МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ИВТИ
Кафедра МКМ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

Направление 01.03.02 Прикладная математика и информатика
(код и наименование)

Образовательная программа Математическое моделирование

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Исследование методов машинного обучения в задаче предсказания результатов матчей на профессиональной сцене в Counter-Strike: Global Offensive

Студент А-14-18 _____ Рылов Д.С.
группа подпись фамилия и инициалы

Руководитель ВКР к.т.н. доцент _____ Князев А.В.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Внешний консультант
уч. степень должность подпись фамилия и инициалы

организация

«Работа допущена к защите»

Заведующий кафедрой к.ф.-м.н. доцент _____ Зубков П.В.
уч. степень звание подпись фамилия и инициалы

Дата

Москва, 2022

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ИВТИ
Кафедра МКМ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(БАКАЛАВРСКУЮ РАБОТУ)

Направление 01.03.02 Прикладная математика и информатика
(код и наименование)

Образовательная программа Математическое моделирование

Форма обучения очная
(очная/очно-заочная/заочная)

Тема: Исследование методов машинного обучения в задаче предсказания результатов матчей на профессиональной сцене в Counter-Strike: Global Offensive

Студент А-14-18 _____ Рылов Д.С.
группа подпись фамилия и инициалы

Руководитель ВКР к.т.н. доцент _____ Князев А.В.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Внешний консультант
уч. степень должность подпись фамилия и инициалы

организация

Заведующий кафедрой к.ф.-м.н. доцент _____ Зубков П.В.
уч. степень звание подпись фамилия и инициалы

Место выполнения работы ФГБОУ ВО «НИУ «МЭИ»

СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

1) Изучение литературы.
2) Изучение линейных методов классификации.
3) Изучение методов классификации, основанных на деревьях решений.
4) Изучение методов классификации, основанных на многослойных нейронных сетях.
5) Построение системы сбора данных.
6) Изучение методов преобразования данных.
7) Применение методов классификации к полученным данным.
8) Анализ результатов.
9) Подготовка выпускной работы.

ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

Количество листов _____

Количество слайдов в презентации _____

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1) Николенко, С.И. Глубокое обучение / С.И. Николенко, А. А. Кадурин – С-П: издательство Питер, 2017. – 480 с.
2) Рашка, С. Python и машинное обучение. Машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow. С. Рашка, В. Мирджалили – М: Вильямс, 2019. – 659 с.

Примечания:

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

Аннотация

Данная работа посвящена актуальной проблеме сравнения методов машинного обучения в задаче предсказания результатов матчей на профессиональной сцене в Counter-Strike: Global Offensive. Дано подробное описание различных методов классификации, проблем этих методов, способы их решения, а также различных трудностей связанных с ошибками (выбросами) в данных и способов их удаления. В ходе работы были проведены исследования различных подходов к обработке и классификации данных.

Abstract

This work is devoted to the actual problem of comparing machine learning methods in the task of predicting the results of matches on the professional stage in Counter-Strike: Global Offensive. A detailed description of the various classification methods, the problems of these methods, how to solve them, as well as the various difficulties associated with errors (outliers) in the data and how to remove them is given. In the course of the work, studies of various approaches to data processing and classification were carried out.

Введение

Искусственный интеллект и машинное обучение – крайне популярная область исследований, интерес к которой только увеличивается на протяжении последних десятилетий.

Искусственный интеллект определяется, как способность компьютера имитировать когнитивные функции человека, такие как, обучение и решение проблем. Машинное обучение – это способ развития интеллекта компьютера.

Машинное обучение используется в различных областях: распознавание изображений, обработка естественного языка и т.д. Одной из самых популярных задач является задача классификации табличных данных. При решении каждой конкретной задачи из этой области используются различные методы машинного обучения, т.к. у каждой такой задачи есть свои особенности. Например, для решения задачи кредитного скоринга, которая считается классической, используются легко интерпретируемые методы для того, чтобы банковскому работнику, было понятно, из-за каких критериев кредит не был одобрен.

Задача предсказания итогов матчей на профессиональной сцене в CS:GO популярна, т.к. игрой интересуется большое количество людей (прямую трансляцию финала последнего международного чемпионата смотрело 2.5 млн человек). Также этой задачей интересуются различные букмекеры. Цель данной работы сравнить различные методы машинного обучения и разные подходы к обработке информации, чтобы получить максимально точную предсказательную модель.

Настоящая пояснительная записка состоит из 3 глав, имеющих следующее содержание:

- Глава 1 содержит общие сведения о моделях машинного обучения, предназначенных для бинарной классификации. Приведены описания их работы, а также сильные и слабые стороны. Рассмотрены такие модели, как линейная модель с ошибкой перцептрона, метод опорных векторов, логистическая регрессия, решающее дерево, случайный лес, градиентный бустинг и многослойная нейронная сеть.
- Глава 2 содержит описание данных используемых для сравнения методов, а также способов их получения и преобразования.
- Глава 3 содержит подробное описание исследований и их анализ. Здесь приведены примеры всех данных, используемых для сравнения, а также результаты предсказания моделей. В конце проведен анализ результатов.
- В приложении приведен листинг программы.

1. Бинарные классификаторы

Постановка задачи бинарной классификации:
X – множество *объектов*.
Y – множество *ответов*.
$Y = \{-1,1\}$
$y: X \to Y$ – неизвестная зависимость.

Дано:

$\{x_1, \dots, x_l\} \subset X$ – обучающая выборка.
$y_i = y(x_i),\ i = 1, \dots, l$ – известные ответы.

Найти:

$a: X \to Y$ – функцию аппроксимирующую y на всем множестве X.

