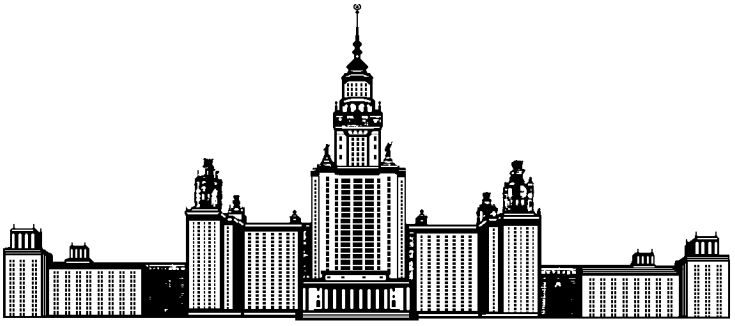
Московский государственный университет имени М. В. Ломоносова
Факультет Вычислительной Математики и Кибернетики



Магистерская программа «Логические и комбинаторные методы анализа данных»

Магистерская диссертация

«Исследование алгоритмов бустинга со встроенными поворотами признакового пространства»

Работу выполнил:
Гой Антон Сергеевич

Научный руководитель:
ассистент, к.ф-м.н.,
Китов Виктор Владимирович

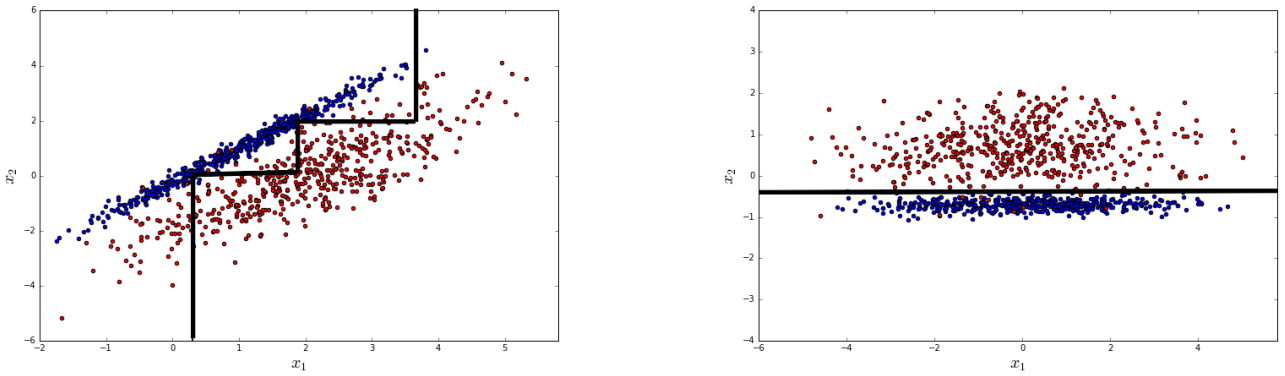
Москва, 2017

Оглавление

1. Введение

В современном машинном обучении одиночные модели используются крайне редко. Более того, повсеместное распространение получили ансамблевые алгоритмы (композиции алгоритмов), которые способны объединять множество базовых методов так, что итоговый алгоритм будет гораздо точнее, чем его отдельные составляющие. Нынешние ансамбли могут содержать сотни базовых алгоритмов, а иногда и тысячи. Популярность ансамблей заключается во множестве факторов, так, например, ансамбли способны аппроксимировать более сложные функции, чем базовые алгоритмы в отдельности. Кроме того, композиции алгоритмов имеют более высокую обобщающую способность, так как финальное предсказание вычисляется на основе ответов множества алгоритмов, что позволяет уменьшить влияние переобучения, с которым могут столкнутся отдельные модели.

Существует множество методов для создания ансамблей. Одним из самых успешных и известных среди них является бустинг [1], который итерационно строит ансамбль, добавляя на каждом шаге новый простой алгоритм, исправляющий ошибки предыдущих. Несмотря на кажущуюся на первый взгляд простоту, большое число «слабых» моделей в контексте бустинга способно строить необыкновенно точные ансамбли. Существует множество разновидностей бустинга, минимизирующие различные функционалы качества и использующие различные методы для оптимизации этих функционалов, но самым известным является алгоритм бустинга AdaBoost [1] и его многоклассовая версия SAMME [2].



(a) Исходные данные. Классификация решающим деревом глубины 3.

(b) Преобразованные данные. Классификация решающим деревом глубины 1.

Рис. 1: Разделяющая кривая решающего дерева.

Итеративная процедура AdaBoost никак не специфицирует тип базовых моделей, из которых состоит финальная композиция, но общепринятым стандартом являются решающие деревья небольшой глубины [3]. Решающие деревья являются стандартной базовой моделью не только для бустинга, но и для множества других ансамблей. Более того, некоторые ансамбли строятся только над решающими деревьями (например, Rotation Forest [4], о котором пойдет речь ниже). Регулируя глубину решающих деревьев, мы получаем необходимые для рассматриваемой композиции свойства:

- для бустинга требуются деревья небольшой глубины, которые постепенно бы улучшали предсказания;
- для бэггинг-алгоритмов [5] необходимы глубокие деревья, усредняя которые, можно значительно повысить обобщающую способность.

К недостаткам решающих деревьев можно отнести кусочно-постоянный вид аппроксимирующей функции, что в некоторых ситуациях становится значительным препятствием. Например, рассмотрим модельные данные, представленные на Рис. 1a. Линейная разделяющая кривая является оптимальным способом разделить два класса, но так как решающие деревья способны выделять только области, границы которых параллельны осям координат, прямая линия заменяется на зигзагообразную кривую, которая неудовлетворительно справляется с такой простой задачей. Похожие ситуации часто происходят в бустинге: одно дерево вряд ли сможет уловить такую структуру, но если строить очень большой ансамбль, то при помощи уменьшающихся разрезов можно добиться квази-линейной границы. Но если к тем же данным применить поворот осей (Рис. 1b), то решающему дереву потребуется всего один разрез пространства для достаточно хорошего разделения классов.

Найти правильные повороты не так просто, особенно если данные многомерные. Кроме того, отсутствие потенциально полезных поворотов всех признаков одновременно не означает, что мы не сможем найти подпространство, где поворот признаков будет оправдан. Интересным примером ансамбля, невероятно успешно применяющим идею поворотов, является алгоритм Rotation Forest [4]. Rotation Forest порождает для каждого решающего дерева преобразованную выборку, применяя в случайных подмножествах признаков поворот вдоль главных компонент. Rotation Forest хорошо себя зарекомендовал [4], показав высокую точность по сравнению с другими известными ансамблями (в том числе и AdaBoost). Кроме поворотов, основанных на методе главных компонент, были исследованы ансамбли из решающих деревьев [6], которые применяют к выборке случайные повороты.

В [7] была сделана попытка построить более точный ансамбль при помощи комбинирования методов Rotation Forest и AdaBoost. Был усилен классический Rotation Forest, с использованием в качестве базового классификатора алгоритм AdaBoost, из чего следует, что каждый из базовых AdaBoost обучается на специально преобразованных данных. Полученный метод был назван RotBoost. В ходе экспериментов было показано, что такая композиция алгоритмов уменьшает процент ошибок. Но данный подход, несмотря на все свои преимущества, не позволяет бустингу извлечь выгоду от поворотов при добавлении нового алгоритма в ансамбль.

Подводя итог, отметим, что несмотря на попытки усилить бустинг при помощи Rotation Forest, не было сделано попыток переноса преобразования признаков с внешнего уровня (до начала обучения ансамбля, как в RotBoost), на внутренний уровень (перед каждой итерацией бустинга). Последний подход потенциально способен усилить классификацию при помощи решающих деревьев и поэтому данное направление нуждается в подробном исследовании.

1.1 Постановка задачи

В данной работе были поставлены следующие задачи:

1. на основе метода AdaBoost разработать алгоритм бустинга, который на каждом шаге применяет повороты признакового пространства;
2. реализовать данный алгоритм;
3. произвести эксперименты, которые позволят сравнить предложенный алгоритм и стандартный AdaBoost;
4. при необходимости произвести дополнительные эксперименты.

