Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
Университет ИТМО
Факультет технологий искусственного интеллекта

ОТЧЕТ

по лабораторной работе «PCA»

по дисциплине
«Линейная алгебра и обработка данных»

Выполнил:
Клемяционок П.А., J3113

Преподаватель:
Москаленко М.А.

Санкт-Петербург, 2025

Содержание

Введение

Метод главных компонент (Principal Component Analysis, PCA) представляет собой один из базовых методов анализа и понижения размерности данных. В его основе лежит преобразование исходных переменных в новую ортогональную систему координат, упорядоченную по убыванию дисперсии. В данной лабораторной работе реализован полный цикл вычислений вручную: от решения систем линейных уравнений и нахождения собственных значений до получения проекций, построения графиков и экспериментов с реальными и зашумлёнными данными.

Целью данной лабораторной работы является освоение метода главных компонент (PCA) и реализация всех этапов анализа вручную, без использования специализированных библиотек машинного обучения. В процессе работы ставились следующие задачи:

- Реализовать алгоритм решения систем линейных уравнений методом Гаусса с возможностью вывода общего решения в векторной форме.
- Выполнить центрирование матрицы и построить ковариационную матрицу.
- Найти собственные значения и соответствующие собственные векторы без использования NumPy.
- Реализовать функциональность PCA: понижение размерности, визуализация и восстановление исходных данных.
- Добавить поддержку работы с шумом и встроенными датасетами.
- Разработать систему автотестов для всех функций.

Часть I

Уровень: Easy

1 Метод Гаусса (Решение СЛАУ)

1.1 Теоретическое обоснование

Метод Гаусса — классический алгоритм решения систем линейных алгебраических уравнений (СЛАУ).

Цель метода — приведение расширенной матрицы системы к ступенчатому виду (треугольной форме) с последующим обратным ходом, позволяющим выразить каждую переменную. Метод применим как для совместных, так и для несовместных и вырожденных систем.

Для получения общего решения используется понятие свободных и ведущих переменных, а также параметризация через свободные переменные.

1.2 Реализованные функции (модуль `gauss.py`)

- `gauss_solver_with_general_solution(A, b)` — основная функция решения СЛАУ методом Гаусса с поддержкой общего решения.
- `remove_linearly_dependent_rows_strict(matrix)` — удаляет линейно зависимые строки и проверяет совместность по правой части.
- `print_solution_as_latex(A, b)` — вывод решения в виде векторной формы (ФСР) с параметрами.

Описание алгоритма

Функция `gauss_solver_with_general_solution` реализует классическую процедуру:

1. Преобразование расширенной матрицы $(A|b)$.
2. Поиск ведущего элемента в каждом столбце.
3. Деление строки на ведущий элемент.
4. Обнуление элементов ниже и выше ведущего с помощью вычитания строк.
5. Подсчёт ранга и формирование общего решения.

1.3 Особенности и ограничения

- Реализация не использует внешние библиотеки, работает на списках Python.
- Обработка вырожденных систем (нулевые строки) и несовместных случаев предусмотрена.

- Автоматическое удаление линейно зависимых строк.
- Поддержка бесконечного множества решений через параметризацию.

Возникшие сложности

Я захотела сделать вывод решения в виде векторов, как в математических книжках: чтобы выводились базисные векторы ФСР , а не просто текст вида $x_1 = a - 2b$. Сначала я написала просто построчную печать, потом пыталась парсить это в LaTeX. Долго не получалось, потому что:

- нужно было отдельно выделить свободные и главные переменные;
- я забыла учесть случаи, когда правые части не совпадают у пропорциональных строк — возникала ошибка;
- сложно оказалось отладить нормальный формат вывода векторов (нужно было обрабатывать и нули, и знаки, и порядок);

Также написала отдельную функцию `remove_linearly_dependent_rows_strict` — она проверяет, нет ли одинаковых строк с разными правыми частями, и выводит сообщение о несовместности. Это оказалось важным, потому что без этой проверки метод Гаусса может вернуть странный результат.

2 Центрирование данных

2.1 Теоретическое обоснование

Центрирование данных — это первый шаг метода главных компонент (PCA). Оно заключается в вычитании среднего значения из каждого столбца матрицы признаков, чтобы каждый признак имел нулевое математическое ожидание.

