

Метод главных компонент

В данном задании вам будет предложено ознакомиться с подходом, который переоткрывался в самых разных областях, имеет множество разных интерпретаций, а также несколько интересных обобщений: методом главных компонент (principal component analysis).

Programming assignment

Задание разбито на две части:

- работа с модельными данными,
- работа с реальными данными.

В конце каждого пункта от вас требуется получить ответ и загрузить в соответствующую форму в виде набора текстовых файлов.

```
In [ ]: import numpy as np
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches
matplotlib.style.use('ggplot')
%matplotlib inline
```

Теория

Любой набор данных представляет собой матрицу X .

Метод главных компонент последовательно находит следующие линейные комбинации признаков (компоненты) из X :

- каждая компонента ортогональна всем остальным и нормированна: $\langle w_i, w_j \rangle = 0, \quad ||w_i|| = 1$,
- каждая компонента описывает максимально возможную дисперсию данных (с учётом предыдущего ограничения).

Предположения, в рамках которых данный подход будет работать хорошо:

- линейность компонент: мы предполагаем, что данные можно анализировать линейными методами,
- большие дисперсии важны: предполагается, что наиболее важны те направления в данных, вдоль которых они имеют наибольшую дисперсию,
- все компоненты ортогональны: это предположение позволяет проводить анализ главных компонент при помощи техник линейной алгебры (например, сингулярное разложение матрицы X или спектральное разложение матрицы $X^T X$).

Как это выглядит математически?

Обозначим следующим образом выборочную матрицу ковариации данных: $\hat{C} \propto Q = X^T X$. (Q отличается от \hat{C} нормировкой на число объектов).

Сингулярное разложение матрицы Q выглядит следующим образом:

$$Q = X^T X = W \Lambda W^T$$

Можно строго показать, что столбцы матрицы W являются главными компонентами матрицы X , т.е. комбинациями признаков, удовлетворяющих двум условиям, указанным в начале. При этом дисперсия данных вдоль направления, заданного каждой компонентой, равна соответствующему значению диагональной матрицы Λ .

Как же на основании этого преобразования производить уменьшение размерности? Мы можем отранжировать компоненты, используя значения дисперсий данных вдоль них.

Сделаем это: $\lambda_{(1)} > \lambda_{(2)} > \dots > \lambda_{(D)}$.

Тогда, если мы выберем компоненты, соответствующие первым d дисперсиям из этого списка, мы получим набор из d новых признаков, которые наилучшим образом описывают дисперсию изначального набора данных среди всех других возможных линейных комбинаций исходных признаков матрицы X .

- Если $d = D$, то мы вообще не теряем никакой информации.
- Если $d < D$

Получается, что метод главных компонент позволяет нам ранжировать полученные компоненты по "значимости", а также запустить процесс их отбора.

Пример

Рассмотрим набор данных, который сэмплирован из многомерного нормального распределения с матрицей ковариации $C = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}$.

```
In [ ]: from sklearn.decomposition import PCA

mu = np.zeros(2)
C = np.array([[3,1],[1,2]])

data = np.random.multivariate_normal(mu, C, size=50)
plt.scatter(data[:,0], data[:,1])
plt.show()
```

Путём диагонализации истинной матрицы ковариаций C , мы можем найти преобразование исходного набора данных, компоненты которого наилучшим образом будут описывать дисперсию, с учётом их ортогональности друг другу:

```
In [ ]: v, W_true = np.linalg.eig(C)

plt.scatter(data[:,0], data[:,1])
# построим истинные компоненты, вдоль которых максимальна дисперсия данн
ых
plt.plot(data[:,0], (W_true[0,0]/W_true[0,1])*data[:,0], color="g")
plt.plot(data[:,0], (W_true[1,0]/W_true[1,1])*data[:,0], color="g")
g_patch = mpatches.Patch(color='g', label='True components')
plt.legend(handles=[g_patch])
plt.axis('equal')
limits = [np.minimum(np.amin(data[:,0]), np.amin(data[:,1])),
           np.maximum(np.amax(data[:,0]), np.amax(data[:,1]))]
plt.xlim(limits[0],limits[1])
plt.ylim(limits[0],limits[1])
plt.draw()
```