1.2 Обозначения

Пусть $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ — обучающая выборка, где $\mathbf{x}_i \in \mathbb{R}^D$ — вектор признаков, а y_i принимает значения из конечного множества $\{1, 2, \ldots, K\}$. Наша цель на основе данной выборки

построить решающее правило $\hat{y} : \mathbb{R}^D \to \{1, 2, \ldots, K\}$, которое для произвольного $\mathbf{x}$ предсказывает $\hat{y}(\mathbf{x})$ — принадлежность $\mathbf{x}$ к одному из K классов. Как правило, выборку $\mathcal{L}$ удобно записать в виде матрицы $\mathbf{X} \in \mathbb{R}^{N \times D}$, строками которой являются $\mathbf{x}_i^\top$, и вектора $\mathbf{y} = (y_1, \ldots, y_N)^\top$.

В данной работе исследуется специальный тип решающего правила $\hat{y} = R(\hat{y}_1, \ldots, \hat{y}_L)$, который называется композицией (ансамблем) алгоритмов. Ансамбли по обучающей выборке строят L простых (базовых) алгоритмов $\hat{y}_l$ и затем, применяя решающее правило R к ответам простых моделей, вычисляют финальный ответ всей композиции. Существуют различные решающие правила R, например, $R(\hat{y}_1, \ldots, \hat{y}_L) = \arg\max_t \sum_{l=1}^{L} [\hat{y}_l = t]$ — голосование алгоритмов y_l. Другим примером решающего правила является взвешенное голосование $R(\hat{y}_1, \ldots, \hat{y}_L) = \arg\max_t \sum_{l=1}^{L} \alpha_l [\hat{y}_l = t]$, которое назначает каждому базовому алгоритму вес α_l. Чем больше этот вес, тем важнее вклад $\hat{y}_l$ в финальное предсказание.

1.3 Обзор алгоритма Rotation Forest

Основная цель данного раздела продемонстрировать очень важный для данной работы алгоритм, в котором уже успешно используется идея поворотов признакового пространства — Rotation Forest.

Алгоритм Rotation Forest был предложен в 2006 году тремя исследователями: *J. Rodriǵuez, L. Kuncheva, C. Alonso* [4]. Алгоритм представляет собой ансамбль решающих деревьев. По своей структуре Rotation Forest наиболее близок к бэггинг-алгоритмам, так как финальные предсказания вычисляются как среднее арифметическое ответов базовых моделей. Кроме того, Rotation Forest, как и бэггинг, стремится обучить базовые алгоритмы так, чтобы их ответы были как можно меньше скоррелированы. Но в отличие от бэггинга, в котором для обучения каждого базового алгоритма используются случайные бутстрэпп-подвыборки (что позволяет уменьшить корреляцию между алгоритмами), Rotation Forest применяет PCA-преобразование (метод главных компонент) к случайным подмножествам признаков. Псевдокод алгоритма представлен в Алгоритме 1.

Рассмотрим на интуитивном уровне базовые идеи и шаги алгоритма. Главная цель строки 2 — это разбить все пространство признаков на случайные подпространства, к которым на последующих шагах будут применены повороты. В строках 3-7 для каждого из полученных подпространств определяются те объекты, на основе которых будет вычисляться поворот. Выбирая только часть объектов, мы увеличиваем непохожесть итоговых поворотов, что, в свою очередь, уменьшает скоррелированность алгоритмов. Определив $\mathbf{X}'_{l,s}$ — матрица объект-признак для подпространства $\mathcal{F}_s$, — применяем метод главных компонент к этой матрице. Данный метод можно рассматривать как некоторое вращение признакового пространства, которое определяет новый базис $\mathbf{a}_{l,s}^1, \ldots, \mathbf{a}_{l,s}^M$, причем данные вектора будут ортонормированными и соответствовать направлениям с наибольшей дисперсией. Теперь вращение $\mathbf{x}$ в данном подпространстве можно достичь умножением подописания $\mathbf{x}$, соответствующее подпространству $\mathcal{F}_s$, на матрицу, которая составлена из векторов базиса $\{\mathbf{a}_{l,s}^r\}$. Для того чтобы осуществлять это вращение во всех подпространствах одновременно, в строке 8 строится матрица $\mathbf{A}_l$, но на прямую использовать эту матрицу для вращения нельзя, так как необходимо упорядочить строки $\mathbf{A}_l$ так, чтобы при умножении слева вектора $\mathbf{x}$ поворот применялся к корректным координатам, что и делается в строке 9. Оставшиеся пункты значительно проще: в строке 10 применяется повороты к исходной матрице и обучается классификатор на преобразованных данных.

В алгоритме Rotation Forest не очевидна мотивация использования именно метода главных компонент для поиска поворотов, но все попытки применения других методов [8] (в том числе случайных поворотов) оказались безуспешными - все методы показали себя хуже в сравнительных экспериментах.

Алгоритм 1 Rotation Forest

Дано:

Выборка $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} = (\mathbf{X}, \mathbf{y})$;
Количество базовых моделей L;
Количество подмножеств признаков S;
Размер подвыборки P;

Найти:

Алгоритм $\hat{y} : \mathbb{R}^D \to \{1, 2, \ldots, K\}$;

1: **Для всех** $l = 1, \ldots, L$
2: Случайным образом разбиваем все множество признаков $\mathcal{F}$ на $\mathcal{F}_1, \ldots, \mathcal{F}_S$ — непересекающиеся подмножества, где S — гиперпараметр алгоритма. Предположим, что S делит D без остатка, тогда, если $M = D/S$, то $|\mathcal{F}_k| = M$;
3: **Для всех** $s = 1, \ldots, S$
4: Построим матрицу $\mathbf{X}_{l,s} \in \mathbb{R}^{N \times M}$, которая содержит в себе те столбцы матрицы $\mathbf{X}$, которые принадлежат $\mathcal{F}_s$
5: Выберем с возвращением P строк матрицы $\mathbf{X}_{l,s}$, где $P \leq N$. Из выбранных различных строк составим матрицу $\mathbf{X}'_{l,s}$;
6: Используя PCA-преобразование, находим все M главных компонент матрицы $\mathbf{X}'_{l,s}$. Пусть $\mathbf{a}_{l,s}^1, \ldots, \mathbf{a}_{l,s}^M$ — вектор-столбцы, соответствующие главным компонентам. Из главных компонент составляем матрицу $\mathbf{A}_{l,s} = [\mathbf{a}_{l,s}^1, \ldots, \mathbf{a}_{l,s}^M] \in \mathbb{R}^{M \times M}$;
7: **Конец итерации**
8: Составим из матриц $\mathbf{A}_{l,1}, \ldots, \mathbf{A}_{l,S}$ блочно-диагональную матрицу $\mathbf{A}_l$:

$$\mathbf{A}_l = \begin{bmatrix} \mathbf{A}_{l,1} & 0 & \ldots & 0 \\ 0 & \mathbf{A}_{l,2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{A}_{l,S} \end{bmatrix}$$

9: Упорядочим строки матрицы $\mathbf{A}_l$ так, чтобы они соответствовали исходному порядку признаков. Матрицу с корректно упорядоченными строками обозначим $\mathbf{R}_l$;
10: Пусть $\mathbf{X}_l = \mathbf{X}\mathbf{R}_l$.
11: Обучим базовый алгоритм $\hat{y}_l$ на $(\mathbf{X}_l, \mathbf{y})$ при помощи весов w_n^l;
12: **Конец итерации**
13: Финальный прогноз классификатора:

$$\hat{y}(\mathbf{x}) = \arg\max_y \sum_{l=1}^{L} \left[\hat{y}_l(\mathbf{x}^\top \mathbf{R}_l) = y\right].$$

Отметим, что строки 2-10 можно рассматривать как преобразование специального вида (будем называть его $r_{M,P}$-*преобразование*), которое по исходной матрице $\mathbf{X}$ вычисляет новую матрицу $\mathbf{X}_l = r_{M,P}(\mathbf{X})$. Контролируется данное преобразование двумя параметрами M (или S) и P. Ввиду важности данного преобразования выделим его в отдельный Алгоритм 2.