Центрирование необходимо для корректного вычисления ковариационной матрицы: смещения в данных могут искажать направление главных компонент.

$$X_{ij}^{\text{new}} = X_{ij} - \bar{X}_j$$

где $\bar{X}_j$ — среднее значение j-го признака по всем наблюдениям.

2.2 Реализованные функции (модуль `center.py`)

- `center_data(X)` — принимает матрицу признаков X (объекты $\times$ признаки), возвращает центрированную матрицу X_{centered}.

Описание алгоритма

Функция вычисляет среднее значение по каждому столбцу (признаку), а затем из каждого элемента соответствующего столбца вычитает это среднее:

- Внешний цикл проходит по всем строкам (объектам).
- Внутренний цикл вычитает соответствующее среднее по каждому столбцу.
- Возвращается новая матрица, содержащая центрированные значения.

2.3 Особенности и ограничения

- Ожидается, что входная матрица имеет хотя бы одну строку и один столбец.
- Не производится нормализация (деление на стандартное отклонение).
- Центрирование применяется ко всем признакам одинаково.

Возникшие сложности

- Задумалась: а если пустая матрица? Поэтому добавила проверку на `n == 0`.
- Визуально проверяла, что среднее по каждому столбцу стало около нуля.

3 Ковариационная матрица

3.1 Теоретическое обоснование

Ковариационная матрица описывает взаимосвязи между признаками после центрирования данных. Элемент на позиции (i, j) показывает ковариацию между признаками i и j:

$$C_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} (X_{ki} - \bar{X}_i)(X_{kj} - \bar{X}_j)$$

где n — число объектов (строк), X_{ki} — значение i-го признака для k-го объекта.

Ковариационная матрица является симметричной и используется для вычисления собственных значений и векторов в методе PCA.

3.2 Реализованные функции (модуль `covariance.py`)

- `covariance_matrix(X_centered)` — вычисляет ковариационную матрицу по центрированным данным.

Описание алгоритма

- Выполняется транспонирование матрицы признаков.
- Далее происходит умножение $X^T \cdot X$, где X — центрированная матрица.
- После этого каждый элемент делится на $(n-1)$, где n — число объектов.
- Результат — ковариационная матрица размерности $(m \times m)$, где m — число признаков.

3.3 Особенности и ограничения

- Ожидается, что на вход поступает центрированная матрица.
- Для стабильной работы размерность входной матрицы должна быть $n \geq 2$.
- Результат используется как вход для поиска собственных значений.

Возникшие сложности

Вручную написала транспонирование (через двойной цикл), потом перемножение.

- Сначала случайно транспонировала неправильно (путала строки со столбцами).
- Потом забыла поделить на $n - 1$, и ковариации получались очень большими.

Часть II

Уровень Normal

4 Собственные значения

4.1 Теоретическое обоснование

Собственные значения (λ) ковариационной матрицы показывают, какую долю дисперсии в данных объясняет каждая главная компонента. Чем больше значение λ, тем больше "вклада"в общее распределение данных в этом направлении.

Для нахождения λ решается характеристическое уравнение:

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

Где C — ковариационная матрица, I — единичная матрица.

4.2 Реализованная функция (модуль `eigen.py`)

- `find_eigenvalues(C)` — вычисляет собственные значения ковариационной матрицы.

Описание алгоритма

- Используется численный метод поиска корней (например, метод бисекции или перебор с шагом).
- Вычисляется определитель $\det(C - \lambda I)$ при различных значениях λ.
- Отбираются те значения, при которых определитель близок к нулю.

4.3 Особенности и ограничения

- Значения считаются приближённо, с точностью до $1e{-}6$.
- Используется сортировка и фильтрация близких значений.
- Возможна потеря точности при плохой обусловленности матрицы.

Возникшие сложности

- Самое сложное — реализовать поиск корней уравнения $det(C - \lambda I) = 0$, особенно через метод бисекции
- Нужно было аккуратно реализовать определитель и не забыть удалять повторяющиеся корни с близкими значениями, иначе находились одинаковые λ

- При большом числе признаков функция начинает работать очень долго, потому что определитель считается рекурсивно — об этом тоже пришлось подумать

5 Собственные векторы

5.1 Теоретическое обоснование

Собственные векторы (v) — направления главных компонент. Каждому собственному значению λ соответствует собственный вектор v, удовлетворяющий уравнению:

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

Собственные векторы задают новое ортогональное пространство, в котором происходит проекция исходных данных.

