Санкт-Петербургский государственный университет
Математика и компьютерные науки

Отчёт по учебной практике 2 (научно-исследовательская работа) (4 семестр)

Реализация алгоритма "наилучшего" разделения выпуклых оболочек двух множеств

Выполнила:
Здрогова Д.О., группа 21.Б13-мм

Научный руководитель:
д. т. н., профессор
Фрадков А.Л.
Кафедра теоретической кибернетики

Санкт-Петербург
2023

Оглавление

Введение

Важность алгоритма разделения выпуклых оболочек двух множеств заключается в его применимости и значимости для различных областей и задач. В компьютерном зрении, например, точное разделение объектов на изображениях может быть критически важным для распознавания и классификации объектов.

Также разделение выпуклых оболочек двух множеств является актуальной задачей в области вычислительной геометрии. Выпуклая оболочка представляет собой минимальную выпуклую фигуру, охватывающую заданное множество точек. Определение наилучшего разделения выпуклых оболочек заключается в поиске границы, которая наиболее эффективно разделяет два множества точек.

Цель данного исследования заключается в реализации алгоритма, который позволит линейно разделять два множества точек. В данной курсовой работе будет представлен подробный обзор такого алгоритма[1] и его реализация и тестирование на различных наборах данных.

1. Постановка задачи

Входные данные:

- Даны два множества точек: $X^{(1)}$ и $X^{(2)}$, представляющие два класса точек в n-мерном пространстве;
- Множество $X^{(1)}$ содержит точки из первого класса;
- Множество $X^{(2)}$ содержит точки из второго класса;
- Каждая точка представлена n координатами $(x_1, ..., x_n)$.

Требования:

- Необходимо найти гиперплоскость, которая наилучшим образом разделяет две выпуклые оболочки классов $X^{(1)}$ и $X^{(2)}$;
- Гиперплоскость представляет собой (n-1)-мерное подпространство в n-мерном пространстве и разделяет множества точек таким образом, что точки одного множества находятся по одну сторону границы, а точки другого множества - по другую сторону;
- Гиперплоскость должна быть определена уравнением $a_1x_1 + ... + a_nx_n + c = 0$;
- Если выпуклые оболочки множеств пересекаются, то алгоритм должен указать на отсутствие искомой гиперплоскости.

Требуется реализовать алгоритм, который найдет гиперплоскость наилучшего разделения выпуклых оболочек классов $X^{(1)}$ и $X^{(2)}$ в n-мерном пространстве, удовлетворяющую требованиям вышеуказанной постановки задачи, или укажет на её отсутствие.

2. Описание алгоритма

2.1. Геометрический смысл алгоритма

Пусть $z_1, z_2, ...$ - элементы тренировочного множества. Пусть $x_1(y_1)$ - первый элемент тренировочного множества, принадлежащий классу $X^{(1)}(X^{(2)})$. В качестве начального приближения к точкам x_0, y_0, реализующим расстояние между выпуклыми оболочками классов $X^{(1)}$ и $X^{(2)}$, выбираются точки x_1, y_1.

Пусть на n-ном шаге алгоритма получены точки x_n, y_n, аппроксимирующие собой точки x_0, y_0, и пусть появилось изображение z_n. Предположим для определённости, что $z_n \in X^{(1)}$. Тогда $y_{n+1} = y_n$, а в качестве x_{n+1} выбирается точка, реализующая расстояние от точки y_n до отрезка, соединяющего точки x_n и z_n.