Алгоритм 2 Преобразование $r_{M,P}$

Дано:

Матрица объект-признак $\mathbf{X} \in \mathbb{N} \times \mathbb{B}$;

Максимальный размер подмножеств M (полагаем, что M делит D без остатка);

Размер подвыборки P;

Найти:

Матрица $\hat{\mathbf{X}} \in \mathbb{R}^{N \times D}$;

1: Случайным образом разбиваем все множество признаков $\mathcal{F}$ на $\mathcal{F}_1, \ldots, \mathcal{F}_S$ — непересекающиеся подмножества, где $S = D/M$.

2: **Для всех** $s = 1, \ldots, S$

3: Построим матрицу $\mathbf{X}_s \in \mathbb{R}^{N \times M}$, которая содержит в себе те столбцы матрицы $\mathbf{X}$, которые принадлежат $\mathcal{F}_s$

4: Выберем с возвращением P строк матрицы $\mathbf{X}_s$, где $P \leq N$. Из выбранных различных строк составим матрицу $\mathbf{X}'_s$;

5: Используя PCA-преобразование, находим все M главных компонент матрицы $\mathbf{X}'_s$. Пусть $\mathbf{a}_s^1, \ldots, \mathbf{a}_s^M$ — вектор-столбцы, соответствующие главным компонентам. Из главных компонент составляем матрицу $\mathbf{A}_s = [\mathbf{a}_s^1, \ldots, \mathbf{a}_s^M] \in \mathbb{R}^{M \times M}$;

6: **Конец итерации**

7: Составим из матриц $\mathbf{A}_1, \ldots, \mathbf{A}_S$ блочно-диагональную матрицу $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 & \ldots & 0 \\ 0 & \mathbf{A}_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{A}_S \end{bmatrix}$$

8: Упорядочим строки матрицы $\mathbf{A}$ так, чтобы они соответствовали исходному порядку признаков. Матрицу с корректно упорядоченными строками обозначим $\mathbf{R}$;

9: Получаем финальную матрицу $\hat{\mathbf{X}} = \mathbf{X}\mathbf{R}$.

В качестве базовых классификаторов были использованы решающие деревья, так как они чувствительны к поворотам координатных осей. То есть за счет поворотов признаков в случайных подпространствах каждое дерево будет с большой вероятностью отличаться от других деревьев в ансамбле, что позволит уменьшить совокупную дисперсию алгоритма (*variance*) в разложении на смещение и дисперсию (*bias-variance decomposition*).

1.4 Обзор алгоритма AdaBoost

Алгоритму AdaBoost посвящено множество научных статей и глав в учебниках. Поэтому не будем вдаваться в его детали, так как полное описание AdaBoost можно найти в [9]. Впервые AdaBoost (от *Adaptive Boosting*) был предложен в [1] (1997) и стал настоящим прорывом в области построения точных алгоритмов на базе «слабых» моделей. Основная идея AdaBoost заключается в перевзвешивании выборки на каждой итерации таким образом, чтобы наиболее сложные объекты получали больший вес. Это позволяет обучить новый алгоритм, который сможет исправить ошибки предыдущих, уделяя усиленное внимание объектам с большим весом. Как и для всех видов бустинга, решающее правило представляет собой взвешенную сумму.

В изначальном виде AdaBoost был способен решать задачи классификации только на два класса. Обобщение алгоритма на многоклассовый случай было предложено в [2]. В данной работе авторы предлагают два алгоритма SAMME и SAMME.R, которые представляют собой многоклассовую модификацию AdaBoost для дискретного (когда базовые алгоритмы возвращают только метки классов) и непрерывного (алгоритмы вычисляют вероятности) случаев. Приведем алгоритм SAMME (Алгоритм 3), так как он понадобится нам для дальнейших рассуждений.

Алгоритм 3 SAMME

Дано:

Выборка $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N = (\mathbf{X}, \mathbf{y})$;

Количество базовых моделей L;

Найти:

Алгоритм $\hat{y} : \mathbb{R}^D \to \{1, 2, \ldots, K\}$;

1: Проинициализируем веса для каждого объекта: $w_i^1 = \frac{1}{N}$, $i = 1, \ldots, N$;

2: **Для всех** $l = 1, \ldots, L$

3: Обучим базовый алгоритм $\hat{y}_l$ при помощи весов w_n^l;

4: Вычислим взвешенную ошибку ε_l алгоритма $\hat{y}_l$:

$$\varepsilon_l = \sum_{i=1}^{N} w_i^l \left[\hat{y}_l(\mathbf{x}_i) \neq y_i\right]$$

5: Вычислим вес нового классификатора:

$$\alpha_l = \ln \frac{1 - \varepsilon_l}{\varepsilon_l} + \ln(K - 1).$$

6: Пересчитаем и нормализуем веса:

$$\overline{w}_i^{l+1} = w_i^l \cdot \exp\Big(\alpha_l \cdot \left[\hat{y}_l(\mathbf{x}_i) \neq y_i\right]\Big),\ i = 1, \ldots, N, \quad w_i^{l+1} = \frac{\overline{w}_i^{l+1}}{\sum_{i=1}^N \overline{w}_i^{l+1}},\ i = 1, \ldots, N.$$

7: **Конец итерации**

8: Финальный прогноз классификатора:

$$\hat{y}(\mathbf{x}) = \arg\max_y \sum_{l=1}^{L} \alpha_l \left[\hat{y}_l(\mathbf{x}) = y\right].$$

Стоит отметить, что SAMME является прямым обобщением AdaBoost, так как при $K = 2$, SAMME становится эквивалентен двухклассовому AdaBoost.

Схематично SAMME изображен на Рис. ??.

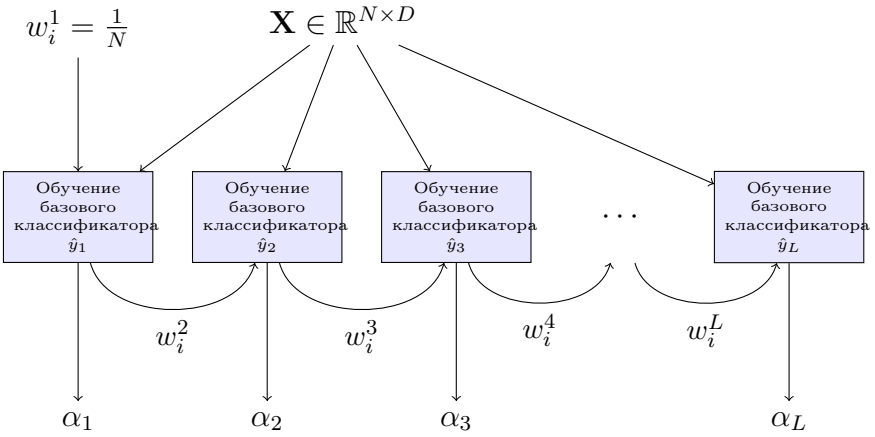


Рис. 2: Схематичное описание алгоритма SAMME. На каждом шаге при помощи весов обучается новая модель y_l на $\mathbf{X}$ и определяется ее вес α_l.

2. Алгоритм Iterative Rotation AdaBoost

2.1 Описание алгоритма

Ключевой подход алгоритма Rotation Forest, который основывается на поворотах в случайных подпространствах признаков, оказался чрезвычайно успешным и позволил значительно опередить своих конкурентов [4, 8]. Во введении были озвучены идеи о том, почему уместно делать поворот признакового пространства. Поэтому естественным образом встает вопрос: возможно ли применить методику Rotation Forest к другим алгоритмам. В частности, требует исследования проблема применимости идей Rotation Forest к бустингу: могут ли повороты пространства помочь решающим деревьям в контексте бустинга улучшить точность. С целью проверки данного предположения в текущей работе предлагается алгоритм, который является соединением идей бустинга и поворотов в случайных подпространствах перед очередной итерацией бустинга. Будем ссылаться на этот алгоритм по имени Iterative Rotation AdaBoost, подчёркивая названием, во-первых, оба подхода, которые лежат в его основе, и, во-вторых, что преобразования признаков применяются для каждого базового алгоритма отдельно.