1.1. Линейные классификаторы

Пусть наши объекты задаются векторами, т.е. каждый объект представляет из себя вектор из $\mathbb{R}^D$, где D – количество признаков нашего объекта. Тогда, если мы говорим про линейные модели, мы должны выбрать наилучшую функцию, представимую в виде $y_i = sign\langle w_i, x_i \rangle$, или же, другими словами, нам нужно найти наилучшую разделяющую гиперплоскость, для которой w является направляющим вектором. Пример такой гиперплоскости можно увидеть на рисунке 1.1

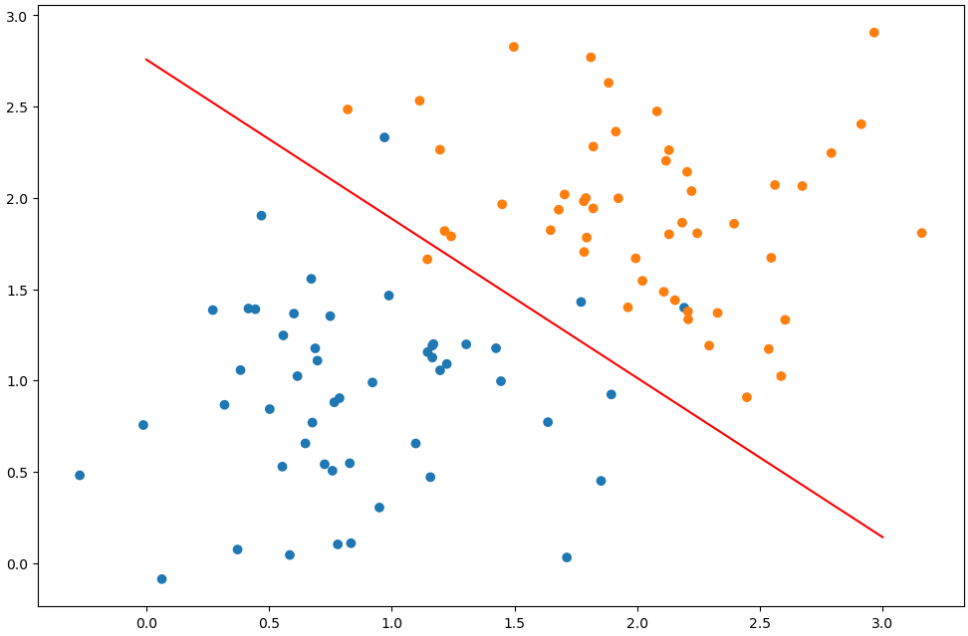

Рисунок 1.1

Создадим теперь функционал ошибки. Мы хотим минимизировать число ошибок классификатора, т.е.

$$\sum_i \mathbb{I}[y_i \neq sign\langle w_i, x_i\rangle] \to \min \text{ по } w$$

Упростим:

$$\sum_i \mathbb{I}[y_i\langle w_i, x_i\rangle < 0] \to \min \text{ по } w$$

Величина $y_i\langle w_i, x_i\rangle$ называется отступом (margin) и обозначается M. Легко увидеть, что:

- Когда отступ положителен, то $sign\langle w_i, x_i\rangle = sign(y_i)$, то есть класс предсказан верно; чем больше отступ, тем дальше от разделяющей гиперплоскости находится предсказываемый объект, что значит большую “уверенность классификатора”.
- Когда отступ отрицателен, то $sign\langle w_i, x_i\rangle \neq sign(y_i)$, то есть класс предсказан неверно; чем больше отступ, тем сильнее ошибается классификатор.

Для каждого отступа вычислим функцию:

$$F(M) = \mathbb{I}[M < 0] = \begin{cases} 1, M < 0 \\ 0, M \geq 0 \end{cases}$$

Тогда наша функция, которую мы пытаемся минимизировать будет выглядеть так:

$$\sum_i F(M_i) \to min$$

Она кусочно-постоянная, т.е. мы не можем использовать градиентные методы, т.к. производная нашей функции равна нулю во всех точках, где она существует. Что бы решить эту задачу давайте заменим нашу функцию потерь на какую-либо другую дифференцируемую функцию, такую, что бы:

$$\sum_i F(M_i) \leq \sum_i L(M_i) \to min$$

В качестве функции L можно использовать различные функции, примеры которых приведены на рисунке 1.2.