А теперь сравним эти направления с направлениями, которые выбирает метод главных компонент:

```
In [ ]: def plot_principal_components(data, model, scatter=True, legend=True):
    W_pca = model.components_
    if scatter:
        plt.scatter(data[:,0], data[:,1])
    plt.plot(data[:,0], -(W_pca[0,0]/W_pca[0,1])*data[:,0], color="c")
    plt.plot(data[:,0], -(W_pca[1,0]/W_pca[1,1])*data[:,0], color="c")
    if legend:
        c_patch = mpatches.Patch(color='c', label='Principal components')
    )
    plt.legend(handles=[c_patch], loc='lower right')
    # сделаем графики красивыми:
    plt.axis('equal')
    limits = [np.minimum(np.amin(data[:,0]), np.amin(data[:,1]))-0.5,
                       np.maximum(np.amax(data[:,0]), np.amax(data[:,1]))+0.5]
    plt.xlim(limits[0],limits[1])
    plt.ylim(limits[0],limits[1])
    plt.draw()
```

```
In [ ]: model = PCA(n_components=2)
        model.fit(data)

        plt.scatter(data[:,0], data[:,1])
        # построим истинные компоненты, вдоль которых максимальна дисперсия данн
        ых
        plt.plot(data[:,0], (W_true[0,0]/W_true[0,1])*data[:,0], color="g")
        plt.plot(data[:,0], (W_true[1,0]/W_true[1,1])*data[:,0], color="g")
        # построим компоненты, полученные с использованием метода PCA:
        plot_principal_components(data, model, scatter=False, legend=False)
        c_patch = mpatches.Patch(color='c', label='Principal components')
        plt.legend(handles=[g_patch, c_patch])
        plt.draw()
```

Видно, что уже при небольшом количестве данных они отличаются незначительно. Увеличим размер выборки:

```
In [ ]: data_large = np.random.multivariate_normal(mu, C, size=5000)

        model = PCA(n_components=2)
        model.fit(data_large)
        plt.scatter(data_large[:,0], data_large[:,1], alpha=0.1)
        # построим истинные компоненты, вдоль которых максимальна дисперсия данн
        ых
        plt.plot(data_large[:,0], (W_true[0,0]/W_true[0,1])*data_large[:,0], col
        or="g")
        plt.plot(data_large[:,0], (W_true[1,0]/W_true[1,1])*data_large[:,0], col
        or="g")
        # построим компоненты, полученные с использованием метода PCA:
        plot_principal_components(data_large, model, scatter=False, legend=False
        )
        c_patch = mpatches.Patch(color='c', label='Principal components')
        plt.legend(handles=[g_patch, c_patch])
        plt.draw()
```

В этом случае главные компоненты значительно точнее приближают истинные направления данных, вдоль которых наблюдается наибольшая дисперсия.

Статистический взгляд на модель

Как формализовать предположения метода, указанные выше? При помощи вероятностной модели!

Задача, стоящая за любым методом уменьшения размерности: получить из набора зашумлённых признаков X истинные значения Y , которые на самом деле определяют набор данных (т.е. сведение датасета с большим количеством признаков к данным, имеющим т.н. "эффективную размерность").

В случае метода главных компонент мы хотим найти направления, вдоль которых максимальна дисперсия, с учётом описанных выше предположений о структуре данных и компонент.

Материал, описанный ниже в данной секции, не обязателен для ознакомления для выполнения следующего задания, т.к. требует некоторых знаний статистики.

Для тех, кто собирается его пропустить: в конце раздела мы получим метрику качества, которая должна определять, насколько данные хорошо описываются построенной моделью при заданном числе компонент. Отбор признаков при этом сводится к тому, что мы выбираем то количество компонент, при котором используемая метрика (логарифм правдоподобия) является максимальной.