Базовая архитектура алгоритма была позаимствована из бустинга: алгоритмы строятся последовательно, используя веса объектов $w_i, i = 1, \ldots, N$, которые пересчитываются на каждой итерации. Но в отличие от привычного AdaBoost, базовый классификатор y_l обучается не на исходной матрице объектов $\mathbf{X}$, а на матрице $\mathbf{X}_l = r_{M,P}(\mathbf{X})$, которая получается при помощи $r_{M,P}$-преобразования матрицы $\mathbf{X}$. Таким образом, на каждом шаге бустинга к случайным подмножествам признаков применяются повороты, которые приводят к ортонормированному базису векторов, отвечающим направлениям с наибольшим изменением дисперсии. На первый взгляд, абсолютно не очевидно преимущество, которое может принести такое преобразование признаков, причем, если учесть, что на каждой итерации бустинга оно будет применятся в случайных подмножествах признаков. Но как будет показано далее, введенные в процедуру алгоритма AdaBoost, существенным образам повышают точность алгоритма.

Пошагово алгоритм Iterative Rotation AdaBoost описан в Алгоритме 4, схематичное описание которого представлено на Рис. 3.

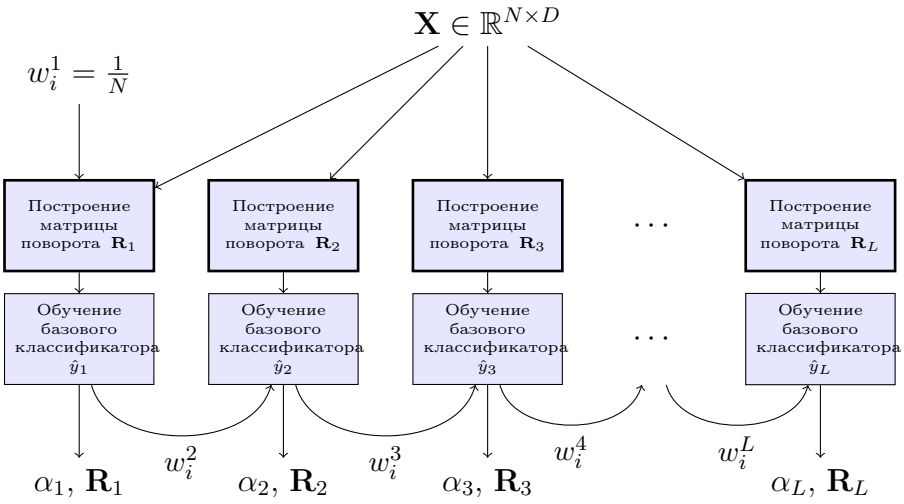


Рис. 3: Схематичное описание алгоритма Iterative Rotation AdaBoost. На каждом шаге к исходной выборке применяется $r_{M,P}$-преобразование, в результате которого вычисляется новая матрица поворота $\mathbf{R}_l$. Алгоритмы $\hat{y}_l$ обучаются на преобразованной выборке.

Приведенный алгоритм предполагает, что базовые алгоритмы $\hat{y}_l$ возвращают метки классов. Аналогичный подход можно применить и к алгоритму SAMME.R, в случае если базовые алгоритмы вычисляют вероятности принадлежности к каждому из классов.

2.2 Взвешенная версия Iterative Rotation AdaBoost

Описанный выше алгоритм применяет PCA-преобразование к подмножествам признаков, что соответствует поворотам специального вида. Но данный подход никак не использует веса $w_i^l, n = 1, \ldots, N$, в которых содержится значительная информация об объектах. По этой причине очевидным образом возникает потребность заменить метод главных компонент его взвешенным вариантом. Поясним отличия.

В основе многих реализаций метода PCA лежит SVD-разложение. Для чего матрицу $\mathbf{X}$ предварительно центрируют:

$$\mathbf{x}'_n = \mathbf{x}_n - \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n.$$

И к новой матрице $\mathbf{X}'$ применяют SVD-разложение с целью найти главные компоненты. Взвешенный аналог метода PCA состоит из тех же самых шагов, но с использованием весов. Центрирование выглядит следующим образом:

$$\mathbf{x}'_n = \mathbf{x}_n - \sum_{n=1}^{N} w_n \mathbf{x}_n,$$

где $w_n, n = 1, \ldots, N$ — веса объектов: $w_n >= 0$, $\sum_{n=1}^{N} w_n = 1$. А SVD-разложение применяется к матрице $\mathbf{W}^{\frac{1}{2}}\mathbf{X}'$, где $\mathbf{W}^{\frac{1}{2}} = \operatorname{diag}\left(\sqrt{w_1}, \ldots, \sqrt{w_N}\right)$.

Таким образом, если всюду в Алгоритме 4 заменить метод главных компонент на его взвешенную версию, то получим модификацию метода Iterative Rotation AdaBoost, в котором при повороте признаков используется информация о «важности» объектов. На данный метод мы будем ссылаться как Iterative Rotation AdaBoost[Weights]. Данная модификация алгоритма также будет участвовать в сравнительных экспериментах.

Алгоритм 4 Iterative Rotation AdaBoost

Дано:

Выборка $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N = (\mathbf{X}, \mathbf{y})$;

Количество базовых моделей L;

Максимальный размер подмножеств M (полагаем, что M делит D без остатка);

Размер подвыборки P;

Найти:

Алгоритм $\hat{y} : \mathbb{R}^D \to \{1, 2, \ldots, K\}$;

1: Проинициализируем веса для каждого объекта: $w_i^1 = \frac{1}{N}$, $i = 1, \ldots, N$;

2: **Для всех** $l = 1, \ldots, L$

3: Применим $r_{M,P}$ преобразование к матрице $\mathbf{X}$. Получим матрицу $\mathbf{X}_l$. Сохраним отдельно матрицу поворота $\mathbf{R}_l$, которая была получена на финальном шаге.

4: Обучим базовый алгоритм $\hat{y}_l$ на $\mathbf{X}_l$ при помощи весов w_n^l;

5: Вычислим взвешенную ошибку ε_l алгоритма $\hat{y}_l$:

$$\varepsilon_l = \sum_{i=1}^{N} w_i^l \left[\hat{y}_l(\mathbf{x}_i^\top \mathbf{R}_l) \neq t_i\right].$$

6: Вычислим вес нового классификатора:

$$\alpha_l = \ln \frac{1 - \varepsilon_l}{\varepsilon_l} + \ln(K - 1).$$

7: Пересчитаем и нормализуем веса:

$$\overline{w}_i^{l+1} = w_i^l \cdot \exp\left(\alpha_l \cdot \left[\hat{y}_l(\mathbf{x}_i^\top \mathbf{R}_l) \neq t_i\right]\right),\ i = 1, \ldots, N, \quad w_i^{l+1} = \frac{\overline{w}_i^{l+1}}{\sum_{i=1}^N \overline{w}_i^{l+1}},\ i = 1, \ldots, N.$$

8: **Конец итерации**

9: Финальный прогноз классификатора:

$$\hat{y}(\mathbf{x}) = \arg\max_y \sum_{l=1}^{L} \alpha_l \left[\hat{y}_l(\mathbf{x}^\top \mathbf{R}_l) = y\right].$$

2.3 Iterative Rotation AdaBoost со случайными поворотами

В [8] было показано, что для Rotation Forest метод главных компонент является предпочтительным и превосходит другие методы преобразования признаков (в том числе случайные повороты). Кроме того, в [6] была озвучена идея, что случайные повороты можно применять в бустинге, но отельного исследования, которое позволило бы судить о том как случайные повороты влияют на точность бустинга, проведено не было. Поэтому для того, чтобы проверить данную гипотезу на практике и сравнить повороты при помощи метода главных компонент и случайных поворотов, мы рассмотрим Iterative Rotation AdaBoost со случайными поворотами (Iterative Rotation AdaBoost[Random]). Данная модификация отличается от предложенного Iterative Rotation AdaBoost в разделе 2.1 тем, что вместо метода PCA для вычисления матриц $\mathbf{A}_s$, состоящих из главных компонент, используется QR-разложение некоторой случайной матрицы $\mathbf{M}$. Таким образом, случайный поворот определяется ортогональной матрицей $\mathbf{Q}$, где $\mathbf{M} = \mathbf{QR}$ — QR-разложение случайной матрицы $\mathbf{M}$. Теперь, полагая $\mathbf{A}_s = \mathbf{Q}$, повторяем все

обычные действия из Алгоритма 2.