5.2 Реализованная функция (модуль `eigenvectors.py`)

- `find_eigenvectors(C, eigenvalues)` — находит собственные векторы для каждого из λ.

Описание алгоритма

- Для каждого найденного собственного значения формируется матрица $(C - \lambda I)$.
- Решается соответствующая однородная система линейных уравнений методом Гаусса.
- Формируется базис решения или выражение через свободные переменные.

5.3 Особенности и ограничения

- Реализация не использует внешние библиотеки.
- Если система имеет только нулевое решение, вектор не возвращается.
- Выводится предупреждение, если собственный вектор найти невозможно.

Возникшие сложности

- Изначально не получалось подставить λ в матрицу $C - \lambda I$, потом исправила
- Сложно было правильно разобрать результат, возвращаемый $\texttt{gauss}_s olver_w ith_g eneral_s olut$

6 Объяснённая дисперсия

6.1 Теоретическое обоснование

Объяснённая (или накопленная) дисперсия — это доля общей дисперсии данных, сохраняемая при проецировании на k главных компонент. Она используется для выбора оптимального количества компонент в PCA.

Вычисляется по формуле:

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

где λ_i — собственные значения ковариационной матрицы, m — общее число компонент.

6.2 Реализованная функция (модуль `explained.py`)

- `explained_variance_ratio(eigenvalues, k)` — вычисляет долю объяснённой дисперсии при использовании первых k компонент.

Описание алгоритма

- Собственные значения λ_i сортируются по убыванию.
- Складываются первые k значений и делятся на сумму всех значений.
- Результат — число от 0 до 1, показывающее, сколько информации сохраняется.

6.3 Особенности и ограничения

- Функция не проверяет наличие отрицательных значений λ.
- При $k = m$ результат всегда равен 1.
- Используется обычное деление и сортировка без внешних библиотек.

ВОзникшие сложности

- Забыла отсортировать собственные значения по убыванию перед подсчётом — это дало некорректные результаты
- Не сразу поняла, что k может превышать количество собственных значений, пришлось дополнительно это проверять

Часть III

Уровень Hard

7 Проекция данных (PCA)

7.1 Теоретическое обоснование

После нахождения собственных векторов ковариационной матрицы, PCA проецирует исходные данные в новое пространство — пространство главных компонент. Это уменьшает размерность данных, сохраняя при этом максимум дисперсии.

Проекция выполняется по формуле:

$$X_{\text{proj}} = X_{\text{centered}} \cdot W$$

где X_{centered} — центрированные данные, W — матрица, составленная из первых k собственных векторов.

7.2 Реализованная функция (модуль `pca.py`)

- `pca(X: Matrix, k: int)` — полный алгоритм PCA: центрирование, ковариация, поиск λ и v, проекция.

Описание алгоритма

- Центрируются данные.
- Вычисляется ковариационная матрица.
- Находятся собственные значения и векторы.
- Векторы сортируются по убыванию λ.
- Строится матрица W из k лучших векторов.
- Проецируются данные: $X_{\text{proj}} = X_{\text{centered}} \cdot W$.

7.3 Особенности и ограничения

- Метод не зависит от внешних библиотек.
- Не выполняется нормализация (масштабирование по дисперсии).
- При нехватке собственных векторов — выдаётся предупреждение.

8 Восстановление данных

8.1 Теоретическое обоснование

После проекции на пространство меньшей размерности, PCA позволяет восстановить приближение к исходным данным. Это важно для оценки качества сжатия.

Восстановление выполняется по формуле:

$$X_{\text{recon}} = X_{\text{proj}} \cdot W^T + \mu$$

где W^T — транспонированная матрица главных компонент, μ — вектор средних значений по признакам.

Реализованная функция (в `main.py` или `pca.py`)

- `reconstruct_data(X_proj, W, means)` — восстанавливает данные из проекции и матрицы компонент.

Описание алгоритма

- Транспонируется матрица W (получаем W^T).
- Вычисляется $X_{\text{proj}} \cdot W^T$.
- К результату прибавляется вектор средних значений по каждому столбцу.

8.2 Особенности и ограничения

- Ожидается, что количество компонент k соответствует числу столбцов в W.
- При отсутствии достаточного числа векторов восстановление не выполняется.
- Ошибка восстановления (MSE) может быть рассчитана отдельно.