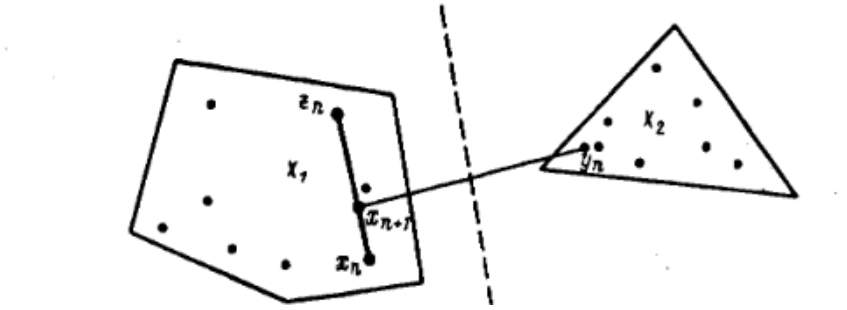


Рис. 1: Геометрическая интерпретация алгоритма [1]

Если $z_n \in X^{(2)}$, то $x_{n+1} = x_n$, а в качестве y_{n+1} выбирается точка, на которой реализуется наименьшее расстояние, то точки x_n до отрезка, соединяющего точки y_n и z_n.

2.2. Описание работы алгоритма "Оптимальное разделение"

Расстояние от точки до заданного отрезка нетрудно вычислить аналитически. Действительно, пусть требуется найти расстояние от заданной точки y_n до отрезка прямой, соединяющего заданные точки

x_n и z_n, $x_n \neq z_n$. Это можно записать следующим образом $\rho^2(\mu) = ||y_n - x_n + \mu(x_n - z_n)||^2$ при изменении μ в интервале $[0, 1]$. Наименьшее значение на всей оси функция $\rho^2(\mu)$ принимает в точке

$$\mu_n^{min} = ||x_n - z_n||^{-2}(x_n - y_n, x_n - z_n) \tag{1}$$

Следовательно, точка $x_n^{'}$, реализующая наименьшее расстояние от точки y_n до отрезка, соединяющего точки x_n и z_n имеет вид

$$x_n^{'} = x_n + \mu_n(z_n - x_n), \text{ где}$$
$$\mu_n = \begin{cases} 0 & \text{если } \mu_n^{min} \leq 0, \\ \mu_n^{min} & \text{если } 0 < \mu_n^{min} \leq 1, \\ 1 & \text{если } \mu_n^{min} > 0. \end{cases} \tag{2}$$

Сам алгоритм будет таким: пусть x_n, y_n - точки, аппроксимирующие после n шагов алгоритма точки x_0, y_0 и $z_n \in X^{(1)}$. Тогда

$$y_{n+1} = y_n, x_{n+1} = x_n + \mu_n(z_n - x_n), \tag{3}$$

где значения μ_n определяются формулами (1) - (2).

Если же $z_n \in X^{(2)}$, то $x_{n+1} = x_n$, а y_{n+1} получится по формулам (3) после замены в них x_n и y_n на y_n и x_n соответственно.

2.3. Конечная сходимость алгоритма

Строго говоря, алгоритм (1) - (3) не является конечно сходящимся. Причина состоит в том, что при больших номерах n точка x_n может мало изменяться. Чтобы сделать алгоритм (1) - (3) конечно-сходящимся, следует при предъявлении точки $z_n \in X^{(1)}$ ($z_n \in X^{(2)}$) не сразу изменять точку x_n на x_{n+1} (y_n на y_{n+1}), а только в том случае, если величина $||x_{n+1} - x_n||$ ($||y_{n+1} - y_n||$) изменится при этом не меньше, чем на некоторое положительное число δ - параметр программы. Алгоритм (1) - (2) в развёрнутой форме запишется в виде (опять полагаем, что $z_n \in X^{(1)}$)

$$x_{n+1} = \begin{cases} x_n, \text{ если } (y_n - x_n, z_n - x_n) \geq ||x_n - z_n||^2 \text{ и} \\ \qquad ||z_n - y_n||^2 + \delta^2 \geq ||x_n - y_n||^2, \\ \qquad \text{либо } (y_n - x_n, z_n - x_n) \leq \delta ||z_n - x_n||, \\ z_n, \text{ если } (y_n - x_n, z_n - x_n) > ||x_n - z_n||^2 \text{ и} \\ \qquad ||z_n - y_n||^2 + \delta^2 \leq ||x_n - y_n||^2, \\ x_n + \frac{(y_n - x_n, z_n - x_n)}{||z_n - x_n||^2}(z_n - x_n), \text{ если} \\ \qquad \delta ||z_n - x_n||^2 < (y_n - x_n, z_n - x_n) \leq ||z_n - x_n||^2. \end{cases} \tag{4}$$

Доказательство конечной сходимости алгоритма (4) теперь тривиально. Действительно, при каждом изменении точки x_n или y_n расстояние между ними убывает на величину, не меньшую, чем δ и, следовательно, через конечное число шагов будет исчерпана величина $||x_1 - y_1||$.

