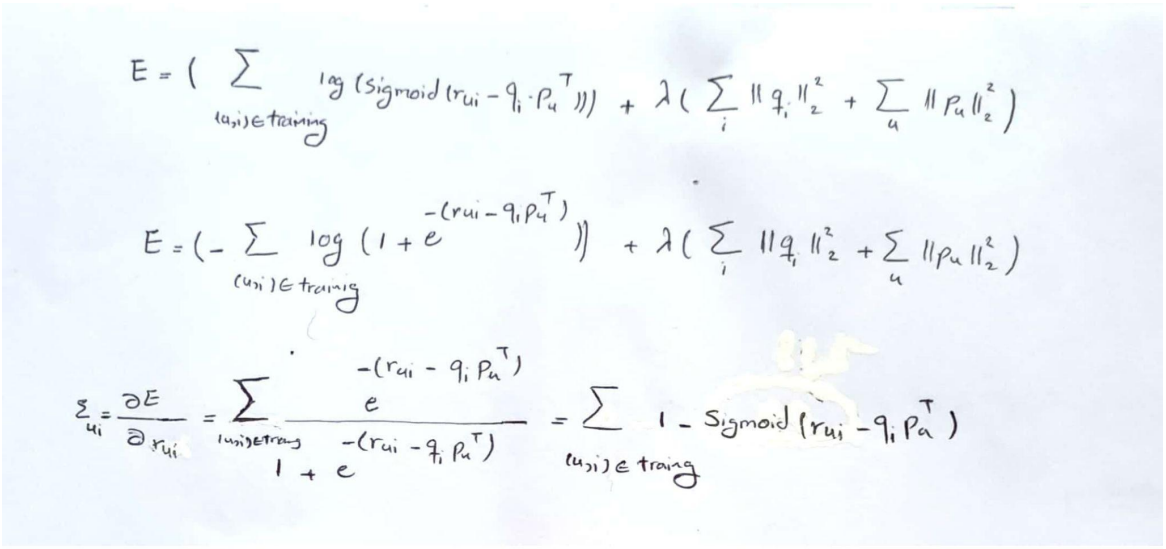
سپس با توجه به روابط بالا ماتریس M را بدست میاوریم و با استفاده از آن و روابط داده شده Y'_1 و Y'_2 را به دست میاوریم و با Y1 , Y2 مقایسه میکنیم:



با توجه به شباهت نمودار ها، میتوان به این نتیجه رسید ماتریس M و روابط به دست آمده درست است.

بخش سوم Recommender System

۳-۱ الف



۳-۲ ب

برای ارزیابی مدل نیاز به دو دسته داده، تست و آموزش است. برای این کار همانطور که از تصویر زیر مشخص است، ۴۰ درصد کاربران به صورت تصادفی انتخاب شده اند و از میان آیتم های مورد علاقه هر کدام از آنها ۲۰ درصد به صورت تصادفی صفر شده است.

```
[33] import random

     R_train = R
     user_test_list = random.sample(range(0, num_users), int(num_users*0.4))
     for i in user_test_list:
        (user_id, item_ids) = data[i]
        item_test_list = random.sample(item_ids, int(len(item_ids)*0.2))
        for item_id in item_test_list:
                R_train[i, item_id] = 0
```

با انجام این کار ماتریس برای آموزش مدل به دست می آید (R_train)و داده های این کاربران برای تست در نظر گرفته میشود(user_test_list).

