Principle Component Analysis

Author: Sakura

Санкт-Петербург, 2025

> “The darker the night, the brighter the stars.”
>
> “Even the darkest night will end and the sun will rise.”

Теоретическая часть

PCA (Principal Component Analysis) — это метод снижения размерности данных, основанный на поиске новых ортогональных осей, которые максимизируют дисперсию проекции данных. Эти новые оси называются главными компонентами.

Этапы работы PCA

PCA состоит из нескольких ключевых шагов:

1. **Центрирование данных:**
   Перенести систему координат так, чтобы среднее значение каждого признака стало равным нулю.
   Формула:

   $$X_c = X - \overline{X},$$

   где X — исходная матрица данных размерности $n \times m$ (n — количество объектов, m — количество признаков), а $\overline{X}$ — вектор выборочных средних значений по каждому признаку, вычисленных по столбцам матрицы X. Таким образом, $\overline{X}$ представляет собой вектор размерности $1 \times m$, где каждый элемент соответствует среднему значению для конкретного признака.

2. **Вычисление матрицы ковариаций:**
   Матрица ковариаций отражает взаимосвязь между признаками. Она является симметричной и положительно определённой.
   Формула:

$$\Sigma = \frac{1}{n-1} X^T X,$$

где X — центрированная матрица данных размерности $n \times m$, а n — количество объектов.
Примечание: Мы используем несмещённую ковариацию[1], чтобы обеспечить корректное статистическое описание данных.

3. **Нахождение собственных значений и векторов:**
Для матрицы ковариаций Σ выполняются следующие утверждения:

   - Все собственные значения вещественные и неотрицательные.
   - Собственные векторы образуют ортонормированный базис.
   - Каждое собственное значение связано с соответствующим направлением максимальной дисперсии данных.

   В рамках данной лабораторной работы вам предстоит написать собственный метод ‘eigen-solver‘, чтобы найти спектр матрицы Σ и построить собственный базис.

4. **Проекция данных:**
После нахождения собственных векторов, данные проецируются на первые k главных компонент, сохраняя наибольшую часть информации.
Формула:

$$X_{\text{proj}} = X V_k,$$

где V_k — матрица первых k собственных векторов размерности $m \times k$. Каждый столбец матрицы V_k представляет собой нормированный собственный вектор матрицы ковариаций Σ, соответствующий одному из наибольших собственных значений.

Чтобы узнать, какую долю данных мы сохранили с помощью k главных компонент, введём долю объяснённой дисперсии:

$$\gamma = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i}.$$

[1] Несмещённая ковариация вычисляется как $\frac{1}{n-1}$ для того, чтобы её математическое ожидание совпадало с теоретической ковариацией.

Математическое обоснование

Задание (Expert Level): Доказать, что оптимальные направления PCA совпадают с собственными векторами матрицы ковариаций.

Практическая часть

Итак, давайте ближе к делу. В прошлом семестре вы уже реализовали собственный класс для векторной алгебры и написали функцию для поиска корня на отрезке. В этот раз вам придётся чуть-чуть их доработать и переиспользовать.

Easy level

Задание (Easy): Реализовать метод Гаусса для решения СЛАУ:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Прототип функции:

```python
from typing import List

def gauss_solver(A: 'Matrix', b: 'Matrix') -> List['Matrix']:
    """
    Вход:
        A: матрица коэффициентов (n×n). Используется класс Matrix из предыдущей
        ↪  лабораторной работы
        b: вектор правых частей (n×1)
    Выход:
        list[Matrix]: список базисных векторов решения системы
    Raises:
        ValueError: если система несовместна
    """
    pass
```

Задание (Easy): Реализовать функцию центрирования данных:

$$X_{\text{centered}} = X - \text{mean}(X).$$

Прототип функции:

```
def center_data(X: 'Matrix') -> 'Matrix':
    """
    Вход: матрица данных X (n×m)
    Выход: центрированная матрица X_centered (n×m)
    """
    pass
```

Задание (Easy): Вычислить матрицу ковариаций:

$$C = \frac{1}{n-1} X^T X.$$

Прототип функции:

```
def covariance_matrix(X_centered: 'Matrix') -> 'Matrix':
    """
    Вход: центрированная матрица X_centered (n×m)
    Выход: матрица ковариаций C (m×m)
    """
    pass
```

Normal Level

Задание (Normal): Найти собственные значения матрицы C методом бисекции:

$$\det(C - \lambda I) = 0.$$

Прототип функции:

```
def find_eigenvalues(C: 'Matrix', tol: float = 1e-6) -> List[float]:
    """
    Вход:
        C: матрица ковариаций (m×m)
        tol: допустимая погрешность
    Выход: список вещественных собственных значений
    """
    pass
```

Задание (Normal): Найти собственные векторы матрицы C:

$$(C - \lambda I)v = 0.$$

Прототип функции:

```
def find_eigenvectors(C: 'Matrix', eigenvalues: List[float]) -> List['Matrix']:
    """
    Вход:
        C: матрица ковариаций (m×m)
        eigenvalues: список собственных значений
    Выход: список собственных векторов (каждый вектор - объект Matrix)
    """
    pass
```

Задание (Normal): Вычислить долю объяснённой дисперсии:

$$\gamma = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i}.$$

Прототип функции:

```
def explained_variance_ratio(eigenvalues: List[float], k: int) -> float:
    """
    Вход:
        eigenvalues: список собственных значений
        k: число компонент
    Выход: доля объяснённой дисперсии
    """
    pass
```

Hard Level

Задание (Hard): Реализовать полный алгоритм PCA:

1. Центрирование данных.
2. Вычисление матрицы выборочных ковариаций.
3. Нахождение собственных значений и векторов.
4. Проекция данных на главные компоненты.