Также без доказательства отметим, что при достаточно малых значениях параметра δ в алгоритме (4) построенные точки $x_\infty = \lim\limits_{n\to\infty} x_n$, $y_\infty = \lim\limits_{n\to\infty} y_n$ дают достаточно хорошую аппроксимацию точек $x_0^{'}$, $y_0^{'}$, реализующих наименьшее расстояние между выпуклыми оболочками множеств $X^{(1)}$ и $X^{(2)}$. Заметим, что сами точки $x_0^{'}$ и $y_0^{'}$ могут быть далёкими соответственно от точек x_∞, y_∞.

Априорное предположение о существовании разделяющей плоскости здесь не является необходимым. После окончания работы алгоритма может оказаться, что $||x_\infty - y_\infty|| > \delta$, и это будет означать, что выпуклые оболочки классов $X^{(1)}$ и $X^{(2)}$ не пересекаются. Если классы $X^{(1)}$ и $X^{(2)}$ пересекаются либо расстояние между ними достаточно мало, то получим $||x_\infty - y_\infty|| < \delta$. Таким образом, алгоритм позволяет найти разделяющую плоскость или указать на её отсутствие.

3. Реализация алгоритма

Алгоритм "оптимального" разделения был реализован на языке C++. Он включает в себя следующие элементы:

- Класс $Vector$: Этот класс является шаблоном с фиксированным размером N и представляет вектор в N-мерном пространстве. Он содержит массив $c[N]$ для хранения компонент вектора. Класс также имеет метод $squared\_norm()$, который вычисляет квадрат нормы вектора.

```
template <size_t N>
class Vector {
public:
    long double c[N];
    Vector () = default;

    long double squared_norm () {
        long double sum = 0;
        for (int i = 0; i < N; ++i) {
            sum += c[i]*c[i];
        }
        return sum;
    }
};
```

- Перегруженные операторы: Код включает перегруженные операторы для операций вычитания (-), сложения (+) и умножения на скаляр (*). Эти операторы позволяют выполнять арифметические операции с векторами.

```
template <size_t N>
Vector<N> operator - (Vector<N> v1, Vector<N> v2) {
    Vector<N> v3;
    for (int i = 0; i < N; ++i)
        v3.c[i] = v1.c[i] - v2.c[i];
    return v3;
}

```

```
template <size_t N>
Vector<N> operator + (Vector<N> v1, Vector<N> v2) {
    Vector<N> v3;
    for (int i = 0; i < N; ++i)
        v3.c[i] = v2.c[i] + v1.c[i];
    return v3;
}

template <size_t N>
Vector<N> operator * (long double n, Vector<N> v) {
    Vector<N> v3;
    for (int i = 0; i < N; ++i)
        v3.c[i] = n * v.c[i];
    return v3;
}
```

- Функция $dot\_product()$: Эта функция вычисляет скалярное произведение двух векторов.

```
template <size_t N>
long double dot_product (Vector<N> v1, Vector<N> v2) {
    long double product = 0;
    for (int i = 0; i < N; ++i)
        product += v1.c[i] * v2.c[i];
    return product;
}
```