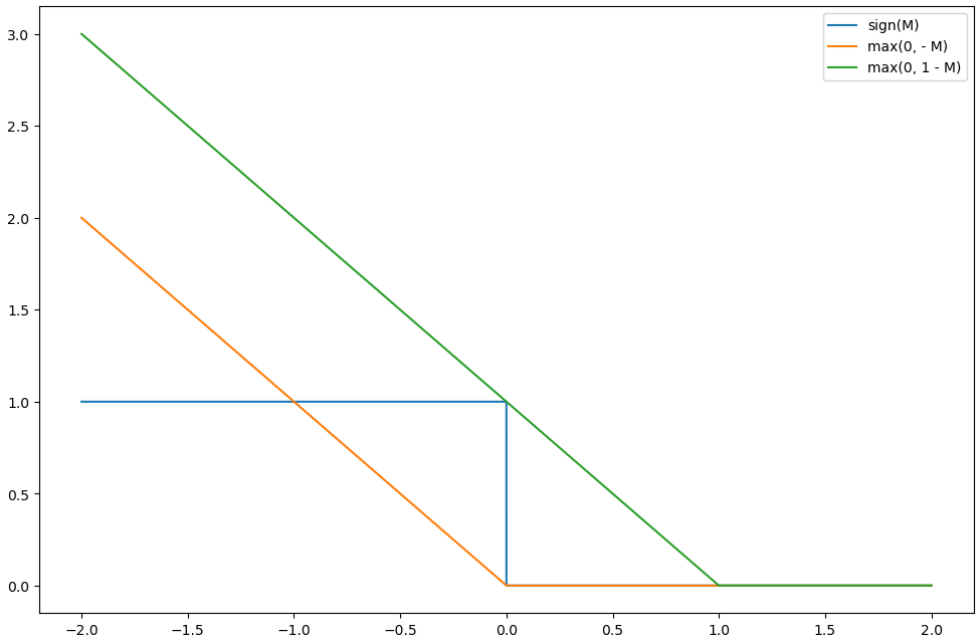

Рисунок 1.2

1.1.1. Ошибка перцептрона

Идея перцептрона проста: давайте считать отступы только на неправильно классифицированных объектах и учитывать не просто наличие ошибки, а линейно, относительно величины отступа. Получается такая функция:

$$F(M) = \max(0, -M)$$

Т.е. мы можем записать весь функцию ошибок в таком виде:

$$L(w, x, y) = \lambda \|w\|_2^2 + \sum_i \max(0, -M_i)$$

Запишем сразу с L^2 регуляризацией. Регуляризация бывает 2-х типов: L^2 и L^1. L^2 штрафует алгоритм за слишком большие веса, а L^1 обнуляет слишком малые.

Найдем градиент для этой функции потерь:

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, M_i > 0 \\ -y_i x_i, M_i \leq 0 \end{cases}$$

Получив аналитическую функцию для градиента, мы можем использовать градиентные методы (стохастический или обычный градиентный спуск) и при помощи этого метода решить задачу.

У линейного перцептрона есть недостаток, его решение не единственно и зависимо от начальных условий. Пример можно увидеть на рисунках 1.3.

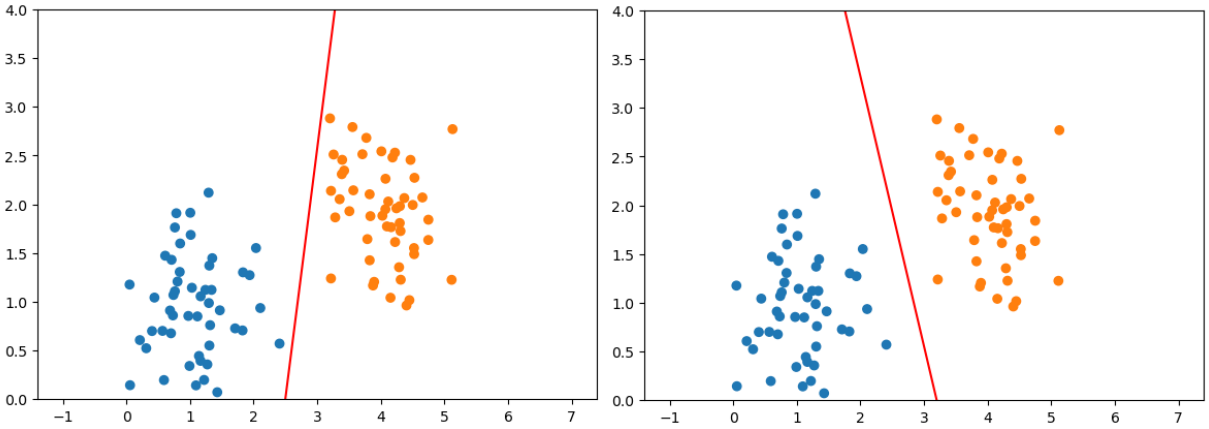

Рисунок 1.3

Это одна и та же выборка, но из-за различного выбора начальных условий ответ различен. Давайте перейдем к методу, который компенсирует этот недостаток.

Впервые эта функция потерь была предложена для перцептрона Розенблатта.

1.1.2. Линейный метод опорных векторов

Для таких ситуаций, как на изображениях выше, появляется желание максимально отодвинуть разделяющую гиперплоскость от обоих классов. Или, говоря другими словами, необходима найти оптимальную разделяющую гиперплоскость, как на рисунке 1.4