Вышеописанное преобразование при помощи случайных поворотов назовем $\hat{r}_{M,P}$-преобразование и приведем его детально в Алгоритме 5.

Алгоритм 5 Преобразование $\hat{r}_{M,P}$

Дано:

Матрица объект-признак $\mathbf{X} \in \mathbb{N} \times \mathbb{B}$

Максимальный размер подмножеств M (полагаем, что M делит D без остатка);

Размер подвыборки P;

Найти:

Матрица $\hat{\mathbf{X}} \in \mathbb{R}^{N \times D}$

1: Случайным образом разбиваем все множество признаков $\mathcal{F}$ на $\mathcal{F}_1, \ldots, \mathcal{F}_S$ — непересекающиеся подмножества, где $S = D/M$.

2: **Для всех** $s = 1, \ldots, S$

3: Построим матрицу $\mathbf{X}_s \in \mathbb{R}^{N \times M}$, которая содержит в себе те столбцы матрицы $\mathbf{X}$, которые принадлежат $\mathcal{F}_s$

4: Выберем с возвращением P строк матрицы $\mathbf{X}_s$, где $P \leq N$. Из выбранных различных строк составим матрицу $\mathbf{X}'_s$;

5: Генерируем случайную матрицу $\mathbf{M}$. Вычисляем QR-разложение матрицы $\mathbf{M}$. Пусть $\mathbf{A}_s \in \mathbb{R}^{M \times M}$ — ортогональная матрица, полученная в результате QR-разложения $\mathbf{M}$

6: **Конец итерации**

7: Составим из матриц $\mathbf{A}_1, \ldots, \mathbf{A}_S$ блочно-диагональную матрицу $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 & \ldots & 0 \\ 0 & \mathbf{A}_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{A}_S \end{bmatrix}$$

8: Упорядочим строки матрицы $\mathbf{A}$ так, чтобы они соответствовали исходному порядку признаков. Матрицу с корректно упорядоченными строками обозначим $\mathbf{R}$;

9: Получаем финальную матрицу $\hat{\mathbf{X}} = \mathbf{X}\mathbf{R}$.

---

3. Эксперименты

3.1 Исходные данные

В Таблице 1 перечислены 30 тестовых выборок, которые были использованы для всех сравнительных экспериментов. Все выборки находятся в открытом доступе в *UCI Machine Learning Repository* [10]. В первом столбце Таблицы 1 указано название выборки, второй и третий столбцы соответствуют количеству классов и размеру выборки (без объектов с пропущенными значениями) соотвественно; в последних двух столбцах представлено количество категориальных и числовых признаков соответственно (под числовыми признаками понимаются не только вещественные, но также и целочисленные и ординальные признаки).

Выборка	Классы	Объекты	Категор. признаки	Числовые признаки
Abalone	28	4177	1	7
Balance Scale	3	625	—	4
Banknote Auth.	2	1372	—	4
Blood Transfusion	2	748	—	4
Breast Cancer	2	569	—	30
Car Eval.	4	1728	21	—
Chess	2	3196	73	—
Cleveland Heart	5	297	7	6
Credit Approval	2	653	9	6
Digits	10	1797	—	64
EEG	2	14980	—	14
Ecoli	8	336	—	7
Forest Type	4	523	—	27
Glass	6	214	—	9
ILPD	2	579	1	9
Ionosphere	2	351	—	34
Iris	3	150	—	4
Letters	26	20000	—	16
Mammographic	2	831	2	2
Musk	2	476	—	166
Occupancy	2	20560	—	5
Pima	2	768	—	8
Sonar	2	208	—	60
Spectf	2	267	—	44
Teaching Assistant	3	151	4	1
Thoraric Surgery	2	470	13	3
Tic-Tac-Toe	2	958	27	—
Vertebral	2	310	—	6
Wine	3	178	—	13
Zoo	7	101	15	1

Таблица 1: Параметры тестовых выборок

Использованный набор выборок является достаточно репрезентативным, так как охватывает широкое множество типов обучающих данных, которые встречаются на практике (многоклассовость, наличие категориальных признаков и т.д.).

Данные были предобработаны следующим образом:

1. объекты, содержащие пропущенные значения, были удалены;

2. категориальные признаки были преобразованы при помощи *One-Hot-Encoding* преобразования, т.е. каждое значение категориального признака заменялось на бинарный вектор длины c, где c — количество категорий в признаке;

3. числовые признаки были нормализованы:

$$x'_i = \frac{x_i - \mu}{\sigma},$$

где $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$, $\sigma^2 = \frac{1}{N}\sum_{i=i}^{N}(x_i - \mu)^2$.

3.2 Описание условий экспериментов и метрик

Допустим нам необходимо сравнить алгоритмы A_1 и A_2, которые зависят от параметров θ и ϕ соответственно, на некоторой выборке $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Вопрос о том как это сделать правильно является не тривиальным, так как:

1. необходимо, чтобы данное сравнение было максимально честным и позволяло каждому алгоритму проявить свой потенциал;

2. результаты полученные в ходе экспериментов должны быть обоснованными со статистической точки зрения;

Кроме того, заранее фиксируется некоторая метрика качества $M = M(A(\theta), \mathcal{L}_{train}, \mathcal{L}_{test})$, которая характеризует «близость» предсказаний алгоритма $A(\Theta)$, обученного на $\mathcal{L}_{train}$, к эталонным меткам классов из $\mathcal{L}_{test}$. Обычно в качестве такой метрики выступает точность алгоритма на тестовой выборке (доля верно угаданных меток классов).

Выборки $\mathcal{L}_{train}$ и $\mathcal{L}_{test}$ можно получить случайно разделив $\mathcal{L}$ в некотором соотношении (например, $3:1$), но данный подход редко используется на практике, так как при недостаточном размере исходной выборки такое разбиение может привести к нерепрезентативным обучающей и тестовой выборкам. Поэтому широко распространен метод кросс-валидации по K блокам, который позволяет задействовать в обучении и тестировании все объекты из выборки $\mathcal{L}$ и финальное значение метрики качества вычисляется следующим образом:

$$M_{cv} = M_{cv}(A(\Theta), \mathcal{L}) = \frac{1}{K}\sum_{(\mathcal{L}_{train}, \mathcal{L}_{test})} M(A(\theta), \mathcal{L}_{train}, \mathcal{L}_{test}).$$

Значение $M_{cv}(A(\Theta), \mathcal{L})$, если не возникает неоднозначности будем обозначать $M_{cv}(A)$.

Существует множество подходов сравнения алгоритмов машинного обучения, которыми пользуется научное сообщество (см. [11]), но в большинстве случаев все сводится к фиксированию некоторых θ_0 и ϕ_0, и задача сравнения A_1 и A_2 сводится к сравнению $A_1(\theta_0)$ и $A_2(\phi_0)$. Далее для того чтобы получить значение метрики качества M для алгоритмов $A_1(\theta_0)$ и $A_2(\phi_0)$ применяют кросс-валидацию по нескольким блокам. Уже на данном этапе можно, например, сказать, что A_1 лучше, чем A_2 на выборке $\mathcal{L}$ в смысле метрики M, так как значение $M_{cv}(A_1)$ больше чем $M_{cv}(A_2)$, но такой результат невозможно было бы считать статистически значимым, так как полученный прирост мог быть вызван разбиением кросс-валидации на блоки, которое оказалось более «удачным» для алгоритма A_1. Чтобы обойти данную проблему мы отказываемся от фиксированного разбиения и запускаем кросс-валидацию T раз (с разными разбиениями), получая таким образом для каждого алгоритма последовательность чисел $\mathcal{M}_i = (M_{cv}^1(A_i), \ldots, M_{cv}^T(A_i))$.

