پروژه اول درس یادگیری ماشین

استاد: دکتر فهیم

دانشجو: ریحانه اسماعیلی‌زاده

۸۱۰۸۰۰۰۰۴

سوال اول:

در این سوال هدف این بود با استفاده از بیشینه درست نمایی به صورت تئوری میانگین و واریانس را تخمین بزنیم. این مراحل بر روی برگه نوشته شده و در زیر قابل مشاهده است.

(۱ $\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} x_i \qquad \hat{\sigma}_n^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$

$\hat{\mu}_n = \frac{1}{8} \times (1 + 1.1 + 1.2 + 1.6 + 0.9 + 0.7 + 0.6 + 1.005) = 0.988$

$\hat{\sigma}_n^2 = \frac{1}{8} \times \left(\sum (x_i - 0.988)^2 \right) = 0.081$

قانون اعداد بزرگ ← اگر تعداد زیادی نمونه از توزیع ای داشته باشیم (یعنی n ها زیاد باشد) مثل توزیع $\hat{\mu}$ آنگاه میانگین آنها به میانگین جامعه‌ای با μ جامعه نزدیک می‌باشد.

اینجا نیز μ جامعه 0.988 فرض شده و میانگین 1000 داده نیز 0.992 شده است که نزدیک به μ می‌باشد.

سپس به سراغ قسمت پیاده‌سازی رفتیم. در این قسمت با استفاده از کتابخانه‌های نامپای[۱] و سایپای[۲]، محاسبات را انجام دادیم و هر قسمت را چاپ کردیم که در صورت اجرا خروجی قابل مشاهده است. کد استفاده شده در زیر قرار گرفته است.

```python
import numpy
import matplotlib.pyplot as plt
from scipy.stats import norm
import scipy.stats as stats
dataarray = [1,1.1,1.2,1.5,0.9,0.7,0.5,1.005]
xaxis = numpy.arange(-2,2,0.01)
variance =numpy.var(dataarray)
average = numpy.mean(dataarray)
print(variance)
print(average)
plt.plot(xaxis, norm.pdf(xaxis,average,numpy.sqrt(variance)))
plt.show()
value = stats.norm.pdf(-1,average,numpy.sqrt(variance))
print(value)
random = numpy.random.normal(average, numpy.sqrt(variance),1000)
averagenew = numpy.mean(random)
print(averagenew)
```

سوال دوم:

ابتدا طبق خواسته سوال این دیتاست را از کتابخانه سایکیت لرن[۳] استخراج کردیم. سپس طبق بخش (الف) ویژگی‌های خواسته شده را مدنظر قرار دادیم. پس از آن با استفاده از کتابخانه مت پلات لیب[۴] داده‌ها در یک فضای دوبعدی و بر اساس ویژگی‌های گفته شده در بخش (ب) نمایش دادیم. سپس برای بخش (ج) طبق کد زیر ابتدا میانگین هر ستون و سپس ماتریس مرکزی و پس از آن برای بخش (د) با استفاده

[1] Numpy
[2] Scipy
[3] Sklearn
[4] Matplotlib

از ماتریس مرکزی بخش قبل، کوواریانس را محاسبه کردیم که در زیر قابل مشاهده است.

```
nselected_features = iris_df.iloc[:, :3]
mean = numpy.mean(nselected_features, axis = 0)
print('mean')
print(mean)
centralized_matrix = nselected_features - mean
print(centralized_matrix)
n = centralized_matrix.shape[0]
covariance_matrix = (1/n) * numpy.transpose(centralized_matrix) @
centralized_matrix
```

پس از آن برای بخش (ه) داده‌های پرت را طبق حدود گفته شده حذف کردیم و دوباره مراحل بخش قبل را تکرار کردیم.

```
Q1 = numpy.percentile(nselected_features, 25, axis=0)
Q3 = numpy.percentile(nselected_features, 75, axis=0)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
mask = numpy.all((nselected_features >= lower_bound) & (nselected_features <=
upper_bound), axis=1)
new_dataset = nselected_features[mask]
mean1 = numpy.mean(new_dataset, axis = 0)
print('new mean')
print(mean1)
centralized_matrix2 = new_dataset - mean1
nn = centralized_matrix2.shape[0]
print('centralized matrix 2')
print(centralized_matrix2)
covariance_matrix2 = (1/(nn)) * numpy.transpose(centralized_matrix2) @
centralized_matrix2
print('covarience matrix 2')
covariance_matrix2 = numpy.array([[0.690946, -0.040585, 1.280638],
                            [-0.040585, 0.157203, -0.282126],
                            [1.280638, -0.282126, 3.069542]])
print(covariance_matrix2)
```

این تغییر را میتوان به این گونه تعبیر کرد که ۳ داده پرت حذف شده‌اند، واریانس ویژگی اول کمی افزایش پیدا کرده و واریانس ویژگی دوم کاهش پیدا کرده است و در همین حال واریانس ویژگی سوم نیز کمی کاهش یافته. همچنین همبستگی خطی مثبت بین دوبه‌دوی ویژگی‌ها افزایش پیدا کرده است.