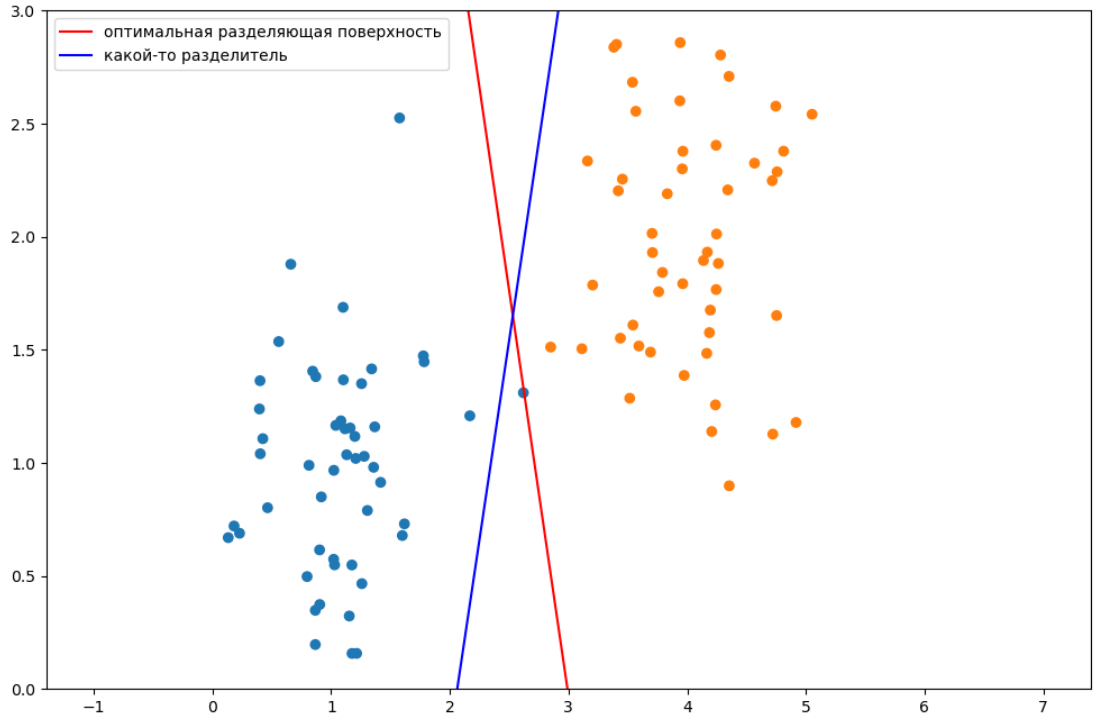

Рисунок 1.4

Для этого поменяем предыдущую функцию ошибок на одном объекте таким образом:

$$F(M) = \max(0,1 - M)$$

Тогда мы можем записать общую функцию ошибок в таком виде:

$$L(w, x, y) = \lambda \|w\|_2^2 + \sum_i \max(0,1 - M_i)$$

Здесь он записан сразу с регуляризацией.

Найдем градиент для этой функции потерь:

$$\nabla_w L(w, x, y) = 2\lambda w + \sum_i \begin{cases} 0, 1 - M_i < 0 \\ -y_i x_i, 1 - M_i \geq 0 \end{cases}$$

Давайте объясним эту функцию. На интуитивном уровне это можно понять так: объекты, которые классифицированы не очень уверено, т.е. $0 \leq M_i < 1$. Продолжают влиять на градиент, продолжая «отодвигать» разделяющую гиперплоскость от себя.

Давайте обсудим это в более строгом виде, для этого нам потребуется взглянуть на наше выражение под другим углом, для этого необходимо взглянуть на рисунок 1.5:

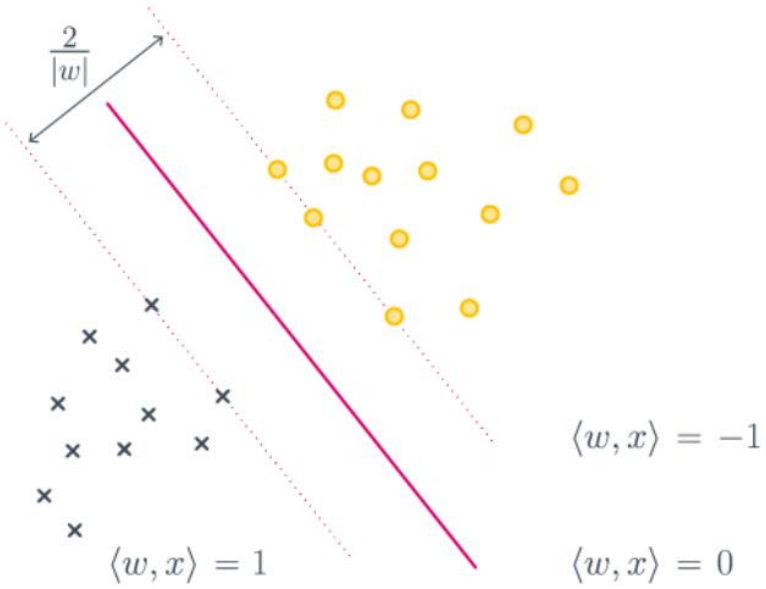

Рисунок 1.5

Что бы максимизировать отступ, нам нужно максимизировать $\frac{2}{|w|}$, т.е. мы увеличиваем ширину полосы при условии, что большая часть объектов предсказываются верно, а это эквивалентно решению начальной задачи:

$$\lambda\|w\|_2^2 + \sum_i \max(0,1 - M_i) \to min$$

Поясним наш вывод. Первое слагаемое обратно ширине полосы, но мы его минизируем. Второе слагаемое соответсвует штрафу за неправильно классифицированные объекты.

Конечное положение разделяющей гиперплоскости всего лишь несколькими наиближайшими к плоскости правильно классифицированными примерами, которые называются опорными векторами.

Этод метод называется **методом опорных векторов** или **support vector machine (SVM)**. До появления методов, основанных на деревьях решений, был одним из сильнейших методов.

Еще одним из важнейших плюсов SVM является достаточно простая модификация, созданная при использование ядерного метода, которая позваляет создавать не линейные разделяющие поверхности.

Это уже не совсем линейный классификатор, но, т.к. он вытекает из линейного метода опорных векторов, его нужно обсудить здесь.

1.1.3. Нелинейный метод опорных векторов

Нелинейный метод опорных векторов – метод машинного обучения, основанный на использовании метода опорных векторов после применения ядерного метода.

Ядерный метод (kernel trick) – метод в машинном обучении, позволяющий перевести элементы для случая линейной неразделимости в новое, линейно разделимое пространство, большей размерности, но, при этом, работать в исходном пространстве. Пример изображен на рисунке 1.6.