- Функция $new\_point()$: Эта функция принимает три вектора (x, y, z) и возвращает новую точку new_x. Она выполняет определенные проверки и вычисления для определения нового значения точки new_x.

```
template <size_t N>
Vector<N> new_point(Vector<N> x, Vector<N> y, Vector<N> z) {
    long double product = dot_product(y - x, z - x);
    Vector<N> new_x = x;
    if ((product ≥ (x - z).squared_norm() && (z - y).squared_norm() +
    DELTA*DELTA ≥ (x - y).squared_norm()) ||
    product ≤ DELTA*sqrt((z-x).squared_norm()))
        new_x = x;
```

```
    if (product > (z - x).squared_norm() && (z - y).squared_norm() +
    DELTA*DELTA ≤ (x - y).squared_norm())
        new_x = z;
    if (DELTA*(z - x).squared_norm() < product && product ≤ (z - x).
       squared_norm())
        new_x = x + ((product / (z - x).squared_norm()) * (z - x));
    return new_x;
}
```

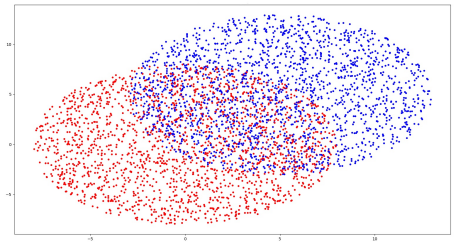
После окончания работы алгоритма, в качестве выходных данным получены точки $x = x_n$ и $y = y_n$. В вектор $norm$ записываются коэффициенты $(a_1, ..., a_n)$ уравнения гиперплоскости $X^{(1)}$ и $X^{(2)}$. Число -c является параметром сдвига.

```
Vector<N> norm = x - y;
Vector<N> mid = 0.5*(x + y)
long double c = 0;
for (int i = 0; i < N; ++i) {
    c += norm.c[i]*mid.c[i];
}
```

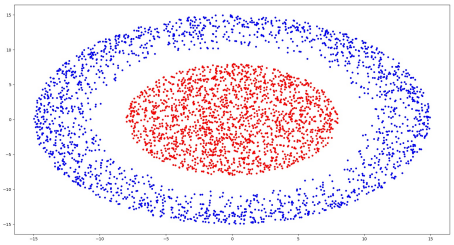
4. Тестирование

4.1. Тестирование на плоских данных

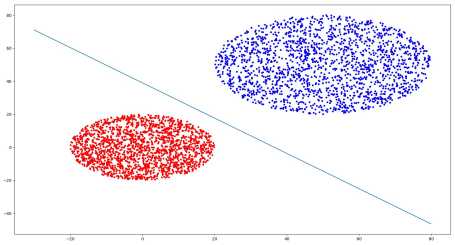
Для тестирования были сгенерированы плоские данные, которые подаются на вход алгоритму. Алгоритм находит прямую, разделяющую эти данные или указывает на отсутствие такой прямой. На графиках показаны примеры работы алгоритма на тестах разного типа.



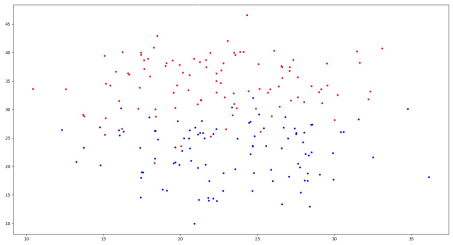
(a) Выпуклые оболочки пересекаются



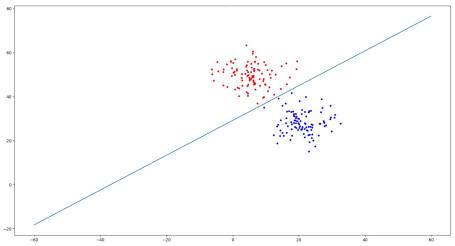
(b) Выпуклые оболочки пересекаются



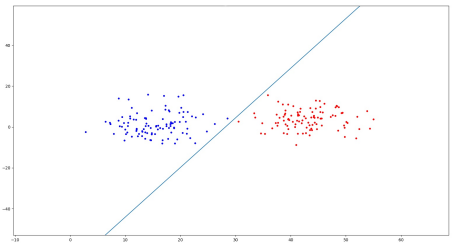
(c) Существует разделяющая прямая



(d) Выпуклые оболочки пересекаются



(e) Существует разделяющая прямая



(f) Существует разделяющая прямая

4.2. Тестирование на данных произвольной размерности

Таблица 1: Точность предсказания в зависимости от размерности (N) и степени разделения данных