Теперь вопрос о том, есть ли какие-либо статистически значимые различия между результатами $\mathcal{M}_1$ алгоритма A_1 и результатами $\mathcal{M}_2$ алгоритма A_2 сводится к применению парного статистического теста к выборкам $\mathcal{M}_1$ и $\mathcal{M}_2$ с некоторым уровнем значимости α.

Вышеописанный подход является широко распространенным и применяется во многих публикациях. Но очевидным минусом этого способа сравнения двух алгоритмов является его исходная посылка: мы сравниваем не алгоритмы A_1 и A_2 при произвольных параметрах, а лишь $A_1(\theta_0)$ и $A_2(\phi_0)$, поэтому рассуждения, что один алгоритм лучше другого, если он лучше при фиксированных параметрах, являются неверными.

В данной работе используется несколько отличная от описанной схема сравнения алгоритмов. Псевдокод описан в Алгоритме 6.

Алгоритм 6 Сравнение алгоритмов A_1 и A_2 на выборке $\mathcal{L}$ на основе метрики M

Дано:

Алгоритмы A_1, A_2
Множество комбинаций перебираемых параметров $\Theta = \{\theta\}$ для алгоритма A_1
Множество комбинаций перебираемых параметров $\Phi = \{\phi\}$ для алгоритма A_2
Выборка $\mathcal{L}$
Метрика M
Количество блоков в кросс-валидации K
Количество запусков кросс-валидаций T
Уровень значимости α

Для всех $t = 1, \dots, T$
  Случайным образом разбиваем $\mathcal{L}$ на $\mathcal{L}_{cv}$ и $\mathcal{L}_{test}$ в соотношении $3:1$
  Для всех $\theta \in \Theta$
    Выполняем кросс-валидацию $CrossValidation(A_1(\theta), \mathcal{L}_{cv}, K)$
    Получаем значение метрики $M_{cv}(A_1(\theta))$
  Конец итерации
  Найти параметры $\theta_\star$, которые минимизируют (или максимизируют) метрику M на кросс-валидации:

$$\theta_\star = \arg\min_{\theta \in \Theta} M_{cv}(A_1(\theta))$$

  Для всех $\phi \in \Phi$
    Выполняем кросс-валидацию $CrossValidation(A_2(\phi), \mathcal{L}_{cv}, K)$
    Получаем значение метрики $M_{cv}(A_2(\phi))$
  Конец итерации
  Найти параметры $\phi_\star$, которые минимизируют (или максимизируют) метрику M на кросс-валидации:

$$\phi_\star = \arg\min_{\phi \in \Phi} M_{cv}(A_2(\phi))$$

  Вычислить значение метрики на тестовой выборке $M_t(A_1) = M(A_1(\theta_\star), \mathcal{L}_{cv}, \mathcal{L}_{test})$
  Вычислить значение метрики на тестовой выборке $M_t(A_2) = M(A_2(\phi_\star), \mathcal{L}_{cv}, \mathcal{L}_{test})$
Конец итерации
Получаем после всех запусков две выборки: $\mathcal{M}_1 = \{M_t(A_1)\}_{t=1}^T$ и $\mathcal{M}_2 = \{M_t(A_2)\}_{t=1}^T$
Применить парный статистический тест к выборкам $\mathcal{M}_1$ и $\mathcal{M}_2$

Главным отличием Алгоритма 6 от общепринятого способа сравнения алгоритмов машинного обучения является то, что мы не фиксируем конкретные параметры, а делаем по ним перебор. Очевидно, что в виду бесконечного числа значений полный перебор параметров сделать невозможно, но возможно сузить поиск, выбрав во-первых наиболее важные параметры и во-вторых перебирать значения каждого из выделенных параметров по конечной сетке значе-

ний. Также отметим, что при таком способе мы не можем использовать кросс-валидацию для поиска параметров, которые минимизируют (максимизируют) метрику, так как если модель A_1 имеет больше параметров, чем модель A_2, то в таком случае A_1 имеет большие шансы получить оптимальное значение метрики M. Для того чтобы не давать преимущества алгоритмам с большим числом параметров мы выбираем лучшие параметры на основе значений метрики M, полученных по кросс-валидации, а итоговое значение метрики вычисляется на отложенном тестовом множестве, которое никак не участвовало в подборе параметров. Чтобы сохранить возможность применять статистический тест, необходимо выполнить описанную выше процедуру множество раз. В результате мы получаем две выборки $\mathcal{M}_1$ и $\mathcal{M}_2$ для алгоритмов A_1 и A_2 соответственно, которые поступают на вход парного теста с уровнем значимости α. В данной работе в качестве парного статистического теста будем использовать T-критерий Вилкоксона — непараметрический статистический тест, который является альтернативой парного t-теста. Нулевой гипотезой T-критерий Вилкоксона является утверждение, что медианы двух выборок парных измерений равны. В зависимости от потребностей мы можем сформулировать разные альтернативные гипотезы: одностороннюю или двухстороннюю. Односторонний критерий соответствует альтернативной гипотезе, что медиана одной выборке больше, чем медиана другой. Двухсторонний критерий соответствует альтернативной гипотезе, что медианы выборок не совпадают. Главным преимуществом T-критерия Вилкоксона является тот факт, что данный тест применим в ситуациях, когда выборки не подчиняются нормальному распределению, более того тест не накладывает никаких ограничений ни на параметры порождающего распределения, ни на его форму. Кроме того T-критерий Вилкоксона более устойчив к выбросам, чем t-тест. В [11] показано, что T-критерий Вилкоксона является более эффективным способом сравнения двух классификаторов по сравнению с t-тестом. Далее всюду при упоминании статистического теста будем иметь в виду T-критерий Вилкоксона.

Сравнивая два классификатора A_1 и A_2 на некоторой выборке нас интересуют три исхода:

1. алгоритмы не имеют статистически значимых отличий (медианы выборок равны — «ничья»);

2. алгоритм A_1 в среднем лучше, чем A_2 (медиана выборки для A_1 больше медианы для A_2 — «победа» A_1);

3. алгоритм A_2 в среднем лучше, чем A_1 (медиана выборки для A_2 больше медианы для A_1 — «проигрыш» A_1);;

Зафиксировать первый исход можно при помощи двухстороннего теста. Если же p-значение меньше установленного уровня значимости, мы должны отвергнуть нулевую гипотезу и принять альтернативную для двустороннего теста — медианы выборок не совпадают. Далее в условиях альтернативной гипотезы мы должны принять решение о наличии второго или третьего исхода, что можно сделать при помощи применения одностороннего теста с альтернативной гипотезой: медиана выборки для A_1 больше медианы для A_2. Таким образом при условии альтернативной гипотезы мы фиксируем «победу» A_1 (или что тоже самое — «проигрыш» A_2), а при условии нулевой гипотезы фиксируем «проигрыш» A_1 (или что тоже самое — «победа» A_2).

Победа на одной выборке не означает, что данный алгоритм можно рассматривать как более успешный во всех других случаях. Чтобы с большей уверенностью говорить о факте, что A_1 лучше в среднем чем A_2 необходимо рассматривать множество выборок. В данной работе мы в качестве определения «лучше в среднем» используем количество выборок, на которых данный алгоритм стал «победителем». Данное определение ни в коем случае нельзя считать формальным, оно лишь является выражением здравого смысла.

Перейдем теперь к описанию конкретных значений из Алгоритма 6, которые использовались во всех экспериментов в данной работе.

- **Метрика:** $M = Accuracy$ — точность (процент верно предсказанных классов)
- **Количество блоков в кросс-валидации:** $K = 3$
- **Количество запусков кросс-валидаций:** $T = 20$
- **Уровень значимости:** $\alpha = 0.05$

В качестве базовых моделей для всех алгоритмов использовались решающие деревья. Важнейшим параметром для всех ансамблей является количество базовых алгоритмов. В своих исследованиях данный параметр выбирался из множества $\{1, 2, \ldots, 1000\}$.

Для алгоритмов бустинга (AdaBoost, Iterative Rotation AdaBoost) перебиралась максимальная глубина решающего дерева из множества $\{1, 3, 5, 7\}$. Для алгоритма Rotation Forest глубина дерева никак не фиксировалась, так как это противоречит идеи алгоритма, которая заключается в том, что базовые модели должны быть переобучены.