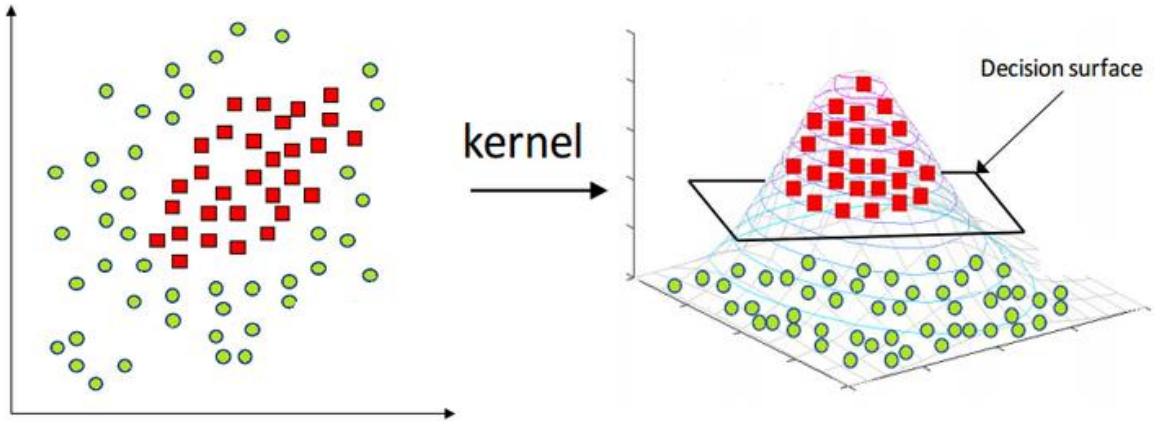

Рисунок 1.6

Нелинейный метод опорных векторов основан на предположение, что существует пространство большей размерности, в котором объекты различных классов линейно разделимы. Для каждой конкретной задачи это предположение может как выполняться, так и нет, потому что у нас могут быть просто неразделимые классы. Основным недостатком данного предположения является переход в пространство большей размерности, так как существенно усложняет вычисления, а в некоторых пространствах (бесконечно мерных), так и вовсе приводит к отсутствию возможности что-либо вычислять.

Как раз эта проблема и решается ядерным методом. Вместо того, чтобы строить пространство H, в которое будут отображаться объекты из исходного пространства. Мы используем функцию $K(x, x') = \langle \varphi(x), \varphi(x') \rangle$, такую, что $\varphi(x): X \to H$. А так как для решения задачи о линейной разделимости объектов нам достаточно скаляров, то нам достаточно пользоваться ядром, а не переходить в пространство большей размерности.

Ядро мы можем придумать и сами, но, т.к. метод известен давно, существуют несколько часто используемых ядер, зарекомендовавших свою эффективность:

1. Линейное ядро: $K(x, x') = \langle x, x' \rangle$
2. Полиномиальное ядро: $K(x, x') = (\langle x, x' \rangle + 1)^d$
3. Сигмоидное ядро: $K(x, x') = th(k_1 \langle x, x' \rangle - k_0)$
4. RBF ядро: $K(x, x') = \exp(-\gamma \|x - x'\|^2)$

На рисунке 1.7 изображено сравнение границ, построенных методом опорных векторов с разными ядрами.

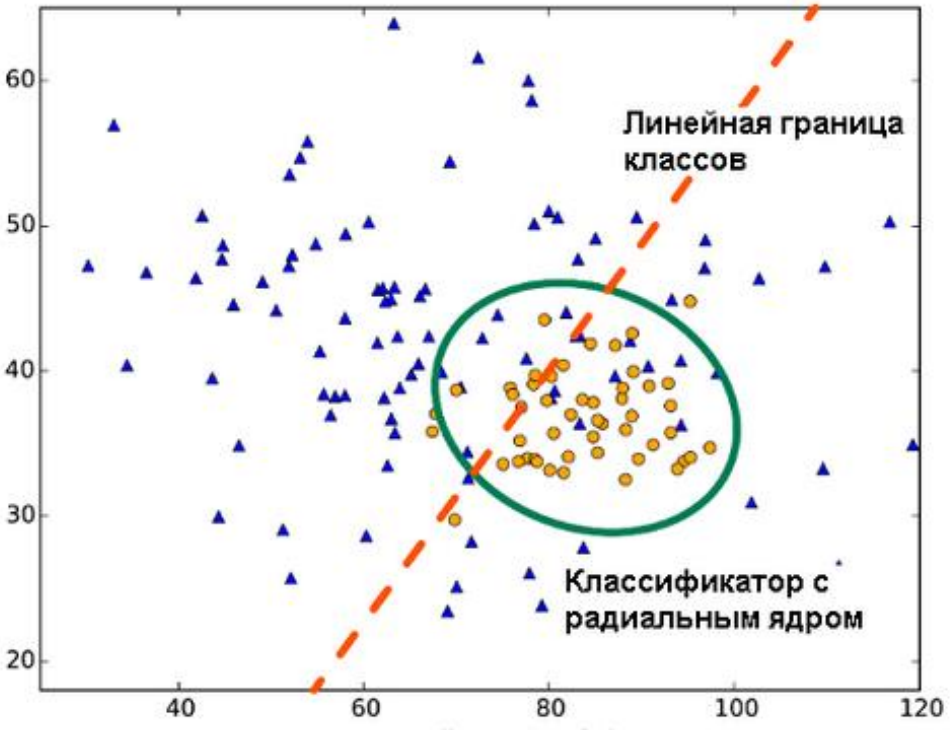

Рисунок 1.7

Скорость и проста нелинейного метода опорных векторов являются одной из причин крайней популярности этого метода в прошлом, а также использованием для решения определенного класса задач в настоящем.