С учётом предположений задача метода главных компонент выглядит следующим образом:

$$x = Wy + \mu + \epsilon$$

где:

- x -- наблюдаемые данные
- W -- матрица главных компонент (каждый столбец -- одна компонента)
- y -- их проекция на главные компоненты
- μ -- среднее наблюдаемых данных
- $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ -- нормальный шум

Исходя из распределения шума, выпишем распределение на x :

$$p(x | y) = \mathcal{N}(Wy + \mu, \sigma^2 I)$$

Введём априорное распределение на y :

$$p(y) = \mathcal{N}(0, 1)$$

Выведем из этого при помощи формулы Байеса маргинальное распределение на $p(x)$:

$$p(x) = \mathcal{N}(\mu, \sigma^2 I + WW^T)$$

Тогда правдоподобие набора данных при условии используемой модели выглядит следующим образом:

$$\mathcal{L} = \sum_{i=1}^N \log p(x_i) = -N/2 \left(d \log(2\pi) + \log |C| + \text{tr}(C^{-1}S) \right)$$

где:

- $C = \sigma^2 I + WW^T$ -- матрица ковариации в маргинальной модели
- $S = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)(x_i - \mu)^T$ -- выборочная ковариация

Значение \mathcal{L} имеет смысл логарифма вероятности получения набора данных X при условии, что он удовлетворяет предположениям модели метода главных компонент. Чем оно больше -- тем лучше модель описывает наблюдаемые данные.

Задание 1. Автоматическое уменьшение размерности данных при помощи логарифма правдоподобия \mathcal{L}

Рассмотрим набор данных размерности D , чья реальная размерность значительно меньше наблюдаемой (назовём её d). От вас требуется:

1. Для каждого значения \hat{d} в интервале $[1, D]$ построить модель PCA с \hat{d} главными компонентами.
2. Оценить средний логарифм правдоподобия данных для каждой модели на генеральной совокупности, используя метод кросс-валидации с 3 фолдами (итоговая оценка значения логарифма правдоподобия усредняется по всем фолдам).
3. Найти модель, для которой он максимален, и внести в файл ответа число компонент в данной модели, т.е. значение \hat{d}_{opt} .

Для оценки логарифма правдоподобия модели для заданного числа главных компонент при помощи метода кросс-валидации используйте следующие функции:

```
model = PCA(n_components=n)
scores = cv_score(model, data)
```

Обратите внимание, что scores -- это вектор, длина которого равна числу фолдов. Для получения оценки на правдоподобие модели его значения требуется усреднить.

Для визуализации оценок можете использовать следующую функцию:

```
plot_scores(d_scores)
```

которой на вход передаётся вектор полученных оценок логарифма правдоподобия данных для каждого \hat{d} .

Для интересующихся: данные для заданий 1 и 2 были сгенерированы в соответствии с предполагаемой PCA моделью. То есть: данные Y с эффективной размерностью d , полученные из независимых равномерных распределений, линейно трансформированы случайной матрицей W в пространство размерностью D , после чего ко всем признакам был добавлен независимый нормальный шум с дисперсией σ .

```
In [ ]: from sklearn.decomposition import PCA
        from sklearn.cross_validation import cross_val_score as cv_score

        def plot_scores(d_scores):
            n_components = np.arange(1, d_scores.size+1)
            plt.plot(n_components, d_scores, 'b', label='PCA scores')
            plt.xlim(n_components[0], n_components[-1])
            plt.xlabel('n components')
            plt.ylabel('cv scores')
            plt.legend(loc='lower right')
            plt.show()

        def write_answer_1(optimal_d):
            with open("pca_answer1.txt", "w") as fout:
                fout.write(str(optimal_d))

        data = pd.read_csv('data_task1.csv')

        # place your code here
        pass
```

Вариационный взгляд на модель

Мы знаем, что каждой главной компоненте соответствует описываемая ей дисперсия данных (дисперсия данных при проекции на эту компоненту). Она численно равна значению диагональных элементов матрицы Λ , получаемой из спектрального разложения матрицы ковариации данных (смотри теорию выше).