N	Степень разделения	Точность предсказания
2	Высокая	0.992
3	Средняя (классы сильно наезжают)	Гиперплоскость не найдена
4	Высокая	1.0
5	Низкая	Гиперплоскость не найдена
6	Средняя (классы немного наезжают)	0.747
7	Низкая	Гиперплоскость не найдена

Поскольку алгоритм изначально предполагает работу только с данными, которые линейно разделимы, он справляется со своей задачей в случаях, когда данные могут быть разделены гиперплоскостью. В таких случаях алгоритм обеспечивает высокую точность предсказания и может успешно разделять классы.

4.3. Тестирование на реальных данных электроэнцефалограмм

Материалы и методы исследования:

- В исследовании приняли участие 12 женщин в возрасте от 18 до 23 лет, с нормальным слухом и нормальным или скорректированным до нормального зрением;
- Испытуемые должны были совершить комплексное самоинициированное движение, состоящее из двух этапов: нажатие на кнопку и касание маркера в обход прозрачной перегородки;
- Задание выполнялось указательным пальцем правой или левой руки в произвольно выбранный момент времени, но не ранее чем через 2 секунды после предъявления стрелки-указателя на мониторе;

- В начале каждой пробы на период от 1 до 6 секунд на экране монитора предъявлялся фиксационный крестик. Затем предъявлялась стрелка-указатель: ‘→’ для нажатия указательным пальцем правой руки и ‘←’ для нажатия указательным пальцем левой руки;

- Регистрация ЭЭГ проводилась при помощи 31 хлорсеребряного электрода, размещенных в положениях: FC1, FC2, F5, F3, Fz, F4, F6, C5, C3, Cz, C4, C6, P5, P3, Pz, P4, P6, CP1, CP2, FC5, FC3, FCz, C2, C1, FC4, FC6, CP5, CP3, CPz, CP4, CP6;

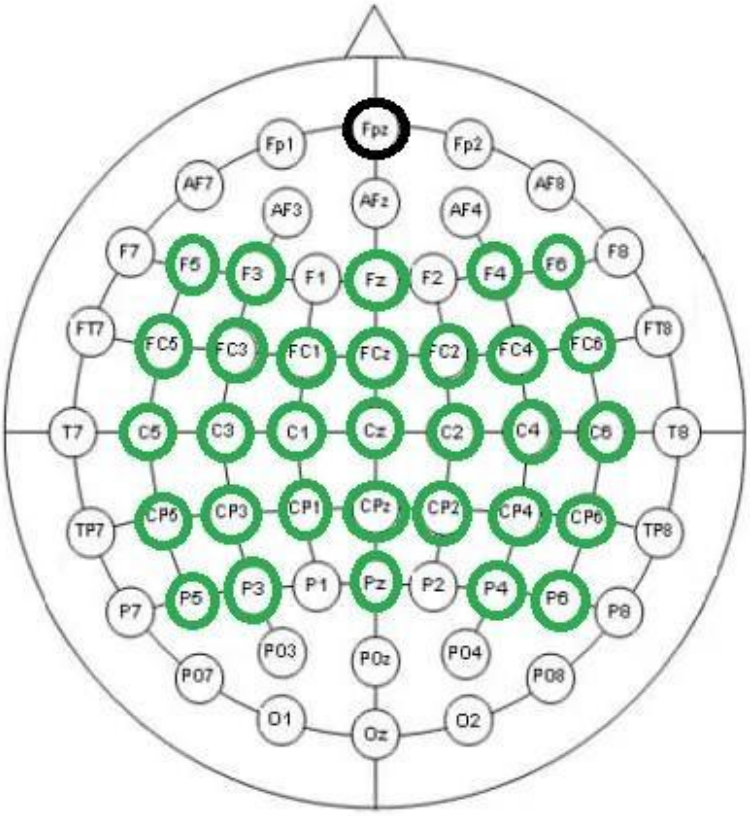


Рис. 3: Расположение электродов

- Данные содержат записи каждого из каналов на протяжении 34 минут, а также временные метки событий (показ левой стрелки-указателя, показ правой стрелки-указателя, нажатие на левый маркер, нажатие на правый маркер, отсутствие движений).