1.1.4. Логистическая регресия

Логистическая регрессия появилась из желания рассмотреть задачу бинарной классификации, как задачу предсказания не самих классов, а их вероятностей.

Как же это сделать? Давайте, для начала, обозначим наши классы не $\{1, -1\}$, а $\{0,1\}$. Следующим появляется вопрос, как заставить линейную модель выдавать

числа из промежутка [0,1], а не (-∞, +∞). Ответ на этот вопрос кроется в log odd, или в логарифме шансов:

$$\log(\frac{p}{1-p}).$$

Если наша линейная модель будет выдавать числа, соответствующие логарифму шансов, то вероятность из него получается достаточно легко:

$$p = \frac{1}{1 + e^{\langle w, x_i \rangle}}$$

Такая функция называется сигмоидой:

$$\sigma(z) = \frac{1}{1 + e^{z}}$$

Функция потерь для данного метода вычисляется через метод максимального правдоподобия, применённый к распределению Бернулли. Итоговая функция потерь выглядит таким образом:

$$L(w, X, y) = -\sum_{i} (y_i \log\big(\sigma(\langle w, x_i \rangle)\big) + (1 - y_i) \log\big(\sigma(-\langle w, x_i \rangle)\big)$$

Её же градиент равен:

$$\nabla_w L(w, x, y) = -\sum_{i} x_i (y_i - \sigma(\langle w, x_i \rangle))$$

Что позволяет нам, решать эту задачу уже известным нам стохастическим градиентным спуском.

Несмотря на то, что линейные методы были необходимы в начале истории искусственного интеллекта, сейчас они используются в редких случаях со специфичными задачами, например, где важна простота интерпретации или же, где данных слишком много и все остальные методы не способны их решить из-за чрезмерного времени. В остальном же части теории из линейных методов получили свое развитие в более сложным моделях машинного обучения.

1.2. Классификаторы, основанные на деревьях решений

1.2.1. Решающее дерево

Решающее дерево предсказывает класс объекта, последовательно применяя простые решающие правила (предикаты). Этот процесс, а определенном роде, согласуется естественным для человека способом принятия решений.

Пример построенного решающего дерева можно увидеть на рисунке 1.8.

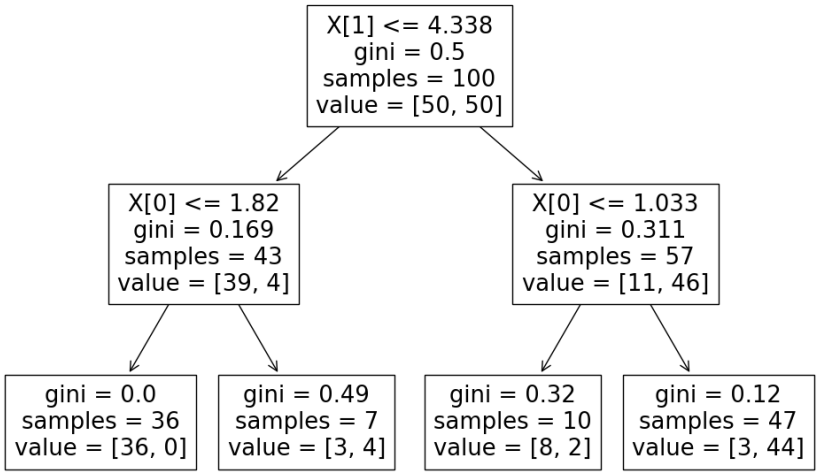

Рисунок 1.8

Обобщающая способность решающих деревьев невысока, но из-за их простоты их часто используют, как части сложных алгоритмов с более высокой обобщающей способностью, например, случайны лес или градиентный бустинг.

Дадим определение решающего дерева:
Решающим деревом называется такое бинарное дерево, что:

- Каждой внутренней вершине v приписан предикат: $B_v, X \to \{0,1\}$. (обычно это просто взятие порога: $B(x,j,t) = [x_j \leq t]$
- Каждой листовой вершине v приписан прогноз $c_v \in \{-1,1\}$

В ходе предсказания для каждый объекта выборки x проходит от вершины до какого-либо листа, в котором ему присваивается определенный класс. В каждой внутренней вершине v, проход продолжится вправо, если $B_v(x) = 1$, и влево, если $B_v(x) = 0$.

Особенности решающего дерева:

- Полученная функция является кусочно-постоянной, т.е. мы не сможем использовать градиентные способы оптимизации.
- Дерево в отличии от линейных методов не способно экстраполировать зависимость обучающей выборки за её пределы.
- Дерево может идеально приблизить обучающую выборку и при этом ничего не выучить, т.е. необходимо пытаться сделать дерево как можно более простым, чтобы не потерять его обобщающую способность.

Построение оптимального решающего дерева является NP-полной задачей, так что мы будем строить не оптимальное, а достаточно хорошее решающее дерево, для этого используется «жадный алгоритм»:

Пусть X – множество всех объектов обучающей выборки, а X_m – множество объектов, попавших в вершину.

- Создаем вершину v.
- Если выполняется условие остановки $Stop(X_m)$, то объявляем эту вершину листом и ставим ей в соответствие ответ $Ans(X_m)$.
- Иначе находим наилучших предикат (сплит) $B_{j,t}$, максимизирующий критерий ветвления $Branch(X_m, j, t)$, который разбивает выборку на две: X_l, X_r.
- Повторяем рекурсивно для X_l, X_r.