Исходя из этого, мы можем отсортировать дисперсию данных вдоль этих компонент по убыванию, и уменьшить размерность данных, отбросив q итоговых главных компонент, имеющих наименьшую дисперсию.

Делать это можно двумя разными способами. Например, если вы в дальнейшем обучаете на данных с уменьшенной размерностью модель классификации или регрессии, то можно запустить итерационный процесс: удалять компоненты с наименьшей дисперсией по одной, пока качество итоговой модели не станет значительно хуже.

Более общий способ отбора признаков заключается в том, что вы можете посмотреть на разности в дисперсиях в отсортированном ряде $\lambda_{(1)} > \lambda_{(2)} > \dots > \lambda_{(D)}$: $\lambda_{(1)} - \lambda_{(2)}, \dots, \lambda_{(D-1)} - \lambda_{(D)}$, и удалить те компоненты, на которых разность будет наибольшей. Именно этим методом вам и предлагается воспользоваться для тестового набора данных.

Задание 2. Ручное уменьшение размерности признаков посредством анализа дисперсии данных вдоль главных компонент

Рассмотрим ещё один набор данных размерности D , чья реальная размерность значительно меньше наблюдаемой (назовём её также d). От вас требуется:

1. Построить модель PCA с D главными компонентами по этим данным.
2. Спроецировать данные на главные компоненты.
3. Оценить их дисперсию вдоль главных компонент.
4. Отсортировать дисперсии в порядке убывания и получить их попарные разности: $\lambda_{(i-1)} - \lambda_{(i)}$.
5. Найти разность с наибольшим значением и получить по ней оценку на эффективную размерность данных \hat{d} .
6. Построить график дисперсий и убедиться, что полученная оценка на \hat{d}_{opt} действительно имеет смысл, после этого внести полученное значение \hat{d}_{opt} в файл ответа.

Для построения модели PCA используйте функцию:

```
model.fit(data)
```

Для трансформации данных используйте метод:

```
model.transform(data)
```

Оценку дисперсий на трансформированных данных от вас потребуется реализовать вручную. Для построения графиков можно воспользоваться функцией

```
plot_variances(d_variances)
```

которой следует передать на вход отсортированный по убыванию вектор дисперсий вдоль компонент.

```
In [ ]: from sklearn.decomposition import PCA
        from sklearn.cross_validation import cross_val_score as cv_score

        def plot_variances(d_variances):
            n_components = np.arange(1,d_variances.size+1)
            plt.plot(n_components, d_variances, 'b', label='Component variances'
            )
            plt.xlim(n_components[0], n_components[-1])
            plt.xlabel('n components')
            plt.ylabel('variance')
            plt.legend(loc='upper right')
            plt.show()

        def write_answer_2(optimal_d):
            with open("pca_answer2.txt", "w") as fout:
                fout.write(str(optimal_d))

        data = pd.read_csv('data_task2.csv')

        # place your code here
        pass
```

Интерпретация главных компонент

В качестве главных компонент мы получаем линейные комбинации исходных признаков, поэтому резонно возникает вопрос об их интерпретации.

Для этого существует несколько подходов, мы рассмотрим два:

- рассчитать взаимосвязи главных компонент с исходными признаками
- рассчитать вклады каждого конкретного наблюдения в главные компоненты

Первый способ подходит в том случае, когда все объекты из набора данных не несут для нас никакой семантической информации, которая уже не запечатлена в наборе признаков.

Второй способ подходит для случая, когда данные имеют более сложную структуру. Например, лица для человека несут больший семантический смысл, чем вектор значений пикселей, которые анализирует PCA.

Рассмотрим подробнее способ 1: он заключается в подсчёте коэффициентов корреляций между исходными признаками и набором главных компонент.

Так как метод главных компонент является линейным, то предлагается для анализа использовать корреляцию Пирсона, выборочный аналог которой имеет следующую формулу:

$$r_{jk} = \frac{\sum_{i=1}^N (x_{ij} - \bar{x}_j)(y_{ik} - \bar{y}_k)}{\sqrt{\sum_{i=1}^N (x_{ij} - \bar{x}_j)^2 \sum_{i=1}^N (y_{ik} - \bar{y}_k)^2}}$$

где:

- \bar{x}_j -- среднее значение j-го признака,
- \bar{y}_k -- среднее значение проекции на k-ю главную компоненту.

