

In the name of God



درس : یادگیری ماشین

استاد : دکتر زارع

دانشجو : توحید حقیقی سیس ( ۸۳۰۵۹۸۰۲۱ )

تمرین تئوری ها :

کل قسمت ها در شکل های زیر است:

2017 JAN 2017  
۱۰ رجب الثانی ۱۳۳۸

دی ۱۳۹۵  
۲۰  
دوشنبه

### Online learning

$$\Sigma = C_n = \frac{1}{n-1} \sum_{i=1}^n (x_i - m_n)(x_i - m_n)^T$$

dimensional  $\leftarrow d$   
تعداد مجهول  $\leftarrow n$

$$m_n = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\Rightarrow nm_n = \sum_{i=1}^n x_i \quad *$$

$$\rightarrow m_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \frac{1}{n+1} \left( \sum_{i=1}^n x_i + x_{n+1} \right) =$$

$$= \frac{1}{n+1} \left( nm_n + x_{n+1} \right) = \frac{n}{n+1} m_n + \frac{x_{n+1}}{n+1}$$

$n = n+1-1$

$$m_{n+1} = m_n + \frac{1}{n+1} [x_{n+1} - m_n] \quad 2.2$$

وقایع حضرت محمودة (س) ۱۱ - ۱۲ هـ ق - شهادت میرزا تقی خان امیر کبیر ۱۲۳۰ هـ ش

امام هادی (ع): خواندنیهای منابع تعلیمی، علم است و انسان را بسوی ترقی و جلال می کشاند.  
(میزان الحکمة، ج ۲، ص ۳۲)

www.bsi.ir

$$2.3 \quad C_{n+1} = \frac{n-1}{n} C_n + \frac{1}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T$$

$$C_{n+1} - \frac{n-1}{n} C_n = \frac{1}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T$$

تکثیر مشترک

$$nC_{n+1} - (n-1)C_n = \frac{n}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T$$

این را اثبات می کنیم

$$\rightarrow n \times \frac{1}{n} \sum_{i=1}^{n+1} (x_i - m_n)(x_i - m_n)^T = (1)$$

$$\rightarrow (n-1) \times \frac{1}{n-1} \sum_{i=1}^n (x_i - m_n)(x_i - m_n)^T = (2)$$

$$(1) - (2) = \sum_{i=1}^{n+1} x_i x_i^T - \sum_{i=1}^n x_i x_i^T - (n+1) m_n m_n^T + n m_n m_n^T$$

$$= \frac{n}{n+1} (x_{n+1} x_{n+1}^T - m_n x_{n+1}^T - x_{n+1} m_n^T + m_n m_n^T)$$

$$= \frac{n}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T$$

ii) How much time does the covariance matrix take per sequential? (Use big-O notation.)

$$C_{n+1} = \frac{n-1}{n} C_n + \frac{1}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T$$

$O(d^2)$

هر بار  $O(d^2)$  محاسبه می شود.

iii) show that we can sequentially update the precision matrix



دی ۱۳۹۵

۱۶

پنجشنبه

5 JAN 2017

۱۳۳۸ ربيع الثاني

$$C_{n+1} = \left( \frac{n-1}{n} C_n + \frac{1}{n+1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T \right)^{-1}$$

$$C_{n+1}^{-1} = \frac{n}{n-1} (*)$$

فرد خواسته شده

با فرض خواسته شده در صفحه قبل برآورد

$C_n$  را در  $t_k$  قبل محاسبه کرده پس با ذخیره مقدار آن از محاسبه (iv) محروم می شویم و می توان جلوی کندی را کم کرد پس  $C$  را می دانیم

what is the time complexity per update?

دی ۱۳۹۵

۱۷

جمعه

6 JAN 2017

۱۳۳۸ ربيع الثاني

$$\Rightarrow C_n^{-1} (x_{n+1} - m_n)(x_{n+1} - m_n)^T C_n^{-1}$$

تفصیل

سرعت  $O(d^2)$  پس در نهایت محاسبه  $C_{n+1}^{-1}$  در  $O(d^2)$

حضرت علی (ع): صبور بر سه گونه است: صبور بر محسبت، و صبور بر اطاعت، و صبور بر ترک [معصیت]  
(مجموعه روایات حضرت علی علیه السلام، ج ۱، ص ۱۰۰)

محاسبه می شود.