Обсудим вспомогательные функции:

- $Stop(X_m)$ – функция, решающая, стоит ли продолжать ветвление или стоит остановится. Обычно зависит от двух параметров: количество элементов в вершине и от количества элементов одного класса.
- $Ans(X_m)$ – функция, вычисляющая ответ для листа по попавшим в него элементам выборки. Самым простым способом является выбрать наиболее частый класс элементов.
- $Branch(X_m, j, t)$ – функция, измеряющая, насколько хорошо предлагаемое разбиение. Обычно считается через энтропию или критерий Джини.

Обобщая все вышесказанное можно назвать основные плюсы и минусы решающих деревьев. Плюсами является высокая скорость их построения и легкость предсказания, минусами же можно назвать низкую обобщающую способность и сильнейшую тягу к переобучению.

1.2.2. Случайный лес

Идея случайного леса вытекает из идеи бэггинга (**bagging, bootstrap aggregation)** над случайными деревьями. Бэггинг – метод машинного обучения, смысл которого заключается в следующем:

- Выберем из обучающей выборки $\boldsymbol{X}$ n примеров. Выбираем равновероятно с повторением.
- Повторим операцию k раз. Получим k выборок размера n.

- На каждой выборке обучим модель.
- Для получения предсказания будем использовать усредненное значение всех моделей. Для задачи бинарной классификации можно использовать самое частое значение.

Алгоритм построения случайного леса:

- Выберем из обучающей выборки $\boldsymbol{X}$ выборку $\boldsymbol{X^i}$. Выбираем равновероятно с повторением.
- Выберем случайно $\boldsymbol{n < N}$ признаков (метод случайных подпространств). Делается это для уменьшения зависимости между базовыми алгоритмами.
- Повторим операцию k раз. Получим k выборок из n признаков.
- На каждой выборке обучим решающее дерево.
- Для получения предсказания будем использовать самое частое значение.

Пример алгоритма построения случайного леса можно рассмотреть на рисунке 1.9.

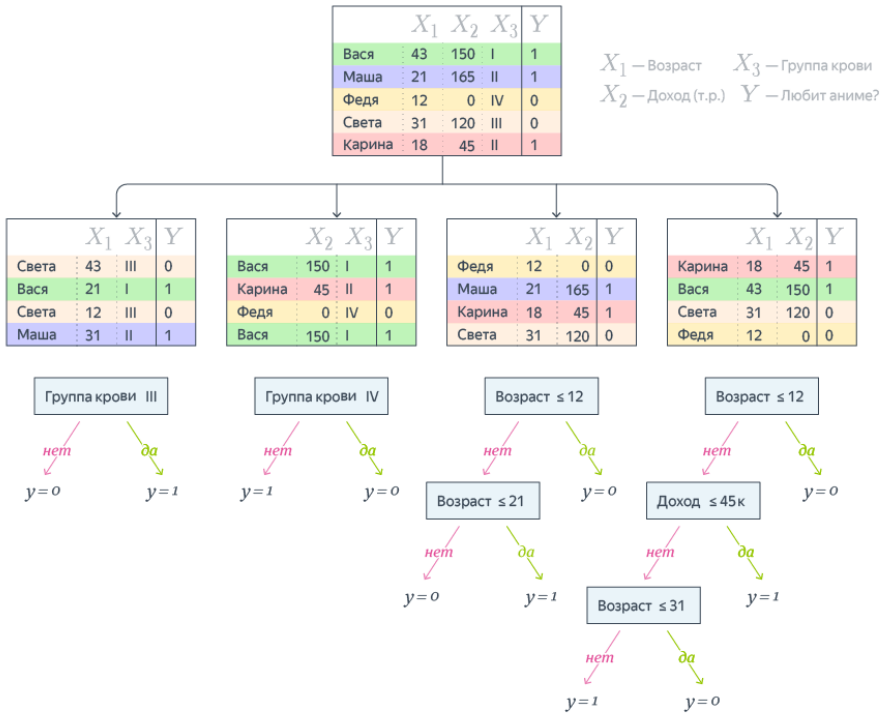

Рисунок 1.9

Если обобщить всё вышесказанное мы получим такие выводы о плюсах и минусах решающих деревьев:

Плюсы решающих деревьев:

- Высокая точность предсказания. Данный алгоритм до сих пор является используемым и востребованным.
- Низкая чувствительность к выбросам и масштабу данных.
- Легко распараллеливается, что ускоряет вычисления. Можно строить каждое дерево по-отдельности.

Минусы:

- Не умеет экстраполировать результаты. В отличие от линейных методов при получении для предсказания объекта, выходящего за границы обучающей выборки алгоритм присвоит ему метку самого крайнего класса, что в общем случае может быть не верно.
- Сложность в интерпретировании результатов.

1.2.3. Градиентный бустинг

Градиентный бустинг – алгоритм, рожденный из идеи, последовательного обучения моделей, для минимизации функционала потерь.

Модель градиентного бустинга выглядит так:

$$H_T(x) = \sum_{t=1}^{T} b_t h(x, a_t)$$

Опишем построение модели градиентного бустинга:

Входные данные:

- (X, y).
- Число итераций/базовых алгоритмов *M*.
- Функция потерь $L(H(X), y)$ и её градиент. В задаче бинарной классификации обычно используется логистическая функция потерь, описанная в параграфе 1.1.4.
- Базовые алгоритмы $h(x, a_t)$. В нашем случае это решающие деревья.
- Гиперпараметры базовых алгоритмов, в нашем случае – это глубина дерева, минимальное количество элементов в вершине для разбиения и т.д.