Постановка задачи классификации: Для анализа выберем данные ЭЭГ одного человека, они содержат показания каждого электрода в виде чисел с плавающей точкой. Пользуясь данными временных меток событий выберем для анализа следующие отрезки времени:

- ['показ левой стрелки-указателя' + 2.5c; 'показ левой стрелки-указателя' + 3.5c] (1)
- ['показ правой стрелки-указателя' + 2.5c; 'показ правой стрелки-указателя' + 3.5c] (2)

Таким образом, получаем множество 31-мерных векторов, каждому из этих векторов припишем класс 1, если вектор относится к диапазону (1), и класс 2, если вектор относится к диапазону (2). Всего для анализа выбрано 52396 точек из первого класса и 52395 точек из второго класса. Задача классификации будет заключаться в разделении этих двух классов. Признаками в данном случае будут являться показания электродов.

Разделим данные на два набора: 33% от всех данных будут являться тестовыми, а 67% - тренировочными.

Для упрощения вычислений и улучшения классификации выберем с помощью алгоритма RandomForest 10 наиболее значимых признаков. Такими будут являться признаки ['CP3', 'C5', 'P3', 'CP5', 'F3', 'C3', 'CP2', 'P6', 'Fz', 'FC1']. Таким образом уменьшили размерность векторов до 10. Далее для анализа используем 10-мерные векторы.

Применим различные алгоритмы классификации для этих данных:

Таблица 2: Точность предсказания в зависимости от выбора алгоритма

Алгоритм	Точность предсказания
"Оптимальное" разделение	Выпуклые оболочки классов пересекаются
SVM (ядро "poly")	0.654
RandomForest (максимальная глубина 20)	0.987
KNN	0.997

Также применим к данным модифицированный алгоритм "Оптимального" разделения. Идея модифицированного алгоритма заключается в следующем: добавим ещё один признак для анализа - класс вектора. Векторам из первого класса добавим ещё один признак равный 1, а векторам из второго класса добавим признак равный -1. В этом случае данные точно будут разделены. Получим уравнение гиперплоскости $a_1x_1 + ... + a_{11}x_{11} = 0$ (в данном случае $c = 0$). При тестировании примем $a_{11} = 0$.

Однако даже при таком подходе получаем, что модифицированный алгоритм "Оптимального" разделения даёт точность около 53%.

Также отметим, что при анализе данных ЭЭГ других испытуемых, точность предсказания различных алгоритмов не сильно отклонялась от значений, отображённых в таблице.

Вышеописанный анализ данных позволяет выдвинуть гипотезу о том, что представленные данные не разделимы линейно. А потому алгоритм "Оптимального" разделения не применим к этой задаче.

5. Заключение

В данной курсовой работе был представлен алгоритм разделения данных с использованием гиперплоскостей. Было проведено тестирование алгоритма на данных произвольной размерности, а также с различными степенями разделения данных.

Результаты тестирования показали, что алгоритм справляется с задачей разделения данных, когда данные имеют хорошую линейную структуру и высокую степень разделения. В таких случаях алгоритм достигает высокой точности предсказания, близкой к 1.0. Однако алгоритм работает только с данными, которые линейно разделимы. В случаях, когда данные имеют сложную структуру или перекрывающиеся классы, алгоритм может лишь указать на отсутствие гиперплоскости и не способен достичь высокой точности предсказания.

В целом, алгоритм разделения данных с использованием гиперплоскостей имеет свои преимущества и ограничения. Его эффективность зависит от структуры данных и требуемой степени разделения. Важно учитывать эти факторы при выборе и применении данного алгоритма для конкретной задачи.

Список литературы

[1] Фомин В.Н. Математическая теория обучаемых опознающих систем. 1976.