برای بخش (و) همبستگی پیرسون[۵] را برای دوبه‌دوی داده‌ها محاسبه کردیم و پاسخ‌هارا مقایسه کردیم که مشاهده شد نتایج تفاوتی ندارند و یکی هستند.

```
pearson_correlation = numpy.zeros((3,3))
for i in range(3):
    for j in range(3):
        pearson_correlation[i][j] = covariance_matrix2[i][j] /
(numpy.sqrt(covariance_matrix2[i][i]) * numpy.sqrt(covariance_matrix2[j][j]))

print('pearson correlation : ')
print(pearson_correlation)
correlation_matrix_pandas = new_dataset.corr(method='pearson')
print("\nCorrelation Matrix (Pandas):")
print(correlation_matrix_pandas)
```

برای بخش اخر سوال دوم یعنی بخش (ز) طبق مراحل کتاب پیش رفتیم و با روش تحلیل مولفه اصلی[۶] ابعاد را کاهش دادیم و توانستیم خواسته سوال را رعایت کنیم و نتایج با مثال کتاب یکسان و یکی بودند.

```
eigenvalues, eigenvectors = numpy.linalg.eig(covariance_matrix)
eigenvalues = numpy.sort(eigenvalues)[::-1]
print('eigenvectors')
print(eigenvectors)
print('eigenvalues')
print(eigenvalues)
f = (eigenvalues[0] + eigenvalues[1])/ (eigenvalues[0] + eigenvalues[1] +
eigenvalues[2])
print('f :')
```

[5] Pearson Correlation
[6] Principal Component Cnalysis (PCA)

```python
print(f)
if f <= 0.95:
    print ('We should not reduce third dimension!')
else:
    print('We should reduce the third dimension! ')
```

سوال سوم:

ابتدا طبقه خواسته سوال ویژگی پنجم را حذف کردیم. سپس به سراغ بخش (الف) رفتیم کرنل مربعی را محاسبه کردیم.

```python
def scaler_square_kernel(x_i, x_j):
    result = numpy.dot(numpy.array(x_i), numpy.array(x_j).T)
    return result ** 2


def matrix_square_kernel(data):
    n = data.shape[0]
    kernel_matrix = numpy.zeros((n, n))
    for i in range(n):
        for j in range(n):
            kernel_matrix[i, j] = scaler_square_kernel(np.array(data.iloc[i, :]),
np.array(data.iloc[j, :]))

    return kernel_matrix
```

سپس برای بخش (ب) برای هر X_i مقدار ϕ را حساب کردیم که در کد ارسال شده قابل مشاهده است. سپس برای بخش (ج) با استفاده از نتایج به دست آمده و ضرب آن‌ها ماتریس کرنل را مجددا ساختیم و نتایج را با بخش (الف) مقایسه کردیم که یکسان بودند.

```python
my_square_kernel_matrix = matrix_square_kernel(selected_features)
eigenvalues, eigenvectors = numpy.linalg.eig(np.array(my_square_kernel_matrix))
print(numpy.dot(eigenvectors.T, eigenvectors))
print('eignvalues')
```

```
print(eigenvalues)
print('eigenvectors')
print(eigenvectors)
eigenvalues = approximate_zero(eigenvalues, 0.0000001)
eigenvectors = approximate_zero(eigenvectors, 0.0000001)
eigenvalues_matrix = numpy.diag(eigenvalues)
print(eigenvalues_matrix.shape)
print("Eigenvalues Matrix:")
print(eigenvalues_matrix)

phi_vector = phi_cal(eigenvalues_matrix, eigenvectors.T)
print('phi_vectors')
print(phi_vector.shape)
print(phi_vector)

new_kernel = numpy.dot(phi_vector.T, phi_vector)
print('my_square_kernel_matrix')
print(my_square_kernel_matrix.shape)
print(my_square_kernel_matrix)
print('new_kernel')
print(new_kernel.shape)
print(new_kernel)

if numpy.array_equal(new_kernel, my_square_kernel_matrix):
    print("The matrices are equal.")
else:
    print("The matrices are not equal.")
```

سپس در نهایت برای بخش (د) الگوریتم تحلیل مولفه اصلی کرنل[۷] را بر روی ماتریس کرنل مرکزی پیاده کردیم و مشاهده شد با حفظ تنها یک ویژگی ۹۳ درصد از اطلاعات حفظ می‌شود.

```
centeralizedkernelmatrise = my_square_kernel_matrix - 
numpy.mean(my_square_kernel_matrix, axis = 0)
eigenvaluescen, eigenvectorscen = numpy.linalg.eig(centeralizedkernelmatrise)
print('eigenvectorscen')
print(eigenvectorscen.T)
print('transpose')
print(eigenvectorscen)
```

[7] Kernel PCA

```python
eigenvaluescen = approximate_zero(eigenvaluescen, 0.0000001)
eigenvectorscen = approximate_zero(eigenvectorscen, 0.0000001)
n = eigenvaluescen.shape[0]
print(n)
landa = eigenvaluescen/n
print(landa)
print('landa')
print(eigenvaluescen.shape)
eigenvectorscen = eigenvectorscen.T / np.sqrt(eigenvaluescen[:, np.newaxis])
print('eigenvectorscen')
print(eigenvectorscen.shape)
print('eigenvaluescen')
print(eigenvaluescen)
r = find_smallest_r(landa, 0.90)
print('r : ')
print(r)
neweigenvectorscen = eigenvectorscen[:][:r]
final = neweigenvectorscen.T
print(neweigenvectorscen.shape)
analysis = np.sum(landa[:r]) / sum(landa)
print(analysis)
Am = final.T @ centeralizedkernelmatrise
print('A matris : ')
print(Am.shape)
print(Am)
```

با تشکر از توجه شما.