8.3 Возникшие сложности

- Транспонирование матрицы W перед умножением оказалось критично — без него матричное умножение давало ошибку
- Средние значения по признакам (`means`) должны быть рассчитаны строго по исходной матрице A, а не по центрированной — с этим тоже поначалу была путаница.

9 Полный алгоритм PCA (интеграция модулей)

9.1 Описание этапа

В данном разделе объединяются все ранее реализованные этапы (центрирование, ковариационная матрица, собственные значения, собственные векторы, объяснённая дисперсия) в один общий алгоритм **Principal Component Analysis**

(PCA). Этот этап оформлен как отдельная функция, включающая вызовы всех предыдущих функций, а также добавлены две новые: проекция и восстановление данных.

Зависимости от предыдущих разделов (Easy, Normal)

Данный раздел использует:

- из Easy:
  - `gauss_solver_with_general_solution` — для решения СЛАУ при нахождении собственных векторов;
- из Normal:
  - `center_data` — центрирование данных;
  - `covariance_matrix` — построение ковариационной матрицы;
  - `find_eigenvalues` — нахождение собственных значений;
  - `find_eigenvectors` — нахождение собственных векторов;
  - `explained_variance_ratio` — вычисление объяснённой дисперсии.

Что делает `pca(X, k)`

1. Центрирует матрицу X.
2. Строит ковариационную матрицу.
3. Находит собственные значения и векторы.
4. Сортирует их по убыванию λ.
5. Формирует матрицу W из k лучших векторов.
6. Строит проекцию: $X_{\text{proj}} = X_{\text{centered}} \cdot W$.

9.2 Особенности и ограничения

- Если не удаётся найти достаточное количество собственных векторов, PCA или восстановление может быть пропущено.
- Все шаги выполняются вручную, без использования библиотек (NumPy и др.).
- Обработка исключений и некорректных входов встроена в алгоритм.

Возникшие сложности

Сложности:

- Очень важно было не перепутать порядок умножения матриц: $X_{\text{centered}}(n \times m)$ умножается на $W(m \times k)$, а не наоборот

- Когда $k = 1$, нужно было автоматически выбрать число компонент, объясняющих не менее 95% дисперсии. Поначалу об этом вообще забыли, потом добавили отдельную функцию $\texttt{auto}_s elect_k, ^-$.

10 Визуализация проекции данных

10.1 Теоретическое обоснование

После проекции данных в пространство главных компонент (обычно на первые две), становится возможным визуализировать структуру данных. Это помогает:

- Оценить кластеризацию или линейность признаков,
- Найти выбросы или аномалии,
- Проверить эффективность снижения размерности.

Для визуализации используется библиотека `Matplotlib`. Это является **единственным разрешённым сторонним модулем** в данной работе.

10.2 Реализованная функция (модуль `visualize.py`)

- `plot_pca_projection(X_proj)` — строит двумерную scatter-диаграмму проекции данных на первые две главные компоненты.

Описание алгоритма

- Принимается матрица X_{proj} размерности $n \times 2$ (объекты $\times$ 2 компоненты).
- Из каждой строки извлекается координата (x, y).
- Строится scatter-плот с помощью `matplotlib.pyplot`.
- Возвращается объект `Figure`, который затем можно отобразить.

10.3 Особенности и ограничения

- Визуализация возможна только при $k \geq 2$.
- Если компонент меньше 2 — выводится предупреждение.
- Используется только `matplotlib`, установка может потребовать дополнительных зависимостей.

ВОзникшие сложности

- На компьютере графики не отображались в окне, потому что по умолчанию matplotlib использовал неподдерживаемый backend. Пришлось вручную указать `TkAgg`
- Также была проблема: если количество компонент меньше 2, график невозможно построить — добавили проверку и вывод предупреждения
- Иногда график сохранялся в файл, но не открывался — после этого добавили вызов `fig.show()`

11 Оценка качества восстановления и автотесты

11.1 Теоретическое обоснование

Для количественной оценки потерь при снижении размерности используется среднеквадратическая ошибка восстановления (MSE):

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

где X_{orig} — исходная матрица данных, X_{recon} — восстановленная матрица после PCA и обратного преобразования.

Низкое значение MSE означает, что большая часть информации сохранена даже после снижения размерности.

11.2 Реализованная функция (модуль `reconstruction.py`)

- `reconstruction_error(X_orig, X_recon)` — вычисляет среднеквадратическую ошибку восстановления.