Корреляция Пирсона является мерой линейной зависимости. Она равна 0 в случае, когда величины независимы, и ± 1 , если они линейно зависимы. Исходя из степени корреляции новой компоненты с исходными признаками, можно строить её семантическую интерпретацию, т.к. смысл исходных признаков мы знаем.

Задание 3. Анализ главных компонент при помощи корреляций с исходными признаками.

1. Обучите метод главных компонент на датасете `iris`, получите преобразованные данные.
2. Посчитайте корреляции исходных признаков с их проекциями на первые две главные компоненты.
3. Для каждого признака найдите компоненту (из двух построенных), с которой он коррелирует больше всего.
4. На основании п.3 сгруппируйте признаки по компонентам. Составьте два списка: список номеров признаков, которые сильнее коррелируют с первой компонентой, и такой же список для второй. Нумерацию начинать с единицы. Передайте оба списка функции `write_answer_3`.

Набор данных состоит из 4 признаков, посчитанных для 150 ирисов. Каждый из них принадлежит одному из трёх видов. Визуализацию проекции данного датасета на две компоненты, которые описывают наибольшую дисперсию данных, можно получить при помощи функции

```
plot_iris(transformed_data, target, target_names)
```

на вход которой требуется передать данные, преобразованные при помощи PCA, а также информацию о классах. Цвет точек отвечает одному из трёх видов ириса.

Для того чтобы получить имена исходных признаков, используйте следующий список:

```
iris.feature_names
```

При подсчёте корреляций не забудьте центрировать признаки и проекции на главные компоненты (вычитать из них среднее).

```
In [ ]: from sklearn import datasets

def plot_iris(transformed_data, target, target_names):
    plt.figure()
    for c, i, target_name in zip("rgb", [0, 1, 2], target_names):
        plt.scatter(transformed_data[target == i, 0],
                    transformed_data[target == i, 1], c=c, label=target_
name)
    plt.legend()
    plt.show()

def write_answer_3(list_pc1, list_pc2):
    with open("pca_answer3.txt", "w") as fout:
        fout.write(" ".join([str(num) for num in list_pc1]))
        fout.write(" ")
        fout.write(" ".join([str(num) for num in list_pc2]))

# загрузим датасет iris
iris = datasets.load_iris()
data = iris.data
target = iris.target
target_names = iris.target_names

# place your code here
```

Интерпретация главных компонент с использованием данных

Рассмотрим теперь величину, которую можно проинтерпретировать, как квадрат косинуса угла между объектом выборки и главной компонентой:

$$\cos_{ik}^2 = \frac{f_{ik}^2}{\sum_{\ell=1}^d f_{i\ell}^2}$$

где

- i -- номер объекта
- k -- номер главной компоненты
- f_{ik} -- модуль центрированной проекции объекта на компоненту

Очевидно, что

$$\sum_{k=1}^d \cos_{ik}^2 = 1$$

Это значит, что для каждого объекта мы в виде данной величины получили веса, пропорциональные вкладу, которую вносит данный объект в дисперсию каждой компоненты. Чем больше вклад, тем более значим объект для описания конкретной главной компоненты.

Задание 4. Анализ главных компонент при помощи вкладов в их дисперсию отдельных объектов

1. Загрузите датасет лиц Olivetti Faces и обучите на нём модель RandomizedPCA (используется при большом количестве признаков и работает быстрее, чем обычный PCA). Получите проекции признаков на 10 первых главных компонент.
2. Посчитайте для каждого объекта его относительный вклад в дисперсию каждой из 10 компонент, используя формулу из предыдущего раздела ($d = 10$).
3. Для каждой компоненты найдите и визуализируйте лицо, которое вносит наибольший относительный вклад в неё. Для визуализации используйте функцию

```
plt.imshow(image.reshape(image_shape))
```