تمرین اول :

این تمرین ۳ قسمت دارد الگوریتم های کاهش بعد PCA , LDA است

الگوریتم اول الگوریتم PCA است که در زیر به تفسیر توضیح میدهیم :

مراحل پیاده سازی الگوریتم PCA :

۱- بردن داده ها به مرکز مختصات با پیدا کردن Mean داده ها و کم کردن از تک تک داده ها

۲- محاسبه ماتریس Scatter و یا Covariance

۳- محاسبه Eigen Vector, Eigen Value

۴- مرتب کردن Eigen Value و در آوردن ارگومان ها به ترتیب . که این نشان دهنده ویژگی های برتر است که میتوانیم به ان بعد ها کاهش دهیم

۵- یافتن ماتریس W\_matrix برای کاهش بعد داده ها

۶- ضرب ماتریس W\_matrix در داده های اولیه و به دست آوردن داده های جدید

از اول شروع به توضیح میکنیم :

۱- مرحله اول : بردن داده ها به مرکز مختصات با پیدا کردن Mean داده ها و کم کردن از تک تک داده ها

```
۲- داده ها را به مرکز منتقل میکند #  
۳- def fit_transform(self,data):  
۴-     mean_vector = self.Get_Mean_Vector(data)  
۵-     data -= mean_vector  
۶-     return data  
۷-
```

## ۲- محاسبه Scatter Matrix – Covariance

```
def Scatter_Matrix(self,data,mean_vector):
    scatter_matrix = np.zeros((data.shape[1], data.shape[1]))
    for i in range(data.shape[0]):
        scatter_matrix += (data.iloc[i, :].values.reshape(data.shape[1], 1) -
            mean_vector).dot((data.iloc[i, :].values.reshape(data.shape[1], 1) - mean_vector).T)

    # eigenvectors and eigenvalues for the from the scatter matrix
    # eig_val_sc, eig_vec_sc = np.linalg.eig(scatter_matrix)
    return scatter_matrix

# Co variance matrix
def CoVariance_Matrix(self,data):
    return data.cov()
```

## ۳- محاسبه Eigen Vector- Eigen Value

```
# eigen vector
def EigenVector(self,cov_matrix):
    # eigenvectors and eigenvalues for the from the covariance matrix
    eig_val_cov, eig_vec_cov = np.linalg.eig(cov_matrix)
    return eig_vec_cov

# eigen value
def EigenValue(self,cov_matrix):
    # eigenvectors and eigenvalues for the from the covariance matrix
    eig_val_cov, eig_vec_cov = np.linalg.eig(cov_matrix)
    return eig_val_cov
```

## ۴- مرتب کردن Eigen Value

```
#مرتب کردن داده ها
def Sort_EigenValue(self,eigen_value,count):
    # Make a list of (eigenvalue, eigenvector) tuples
    #eig_pairs = [(np.abs(eigen_value[i]), eig_vec[:,i]) for i in range(len(eigen_value))]
```

```

# Sort the (eigenvalue, eigenvector) tuples from high to low
#eig_pairs.sort(key=lambda x: x[0], reverse=True)

# # Visually confirm that the list is correctly sorted by decreasing eigenvalues
# for i in eig_pairs:
#     print(i[0])

return np.argsort(np.abs(eigen_value))[:,::-1][0:2]

```

## ۵- یافتن W\_matrix

```

# ماتریسی که با آن کاهش بعد را محاسبه میکنیم
def W_Matrix(self,data,eig_vec_cov,sorted_eigen):
    matrix_w = eig_vec_cov[:, sorted_eigen[0]].reshape(data.shape[1], -1)
    for i in range(len(sorted_eigen)-1):
        matrix_w = np.hstack((matrix_w, eig_vec_cov[:,sorted_eigen[i+1]].reshape(data.shape[1], -1)))
    return matrix_w

```

## ۶- کاهش بعد داده ها

```

#ماتریس کاهش بعد داده شده
def reduction_array(self,x,w_matrix):
    return x.values @ w_matrix

```

## ۷- کشیدن شکل

```

def Draw_Chart(self,x):
    fig, ax = plt.subplots(1, 1)
    sns.scatterplot(x=x[:,0], y=x[:,1], ax=ax)
    # show chart with new data
    plt.show()

```



## مراحل پیاده سازی LDA(FDA Demension Reduction)

تفاوت LDA با PCA در این است که در PCA کلاس ها مهم نیستند و کل داده ها کاهش بعد مییابند ولی در LDA برای هر کلاس جداگانه مراحل بالا اجرا میشوند و در اخر ماتریس ها جمع میشوند تا ماتریس نهایی را بدهد .