Описание алгоритма

- Итерируется по всем элементам матрицы X.
- Считается разность квадратов между оригинальными и восстановленными значениями.
- Делится на общее количество элементов.

11.3 Особенности и ограничения

- Работает только с матрицами одинаковой размерности.
- Используется стандартное евклидово расстояние.

Система автотестов

Для проверки корректности каждой реализованной части были написаны и выполнены автотесты:

- Решение СЛАУ (метод Гаусса)
- Центрирование данных
- Построение ковариационной матрицы
- Собственные значения и векторы
- Объяснённая дисперсия
- PCA и восстановление

Тесты покрывают как корректные, так и граничные случаи: нулевые строки, пропорциональные строки, нулевая дисперсия, одинаковые строки, несовместные системы и т.д.

Часть IV

Уровень Expert

12 Почему оптимальные направления PCA совпадают с собственными векторами ковариационной матрицы

Постановка задачи

Метод главных компонент (PCA) используется для уменьшения размерности данных. Он позволяет найти новые оси (направления), на которые можно проецировать данные так, чтобы при этом сохранялось как можно больше информации (в терминах дисперсии).

Пусть у нас есть матрица данных $X \in \mathbb{R}^{n \times m}$, где n — количество наблюдений (объектов), а m — количество признаков (переменных).

Предположим, что каждый признак уже центрирован: среднее значение каждого столбца равно нулю:

$$\frac{1}{n}\sum_{i=1}^{n} X_{ij} = 0, \quad \text{для всех } j.$$

Цель PCA

Нужно найти такое направление $w \in \mathbb{R}^m$, чтобы проекция всех точек данных на это направление обладала наибольшей возможной дисперсией.

Проекцией строки x_i матрицы X на направление w является скаляр $x_i^T w$.

Тогда вся матрица проекций — это вектор $Xw \in \mathbb{R}^n$.

Что хотим максимизировать:

$$\text{maximize} \quad \text{Var}(Xw).$$

Но чтобы избежать тривиального решения (например, $w = 1000 \cdot u$, тогда дисперсия тоже увеличится), накладываем ограничение:

$$\|w\|^2 = 1.$$

Как выражается дисперсия проекций

Дисперсия вектора Xw определяется как:

$$\text{Var}(Xw) = \frac{1}{n}\sum_{i=1}^{n}(x_i^T w)^2 = \frac{1}{n}\|Xw\|^2.$$

Переводим в матричную форму:

$$\|Xw\|^2 = (Xw)^T(Xw) = w^T X^T X w.$$

Обозначим ковариационную матрицу:

$$C := \frac{1}{n} X^T X \in \mathbb{R}^{m \times m}.$$

Тогда:

$$\mathrm{Var}(Xw) = w^T C w.$$

Задача сводится к:

$$\begin{cases} \text{maximize } w^T C w, \\ \text{subject to } w^T w = 1. \end{cases}$$

Это задача **максимизации квадратичной формы на единичной сфере** — классическая задача из линейной алгебры.

Решение через метод Лагранжа

Переходим к методу множителей Лагранжа:

$$\mathcal{L}(w, \lambda) = w^T C w - \lambda(w^T w - 1).$$

Берём градиент по w:

$$\nabla_w \mathcal{L} = 2Cw - 2\lambda w = 0 \quad \Rightarrow \quad Cw = \lambda w.$$

Это и есть определение собственного вектора: w — собственный вектор матрицы C, а λ — соответствующее собственное значение.

Интерпретация результата

Значение целевой функции в этом случае:

$$w^T C w = \lambda.$$

То есть: дисперсия проекций вдоль направления w равна λ, если w — собственный вектор C.

Значит, чтобы дисперсия была максимальной, нужно выбрать направление w, соответствующее **наибольшему собственному значению** ковариационной матрицы C.

Выбор следующих компонент

После нахождения первого главного направления w_1, ищем следующее направление w_2, которое:

- тоже максимизирует дисперсию,
- и ортогонально предыдущим $w_1, w_2, \ldots$.

По теореме о спектральном разложении симметрических матриц (а C — симметричная ковариационная матрица), все собственные векторы можно выбрать ортонормированными. Поэтому следующий компонент PCA — это просто следующий по величине собственный вектор C, и так далее.

Итоговое утверждение

Оптимальные направления для PCA — это собственные векторы ковариационной матрицы C. Первый компонент соответствует наибольшему собственному значению, второй — второму по величине и т.д.