Для «rotation»-алгоритмов важным параметром является размер подмножеств, на которые делится все множество признаков. В данной работе был принят следующий подход: вместо прямого перебора различных размеров подмножеств ($\{2, 3, \ldots\}$) мы применяли различные целочисленные функции f к количеству признаков D, получая таким образом $M = f(D)$ максимальный размер случайных подмножеств. В экспериментах перебирались следующие функции:

1. $f(x) = \lfloor \log_2(x) \rfloor$;
2. $f(x) = \lfloor \sqrt{x} \rfloor$;
3. $f(x) = \lfloor \frac{x}{2} \rfloor$;
4. $f(x) = x$.

Таким образом последний случай эквивалентен повороту всего пространства одновременно.

Для бустинг-алгоритмов фиксированным был выбран темп обучения (learning rate, shrinkage) равный 0.01. А для rotation-алгоритмов был зафиксирован размер выборки $P = N$, которая используется в методе главных компонент.

Выборка	IterRotAdaBoost	AdaBoost	RotForest
Abalone	26.99	25.89	23.83
Balance Scale	85.99	86.31	87.58
Banknote Auth.	99.71	99.13	99.71
Blood Transfusion	79.14	78.61	72.19
Breast Cancer	96.50	95.10	97.20
Car Eval.	98.38	96.53	97.92
Chess	99.50	99.25	99.69
Cleveland Heart	58.00	56.00	53.33
Credit Approval	85.67	86.59	85.37
Digits	98.11	97.33	98.44
EEG	90.63	82.96	93.82
Ecoli	83.93	80.95	79.76
Forest Type	88.55	85.88	89.69
Glass	71.30	66.67	73.15
ILPD	68.97	70.34	70.34
Ionosphere	93.18	90.91	93.75
Iris	93.42	92.11	94.74
Letters	95.59	92.56	81.18
Mammographic	80.05	79.33	76.20
Musk	89.08	83.19	90.34
Occupancy	99.22	99.13	99.30
Pima	74.22	74.74	74.22
Sonar	79.81	73.08	82.69
Spectf	76.12	74.63	77.61
Teaching Assistant	52.63	47.37	60.53
Thoraric Surgery	84.32	84.32	83.47
Tic-Tac-Toe	98.75	97.08	98.75
Vertebral	80.77	80.13	79.49
Wine	96.67	92.22	97.78
Zoo	92.31	94.23	96.15

Таблица 2: Медиана точности для каждой тестовой выборки

3.3 Основной эксперимент. Сравнение Iterative Rotation AdaBoost, AdaBoost, Rotation Forest.

В данном разделе опишем главный эксперимент: сравнение трех алгоритмов Iterative Rotation AdaBoost, AdaBoost и RotationForest. В Таблице 2 приведены аггрегированные результаты алгоритмов по 20 запускам.

Кроме того в Таблице 3 представлено попарное сравнение алгоритмов. В позиции (i, j) кортеж (W, T, L) интерпретируется как количество побед W, ничей T, проигрышей L алгоритма из строки i над алгоритмом из столбца j. Напомним, что под победой (выигрышем) алгоритма A над алгоритмом B подразумевается то, что использование алгоритма A дает статистически значимое улучшение в точности. Под ничьей - отличия в точности алгоритмов являются статистически незначимыми. Проигрыш алгоритма A алгоритму B есть тоже самое, что выигрыш алгоритма B над алгоритмом A.

Алгоритмы	AdaBoost	RotForest
IterRotAdaBoost	(19, 11, 0)	(7, 14, 9)
AdaBoost	—	(4, 10, 16)

Таблица 3: Попарное сравнение алгоритмов Iterative Rotation AdaBoost, AdaBoost и Rotation Forest

Важнейшим результатом является то, что Iterative Rotation AdaBoost на 19 (из 30) выборках показал более высокую точность, и что более удивительно ни разу не оказался хуже чем оригинальный AdaBoost. Таким образом, теперь можно с уверенностью сказать, что идея поворотов признакового пространства, которая была перенесена из Rotation Forest и применена к алгоритму AdaBoost, позволяет улучшить качество классификации бустинга. Такое подавляющее превосходство достаточно впечатляет, если учесть, что оно достигается за счет поворотов в случайных подпространствах.

Проведенный эксперимент подтвердил результаты [4]: алгоритм Rotation Forest оказался лучше алгоритма AdaBoost (хотя это отличие не такое однозначное, как в случае с Iterative Rotation AdaBoost). Сравнение Iterative Rotation AdaBoost и Rotation Forest показало, что небольшой перевес на стороне последнего.

Подытожим результаты: предложенный в данной работе алгоритм Iterative Rotation AdaBoost показал впечатляющую точность, показав, что он не просто не хуже алгоритма AdaBoost, а в большенстве случае лучше. Ему не удалось выйти вперед алгоритма Rotation Forest, но отметим, что данный результат может быть обусловлен ограничением на глубину решающих деревьев и размером ансамбля.

3.4 Сравнение AdaBoost и AdaBoost с фиксированным поворотом признаков.

Предыдущий раздел продемонстрировал, что предложенный в данной работе алгоритм способен улучшить классический AdaBoost. Но, возможно, для достижения данного прироста совершенно не обязательно делать повороты в случайных подпространствах на каждой итерации бустинга. Чтобы показать, что внедрение поворотов в процедуру бустинга является существенным фактором для получения прироста в качестве мы проведем сравнение обычного AdaBoost и Iterative Rotation AdaBoost, в котором на каждом шаге реализуются фиксированное разбиение (одно и тоже на всех итерациях) и осуществляется повороты при помощи метода главных компонент, как было описано ранее. Последний алгоритм можно интерпретировать как обычный AdaBoost, но к входу которого применяют преобразование признаков, поэтому будем ссылаться на него как AdaBoost[FixedRotation]. Так как AdaBoost[FixedRotation] целиком зависит от разделения на подмножества признаков, которое будет оставаться фиксированным на протяжении всего обучения, необходимо осуществлять дополнительный перебор по различным разбиениям. Поэтому в наших экспериментах мы пробовали три разных разбиения и затем на кросс-валидации выбиралось наилучшее. Финальный результат как и всегда вычислялся на отложенной тестовой выборке.

В Таблице 4 представлены результаты эксперимента в виде попарных сравнений (Таблицу с медианами по каждой выборке можно найти в Приложении).

Алгоритмы	AdaBoost	AdaBoost[FixedRotation]
IterRotAdaBoost	(19, 11, 0)	(18, 12, 0)
AdaBoost	—	(2, 24, 4)

Таблица 4: Попарное сравнение алгоритмов Iterative Rotation AdaBoost, AdaBoost и AdaBoost[FixedRotation].

Отметим, что отличия AdaBoost от AdaBoost[FixedRotation] оказались не существенными: алгоритмы на 24 выборках из 30 показывают одинаковые результаты. В тоже время Iterative Rotation AdaBoost одинаково опережает как алгоритм AdaBoost так и алгоритм AdaBoost[FixedRotati Следовательно, можно заключить, что повороты признаков встроенные в процедуру бустинга играют ключевую роль в предложенном алгоритме.

3.5 Сравнение Iterative Rotation AdaBoost и его взвешенной версии.

В данном разделе перейдем сравнение двух вариантов предложенного алгоритма: первый — обычный Iterative Rotation AdaBoost и Iterative Rotation AdaBoost, в котором метод главных компонент заменен взвешенными аналогом, а в качестве весов используются веса объектов w_i. Взвешенную версию Iterative Rotation AdaBoost будем называть Iterative Rotation AdaBoost[Weights]. Изначально взвешенный вариант казался наиболее перспективным, так как в нем заложена дополнительная информация о важности каждого объекта. Но из результатов сравнения (Таблица 5) видно, что отличия между алгоритмами в подавляющем большинстве случаев являются статистически незначимыми.

Алгоритмы	IterRotAdaBoost[Weights]
IterRotAdaBoost	(2, 25, 3)

Таблица 5: Сравнение Iterative Rotation AdaBoost и Iterative Rotation AdaBoost[Weights]

3.6 Сравнение поворотов при помощи метода главных компонент и случайных поворотов.