۱- بردن داده ها به مرکز مختصات با پیدا کردن Mean داده ها و کم کردن از تک تک داده ها در هر کلاس

۲- محاسبه ماتریس Scatter و یا Covariance هر کلاس و جمع ان ها

۳- محاسبه Eigen Vector, Eigen Value

۴- مرتب کردن Eigen Value و در آوردن ارگومان ها به ترتیب . که این نشان دهنده ویژگی های برتر است که میتوانیم به ان بعد ها کاهش دهیم

۵- یافتن ماتریس W\_matrix برای کاهش بعد داده ها

۶- ضرب ماتریس W\_matrix در داده های اولیه و به دست آوردن داده های جدید

مرحله اول : پیدا کردن Mean :

```
def Mean_Vector(self,x,y):
    mean_vectors = []
    for c in np.unique(y):
        mean_vectors.append(np.mean(x[y==c], axis=0))
    return mean_vectors
```

مرحله دوم : یافتن Scatter Matrix

```
def Scatter_Matrix(self,x,y,mean_vector):
    s_w = np.zeros((x.shape[1], x.shape[1]))
    for c, mv in zip(range(1, x.shape[1]), mean_vector):
        class_sc_mat = np.zeros((x.shape[1], x.shape[1]))
        for index, row in x[y == c].iterrows():
```

```

        row1, mv1 = (row.values.reshape(x.shape[1], 1), mv.values.reshape(x.shape[1], 1))
        class_sc_mat += (row1 - mv1).dot((row1 - mv1).T)
        s_w += class_sc_mat
    return s_w

```

مرحله سوم : محاسبه Eigen Vector , Eigen Value

```

def EigenVector(self,s_w,s_b):
    eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(s_w).dot(s_b))
    return eig_vecs

def EigenValue(self,s_w,s_b):
    eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(s_w).dot(s_b))
    return eig_vals

```

مرحله چهارم : مرتب کردن Eigen Value

```

def Component_Sort(self,eigen_val,count):
    return np.argsort(np.abs(eigen_val))[:,::-1][0:count]

```

مرحله پنجم : محاسبه W\_matrix

```

def W_Matrix(self,eig_vecs,sorted_vec,x):
    matrix_w = eig_vecs[:, sorted_vec[0]].reshape(x.shape[1], -1)
    for i in range(len(sorted_vec)-1):
        matrix_w = np.hstack((matrix_w, eig_vecs[:,sorted_vec[i+1]].reshape(x.shape[1], -1)))
    return matrix_w

```

مرحله ششم : کاهش بعد

```

def LDA_Matrix(self,x,matrix_w):
    final_matrix= x.values @ matrix_w.real
    sns.scatterplot(final_matrix[:,0], final_matrix[:,1])
    self.LDA_Matrix=final_matrix

```

پیاده سازی : LDA classifier

$$C' = \log \pi_k - \frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k)$$

Since the term  $\log C'$  does not depend on  $k$  and we aim to maximize the posterior probability over we can ignore it:

$$\begin{aligned} & \log \pi_k - \frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) \\ &= \log \pi_k - \frac{1}{2} [x^T \Sigma^{-1} x + \mu_k^T \Sigma^{-1} \mu_k] + x^T \Sigma^{-1} \mu_k \\ &= C'' + \log \pi_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + x^T \Sigma^{-1} \mu_k \end{aligned}$$

And so the objective function, sometimes called the *linear discriminant function* is:

$$\delta_k(x) = \log \pi_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + x^T \Sigma^{-1} \mu_k$$

Which means that given an input  $x$  we predict the class with the highest value of  $\delta_k(x)$ .

See [here](#) for an implementation in Python

با استفاده از فرمول بالا میتوانیم این الگوریتم را جداسازی کنیم

این الگوریتم به صورت احتمالی جواب را به ما میدهد :

```
def get_lda_score(self,X,MU_k,SIGMA,pi_k):
    #Returns the value of the linear discriminant score function for a given
class "k" and
    # a given x value X
    return (np.log(pi_k) - 1/2 * (MU_k).T @ np.linalg.inv(SIGMA)@(MU_k) + X.T
@ np.linalg.inv(SIGMA)@ (MU_k)).flatten()[0]

def predict_lda_class(self,X,MU_list,SIGMA,pi_list, y):
    scores_list = []
    classes = np.unique(y)
    for p in range(len(classes)):
        score = get_lda_score(X.reshape(-1,1),MU_list[p].reshape(-
1,1),SIGMA,pi_list[0])
        scores_list.append(score)
    self.FDA_List=scores_list
    return classes[np.argmax(scores_list)]
```

خروجی PCA به صورت زیر است :

