In the name of God

درس : یادگیری ماشین

استاد : دکتر زارع

دانشجو : توحید حقیقی سیس (۸۳۰۵۹۸۰۲۱)

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 Covarianse
۳- محاسبه Eigen Vector,Eigen Value
٤- مرتب کردن Eigen Value و در اوردن ارگومان ها به ترتیب . که این نشان دهنده ویژگی های برتر است که میتوانیم به ان بعد ها کاهش دهیم
٥- یافتن ماتریس W_matrix برای کاهش بعد داده ها
٦- ضرب ماتریس W_matrix در داده های اولیه و به دست اوردن 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

```
    # داده ها را به مرکز منتقل میکند
    def fit_transform(self,data):
        mean_vector = self.Get_Mean_Vector(data)
        data -= mean_vector
        return data

```

۲- محاسبه Scatter Matrix – Covarianse

```
    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 varianse matrix
    def CoVarianse_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]), eigen_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 eige
nvalues
        # 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 ان 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]].resh
ape(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 میبابند ولی در LDA برای هر کلاس جداگانه مراحل بالا اجرا میشوند و در اخر ماتریس ها جمع میشوند تا ماتریس نهایی را بدهد .

١- بردن داده ها به مرکز مختصات با پیدا کردن Mean داده ها و کم کردن از تک تک داده ها در هر کلاس
٢- محاسبه ماتریس Scatter و یا Covarianse هر کلاس و جمع ان ها
٣- محاسبه Eigen Vector,Eigen Value
٤- مرتب کردن Eigen Value و در اوردن ارگومان ها به ترتیب . که این نشان دهنده ویژگی های برتر است که میتوانیم به ان بعد ها کاهش دهیم
٥- یافتن ماتریس W_matrix برای کاهش بعد داده ها
٦- ضرب ماتریس W_matrix در داده های اولیه و به دست اوردن 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 :

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)]
```