В финале проведем сравнение стандартного Iterative Rotation AdaBoost, предложенного в разделе 2.1, которые использует повороты при помощи метода главных компонент и модификации Iterative Rotation AdaBoost, в котором на каждой итерации выполняются случайные повороты при помощи $\hat{r}_{M,P}$ преобразования (будем называть данную модификацию Iterative Rotation AdaBoost[Random]). Результаты данного эксперимента могут позволить понять на сколько важно использовать процедуру построения матрицы поворота, основанную на методе PCA или поворот может осуществляться из других соображений. Как и в разделе 3.4 для метода Iterative Rotation AdaBoost[Random] будем дополнительно перебирать случайные состояния (в данном эксперименте использовалось три различных состояния), которые будут определять вид случайных матриц поворота на каждой итерации.

Алгоритмы	IterRotAdaBoost[Random]
IterRotAdaBoost	(3, 25, 2)

Таблица 6: Сравнение Iterative Rotation AdaBoost и Iterative Rotation AdaBoost[Random]

Из результатов сравнения (Таблица 6) видно, что отличия между алгоритмами в подавляющем большинстве случаев являются статистически незначимыми. То есть мы приходим к выводу, что случайные повороты практически также «хороши», как и повороты при помощи метода главных компонент. Данный факт прежде всего показывает, что повороты позволяют обогатить признаковое пространство выборки, тем самым усиливая точность бустинга.

4. Полученные результаты

В данной работе были рассмотрены алгоритмы бустинга, а также повороты признакового пространства как способ, который позволяет улучшить классификацию. Основным результатом работы стал предложенный алгоритм — Iterative Rotation AdaBoost, который является специально модифицированным алгоритмом AdaBoost, на каждом шаге которого применяется преобразование признаков эквивалентное поворотам в подпространствах признакового пространства вдоль главных компонент. Данный подход позволяет усилить разнообразность моделей, упрощая классификацию для решающих деревьев.

Предложенный алгоритм был реализован на языке Python. Код реализованных алгоритмов можно найти по следующей ссылке: https://github.com/antongoy/RotationAlgorithms

При помощи сравнительных экспериментов было показано, что алгоритм Iterative Rotation AdaBoost в 19 из 30 раз показывает более высокую точность, чем классический AdaBoost, а в остальных 11 случаях отличия являются статистически не значимыми.

Кроме того, было показано, что применение поворотов на каждой итерации предложенного алгоритма, является существенным фактором для повышения точности, которого невозможно достичь используя применяя однократно преобразование $r_{M,P}$ в начале AdaBoost.

Также было продемонстрировано, что замена PCA на его взвешенный аналог в алгоритме Rotation AdaBoost не дает прироста в точности, что опровергло изначальную гипотезу, что веса объектов способны скорректировать поворот.

Было показано, что случайные повороты позволяют достичь прироста в точности для бустинга. Более того случайные повороты оказались сравнимы с поворотами при помощи метода главных компонент.

Возможные направления дальнейших исследований:

1. Метод главных компонент вычислительно трудоемкая операция, поэтому необходимо исследовать возможное ускорение, путем преобразования методом главных компонент не на каждом шаге бустинга, а лишь на некоторых без потери точности классификации.

2. Естественным направлением дальнейших исследований является обобщение предложенного метода к градиентному бустингу, который позволит применить предложенный в работе подход к произвольным функциям потерь.

3. Необходимо более детально исследовать другие преобразования признаков, а также опопробовать применить более интеллектуальный отбор признаков для поворота на каждой итерации (например, применять какую-либо меру информативности признаков и применять поворот только к ним).

4. AdaBoost сейчас почти не используется на практике (в том числе из-за того что не стабилен при наличии шумов в выборке). Важным направлением дальнейших исследований станет попытка применить ту же идею поворотов к градиентному бустингу, который является на сегодняшний день одним из самых мощных методов машинного обучения.

Список литературы

[1] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

[2] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Ann Arbor*, 1001(48109):1612, 2006.

[3] Breiman L., Friedman J. H., Olshen R. A., and Stone C. J. *Classification and regression trees.* Wadsworth & Brooks, 1984.

[4] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, Oct 2006.

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] Rico Blaser and Piotr Fryzlewicz. Random rotation ensembles. *Journal of Machine Learning Research*, 17(4):1–26, 2016.

[7] Chun xia Zhang and Jiang she Zhang. Rotboost: a technique for combining rotation forest and adaboost. *Pattern Recognition Letters*, pages 1524–1536, 2008.

[8] Ludmila I. Kuncheva and Juan J. Rodríguez. An experimental study on rotation forest ensembles. In *Proceedings of the 7th International Conference on Multiple Classifier Systems*, MCS'07, pages 459–468, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.

[10] M. Lichman. UCI machine learning repository, 2013.

[11] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

5. Приложение

Выборки	IterRotAB	IterRotAB[Weights]	IterRotAB[Random]	AB[Fixed]	AB	RotForest
Abalone	26.99	26.75	27.08	26.41	25.89	23.83
Balance.Scale	85.99	94.90	88.54	87.26	86.31	87.58
Banknote.Auth.	99.71	99.71	99.42	99.27	99.13	99.71
Blood.Transfusion	79.14	79.14	79.41	78.61	78.61	72.19
Breast.Cancer	96.50	96.50	96.50	95.10	95.10	97.20
Car.Eval.	98.38	98.73	98.61	97.45	96.53	97.92
Chess	99.50	99.56	99.56	99.12	99.25	99.69
Cleveland.Heart	58.00	56.67	56.00	54.00	56.00	53.33
Credit.Approval	85.67	85.37	85.37	85.06	86.59	85.37
Digits	98.11	98.11	98.22	97.45	97.33	98.44
EEG	90.63	90.67	89.84	82.58	82.96	93.82
Ecoli	83.93	83.33	83.93	80.36	80.95	79.76
Forest.Type	88.55	89.31	87.79	87.02	85.88	89.69
Glass	71.30	75.00	72.22	70.37	66.67	73.15
ILPD	68.97	70.00	70.34	66.90	70.34	70.34
Ionosphere	93.18	92.05	93.18	90.34	90.91	93.75
Iris	93.42	93.42	94.74	93.42	92.11	94.74
Letters	95.59	95.33	95.31	92.60	92.56	81.18
Mammographic	80.05	80.05	80.53	79.09	79.33	76.20
Musk	89.08	88.66	89.50	84.87	83.19	90.34
Occupancy	99.22	99.22	99.22	99.12	99.13	99.30
Pima	74.22	75.26	76.04	74.74	74.74	74.22
Sonar	79.81	76.92	80.77	76.92	73.08	82.69
Spectf	76.12	76.12	76.12	74.63	74.63	77.61
Teaching.Assistant	52.63	56.58	50.00	50.00	47.37	60.53
Thoraric.Surgery	84.32	83.90	83.90	83.05	84.32	83.47
Tic.Tac.Toe	98.75	98.75	98.75	98.12	97.08	98.75
Vertebral	80.77	80.77	82.69	77.56	80.13	79.49
Wine	96.67	95.56	95.56	90.00	92.22	97.78
Zoo	92.31	92.31	96.15	92.31	94.23	96.15

Таблица 7: Медианы точности по 20 запускам для всех алгоритмов, которые участвовали в экспериментах. Для краткости AdaBoost сокращено до AB.

Методы	IterRotAdaBoost[Weights]	IterRotAdaBoost[Random]	AdaBoost[Fixed]	AdaBoost	RotForest
IterRotAdaBoost	(2, 25, 3)	(3, 25, 2)	(18, 12, 0)	(19, 11, 10)	(7, 14, 9)
IterRotAdaBoost[Weights]		(4, 25, 1)	(21, 9, 0)	(17, 13, 0)	(8, 16, 6)
IterRotAdaBoost[Random]			(18, 12, 0)	(18, 11, 1)	(9, 16, 5)
AdaBoost[Fixed]				(4, 24, 2)	(4, 12, 14)
AdaBoost					(4, 10, 16)

Таблица 8: Попарные сравнения всех алгоритмов, которые участвовали в экспериментах