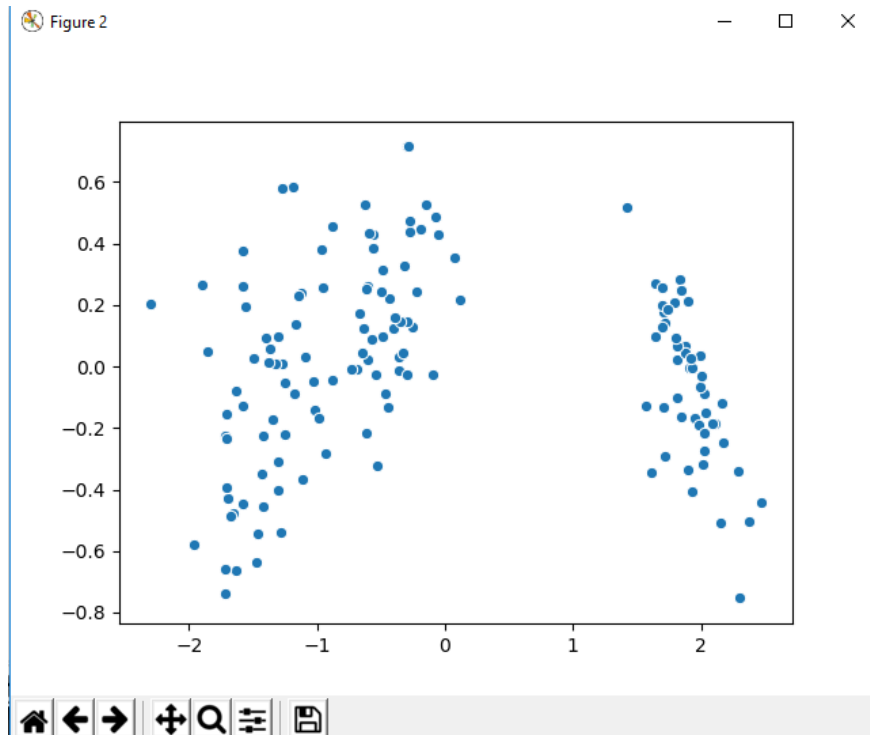


Figure 1 خروجی pca

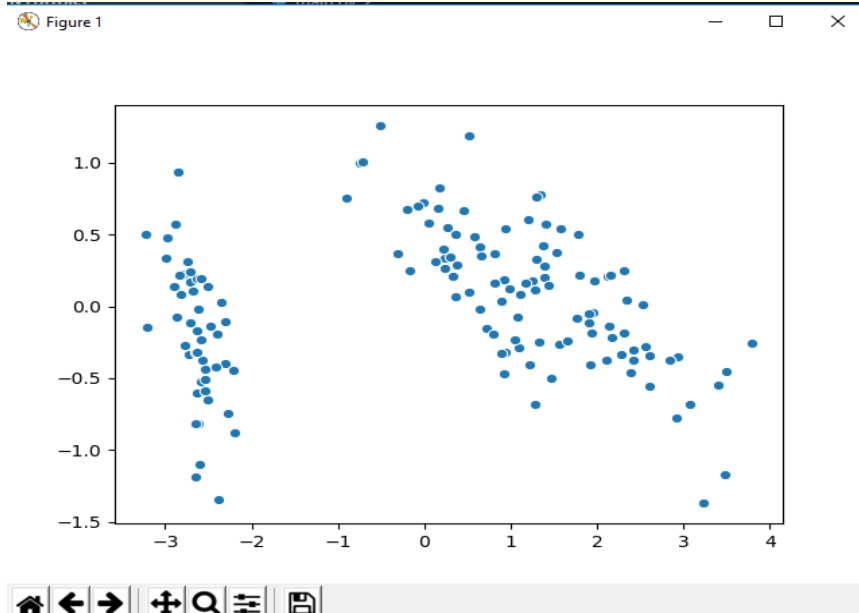


Figure 2 خروجی lda

در ادامه برای یافتن **Error rate** از تابع زیر استفاده میکنیم که داده ها را به ۵ قسمت مساوی تقسیم میکند و برای هر یک از داده های **Train , Test** را در الگوریتم محاسبه و خروجی به صورت زیر است .

```
y = column_or_1d(y, warn=True)  
0.12
```

Figure ۳ خروجی fda

```
y = column_or_1d(y, warn=True)  
0.18666666666666668
```

Figure ۴ خروجی fda

Accuracy in PCA = ۸۲ %

Accuracy in LDA = ۸۸ %