قسمت ارزیابی برای کاربران مجموعه تست(user_test_list) اجرا میشود، ابتدا برای آنها نمرات آیتم ها پیش بینی می شود و ۱۰ آیتم برتر پیش بینی شده، با داده اصلی مقایسه میشود.

```
    num_users = len(users_list)
    total_precision = 0

    for u in users_list:

        ground_truth = groundTruth_list[u]
        predicted_ratings = np.dot(self.P[u, :], self.Q.T)
        sorted_indices = np.argsort(predicted_ratings)[::-1]
        top_items = sorted_indices[:topk]
        intersection = set(top_items).intersection(ground_truth
        precision = len(intersection) / len(ground_truth)
        total_precision += precision

    average_precision = total_precision / num_users

    return average_precision
```

ماتریس اولیه P , Q به صورت زیر به دست می آید، که با اعدادی میان ۰ تا ۱ پر میشود.

```
# Initialize Q and P matrices with random values
# Start your code
#self.P = np.random.normal(
  #  scale=1./self.num_factors, size=(self.num_users, self.num_factors))
#self.Q = np.random.normal(
 #   scale=1./self.num_factors, size=(self.num_items, self.num_factors))
self.P = np.random.rand(self.num_users, self.num_factors)
self.Q = np.random.rand(self.num_items, self.num_factors)
# End your code
```

برای آموزش مدل در هر تکرار داده ها shuffle میشوند و ماتریس P , Q به صورت زیر آپدیت میشوند:

```
for n in range(self.num_iterations):
    np.random.shuffle(self.samples)
    for u, i, r in self.samples:
      # Computer prediction and error
      prediction = self.predict_rating(i, u)
      sigmoid_grad = 1 - self.sigmoid((r - prediction))
      # Update user and item latent feature matrices
      self.P[u, :] -= self.learning_rate * (((-1)* (sigmoid_grad * self.Q[i, :])) + (2 * (self.regularization_rate * self.P[u, :])))
      self.Q[i, :] -= self.learning_rate * (((-1)* (sigmoid_grad * self.P[u, :])) + (2 * (self.regularization_rate * self.Q[i, :])))
```

هدف از این آموزش min کردن تابع خطا در به روز رسانی های P , Q است، برای انجام این کار مشتق تابع خطا نسبت به P , Q محاسبه میشود و براساس آن آپدیت میشوند

$$Q_{i+1} = Q_i - \frac{\partial E}{\partial Q}$$
$$P_{i+1} = P_i - \frac{\partial E}{\partial P}$$

امتیاز پیش بینی هر کاربر (u)برای آیتم(i) به صورت زیر با استفاده از ماتریس P,Q به دست میاید:

```
def predict_rating(self, i, u):
    """
    Predict the rating for item i and user u.

    Args:
        i (int): Item index.
        u (int): User index.

    Returns:
        float: Predicted rating.
    """
    # Start your code
    return self.Q[i, :].dot(self.P[u, :].T)
    # End your code
```

مقادیر اولیه در نظر گرفته شده برای اجرا:

```
num_factors = 200

regularization_rate = 0.1

num_iterations = 50
```

برای به دست آوردن نرخ بهینه یادگیری، هر بار به ازای learning rate های متفاوت اجرا می شود تا دقت آنها مورد مقایسه قرار بگیرد:

```
learning_rate_list = [0.0001,0.015,0.001,0.1]
```

اجرا برای Lr های متفاوت به صورت زیر است:

```
Learning_rate = 0.0001
Predicted rating for item 0 and user 0: 75.71717332965171
Accuracy for model: 0.016721959788646965
*****************************************
<ipython-input-2-afb18204880e>:42: RuntimeWarning: overflow encountered in exp
  return 1 / (1 + np.exp(-x))
Learning_rate = 0.001
Predicted rating for item 0 and user 0: 1296.0185373579154
Accuracy for model: 0.006240015502583984
*****************************************
Learning_rate = 0.015
Predicted rating for item 0 and user 0: 1.1173554975690635e+32
Accuracy for model: 0.0064323211414480455
*****************************************
Learning_rate = 0.1
Predicted rating for item 0 and user 0: 3.5574048719900194e+235
Accuracy for model: 0.004659644174148825
*****************************************
```

بالاترین میزان دقت 0.016 میباشد که به ازای lr = 0.0001 است، بنابراین این مقدار نرخ بهینه یادگیری است.