Прототип функции:

```
def pca(X: 'Matrix', k: int) -> Tuple['Matrix', float]:
    """
    Вход:
        X: матрица данных (n×m)
        k: число главных компонент
    Выход:
        X_proj: проекция данных (n×k)
        : доля объяснённой дисперсии
    """
    pass
```

Задание (Hard): Визуализировать проекцию данных на первые две главные компоненты. **Задание является единственным исключением из правил, в котором можно пользоваться сторонними модулями для построения графиков. Рекомендуется Matplotlib.**
Прототип функции:

```
from matplotlib.figure import Figure

def plot_pca_projection(X_proj: 'Matrix') -> Figure:
    """
    Вход: проекция данных X_proj (n×2)
    Выход: объект Figure из Matplotlib
    """
    pass
```

Задание (Hard): Вычислить среднеквадратическую ошибку восстановления данных:

$$\text{MSE} = \frac{1}{n \cdot m} \sum_{i,j} (X_{\text{orig}} - X_{\text{recon}})^2.$$

Прототип функции:

```
def reconstruction_error(X_orig: 'Matrix', X_recon: 'Matrix') -> float:
    """
    Вход:
        X_orig: исходные данные (n×m)
        X_recon: восстановленные данные (n×m)
    Выход: среднеквадратическая ошибка MSE
    """
    pass
```

Expert Level

Задание (Expert): Добавить автоматический выбор числа главных компонент на основе порога объяснённой дисперсии (встроить это в реализованную функцию pca):

$$k = \min \left\{ k : \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i} \geq \text{threshold} \right\}.$$

Прототип функции:

```
def auto_select_k(eigenvalues: List[float], threshold: float = 0.95) -> int:
    """
    Вход:
        eigenvalues: список собственных значений
        threshold: порог объяснённой дисперсии
    Выход: оптимальное число главных компонент k
    """
    pass
```

Задание (Expert): Обработать пропущенные значения в данных:

$$X_{\text{filled}} = \begin{cases} X_{ij}, & \text{если } X_{ij} \neq \text{NaN}, \\ \text{mean}(X_j), & \text{иначе.} \end{cases}$$

Прототип функции:

```
def handle_missing_values(X: 'Matrix') -> 'Matrix':
    """
    Вход: матрица данных X (n×m) с возможными NaN
    Выход: матрица данных X_filled (n×m) без NaN
    """
    pass
```

Задание (Expert): Исследовать влияние шума на PCA:

- Добавить случайный шум к данным.
- **Сравнить**[a] результаты PCA до и после добавления шума.

Прототип функции:

```
def add_noise_and_compare(X: 'Matrix', noise_level: float = 0.1):
    """
    Вход:
        X: матрица данных (n×m)
        noise_level: уровень шума (доля от стандартного отклонения)
    Выход: результаты PCA до и после добавления шума.
    В этом задании можете проявить творческие способности, поэтому выходные данные не
    ↪  типизированы.
    """
    pass
```

[a] Придётся порассуждать и проанализировать результаты

Задание (Expert): Применить PCA к реальному датасету:

- Загрузить данные[a].
- Сравнить метрики качества до и после снижения размерности.

Прототип функции:

```
def apply_pca_to_dataset(dataset_name: str, k: int) -> Tuple['Matrix', float]:
    """
    Вход:
        dataset_name: название датасета
        k: число главных компонент
    Выход: кортеж (проекция данных, качество модели)
    """
    pass
```

[a] Например, из открытых датасетов Kaggle: `https://www.kaggle.com/datasets`

Правила выполнения работы

1. **Теоретические задания** требуют развёрнутых ответов с математическими выкладками и пояснениями. Односложные ответы («да/нет», «возможно») считаются некорректными.
2. **Запрещено использовать сторонние библиотеки**, не входящие в стандартную поставку Python[2]. Разрешены только:
   - `math`, `random`, `collections`
   - `matplotlib` (для визуализации)
   - `unittest` (для тестирования)
3. Все численные методы должны быть реализованы самостоятельно. Использование `numpy.linalg` или аналогов запрещено.
4. Для работы с данными используйте CSV-файлы или встроенные датасеты `sklearn.datasets` (только для загрузки данных!).

Рекомендуемая литература

1. **Математическая статистика**
   Бородин А.Н. — Элементарный курс теории вероятностей и математической статистики
   Casella G., Berger R. — Statistical Inference
2. **Математическое моделирование и вычислительная математика**
   Воробьёва Г.Н., Данилова А.Н. — Практикум по вычислительной математике
   Устинов С.М., Зимницкий В.А. — Вычислительная математика: алгоритмы и реализация
3. **Статистическое машинное обучение**
   James G., Witten D., Hastie T., Tibshirani R. — An Introduction to Statistical Learning
   Hastie T., Tibshirani R., Friedman J. — The Elements of Statistical Learning
4. **Документация и практическая реализация**
   Официальная документация Python: `https://docs.python.org/3/`

[2] Список стандартных библиотек: `https://docs.python.org/3/library/`