4. Передайте в функцию `write_answer_4` список номеров лиц с наибольшим относительным вкладом в дисперсию каждой из компонент, список начинается с 0.

```
In [ ]: from sklearn.datasets import fetch_olivetti_faces
        from sklearn.decomposition import RandomizedPCA

        def write_answer_4(list_pc):
            with open("pca_answer4.txt", "w") as fout:
                fout.write(" ".join([str(num) for num in list_pc]))

        data = fetch_olivetti_faces(shuffle=True, random_state=0).data
        image_shape = (64, 64)
```

Анализ основных недостатков метода главных компонент

Рассмотренные выше задачи являются, безусловно, модельными, потому что данные для них были сгенерированы в соответствии с предположениями метода главных компонент. На практике эти предположения, естественно, выполняются далеко не всегда. Рассмотрим типичные ошибки PCA, которые следует иметь в виду перед тем, как его применять.

Направления с максимальной дисперсией в данных неортогональны

Рассмотрим случай выборки, которая сгенерирована из двух вытянутых нормальных распределений:

```
In [ ]: C1 = np.array([[10,0],[0,0.5]])
        phi = np.pi/3
        C2 = np.dot(C1, np.array([[np.cos(phi), np.sin(phi)],
                                   [-np.sin(phi), np.cos(phi)]]))

        data = np.vstack([np.random.multivariate_normal(mu, C1, size=50),
                           np.random.multivariate_normal(mu, C2, size=50)])
        plt.scatter(data[:,0], data[:,1])
        # построим истинные интересующие нас компоненты
        plt.plot(data[:,0], np.zeros(data[:,0].size), color="g")
        plt.plot(data[:,0], 3*0.5*data[:,0], color="g")
        # обучим модель pca и построим главные компоненты
        model = PCA(n_components=2)
        model.fit(data)
        plot_principal_components(data, model, scatter=False, legend=False)
        c_patch = mpatches.Patch(color='c', label='Principal components')
        plt.legend(handles=[g_patch, c_patch])
        plt.draw()
```

В чём проблема, почему pca здесь работает плохо? Ответ прост: интересующие нас компоненты в данных коррелированы между собой (или неортогональны, в зависимости от того, какой терминологией пользоваться). Для поиска подобных преобразований требуются более сложные методы, которые уже выходят за рамки метода главных компонент.

Для интересующихся: то, что можно применить непосредственно к выходу метода главных компонент, для получения подобных неортогональных преобразований, называется методами ротации. Почитать о них можно в связи с другим методом уменьшения размерности, который называется Factor Analysis (FA), но ничего не мешает их применять и к главным компонентам.

Интересное направление в данных не совпадает с направлением максимальной дисперсии

Рассмотрим пример, когда дисперсии не отражают интересующих нас направлений в данных:

```
In [ ]: C = np.array([[0.5,0],[0,10]])
        mu1 = np.array([-2,0])
        mu2 = np.array([2,0])

        data = np.vstack([np.random.multivariate_normal(mu1, C, size=50),
                           np.random.multivariate_normal(mu2, C, size=50)])
        plt.scatter(data[:,0], data[:,1])
        # обучим модель pca и построим главные компоненты
        model = PCA(n_components=2)
        model.fit(data)
        plot_principal_components(data, model)
        plt.draw()
```

Очевидно, что в данном случае метод главных компонент будет считать вертикальную компоненту более значимой для описания набора данных, чем горизонтальную.

Но, например, в случае, когда данные из левого и правого кластера относятся к разным классам, для их линейной разделимости вертикальная компонента является шумовой. Несмотря на это, её метод главных компонент никогда шумовой не признает, и есть вероятность, что отбор признаков с его помощью выкинет из ваших данных значимые для решаемой вами задачи компоненты просто потому, что вдоль них значения имеют низкую дисперсию.

Справляться с такими ситуациями могут некоторые другие методы уменьшения размерности данных, например, метод независимых компонент (Independent Component Analysis, ICA).