Договоримся, что под функцией $LEARN(X, y)$ будем понимать обучение базового алгоритма. В случае, когда базовыми алгоритмами являются деревья решений под $LEARN(X, y)$ будем понимать построение дерева при помощи «жадного алгоритма»

Алгоритм построения:

1. Обучим первый базовый алгоритм: $H_0(x) = LEARN(X, y)$, присвоим

2. For t = 1 to T do:
3. Находим градиент функции потерь:

$$\nabla L^{(t-1)} = \left[\frac{\delta L(H_{t-1}, y)}{\delta H_{t-1}}(x_i)\right]_{i=1}^{|X|},$$

   который представляет из себя вектор, длиной с нашу выборку.
4. Обучим следующий базовый алгоритм минимизирующий функцию потерь:

$$h(x, a_t) = LEARN\left(X, \nabla L^{(t-1)}\right)$$

5. Найдем коэффициент при нём:

$$b_t = argmin_b \sum_{i=1}^{|X|} L(y_i, H_{t-1}(x_i) + bh(x, a_t)),$$

   Обычно, этот коэффициент ищется при помощи МНК.
6. $H_t = H_{t-1}(x_i) + b_t h(x, a_t)$
7. Получили модель H_T

Рассмотрим подробнее пункт 5:

Достаточно часто вместо оптимизации параметра b_t, подставляется какая-либо константа, называемая темпом обучения (learn rate), это гиперпараметр, который настраивается вручную.

Данный алгоритм является одним из наиболее эффективных алгоритмов для работы с табличными данными.

Пример работы алгоритма:

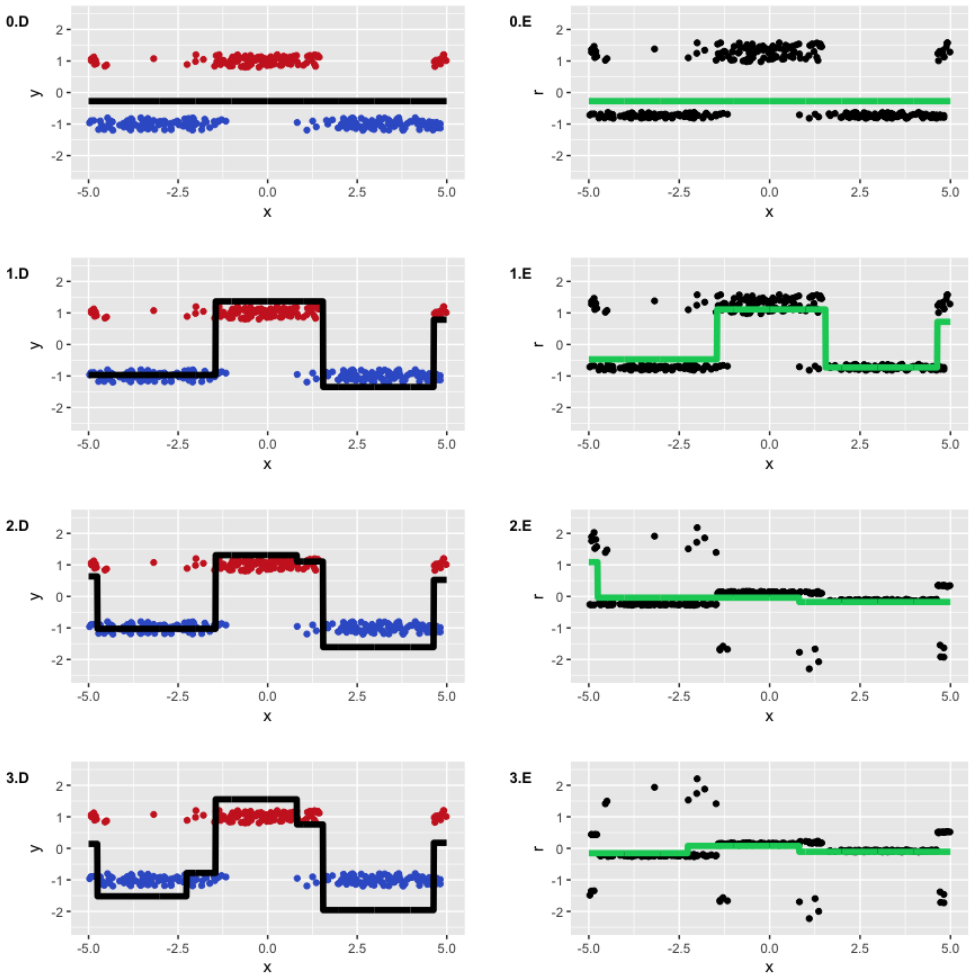

Рисунок 1.10

На графиках, изображенных на рисунке 1.10 слева актуальное приближение $H_t(x)$, а на графике справа новые базовые алгоритмы, построенные на градиенте.

Алгоритм градиентного бустинга над решающими деревьями является одной из сильнейших моделей для решения задач на табличных данных. Единственные, кто может с ней поспорить в эффективности, это нейросетевые методы.

1.3. Классификаторы, основанные на нейронных сетях

1.3.1. Общие сведения о многослойных нейронных сетях

Искусственная нейронная сеть – это сложная дифференцируемая функция, задающая отображение из пространства признаков в пространство ответов. Все параметры сети могут настраиваться одновременно и взаимосвязано.

В самом часто встречающемся случае представляет из себя последовательность дифференцируемых параметрических преобразований.

Первым идею искусственной нейронной, похожей на естественную, предложил Алан Тьюринг в 1948 году. Первой же реализацией искусственной нейронной сети является перцептрон Розенблатта, сделанный в 1960 году.

Обычно, нейросеть представляет ацикличный направленный граф, где вершинами являются нейроны. Моделирование одного нейрона показано на рисунке 1.11.

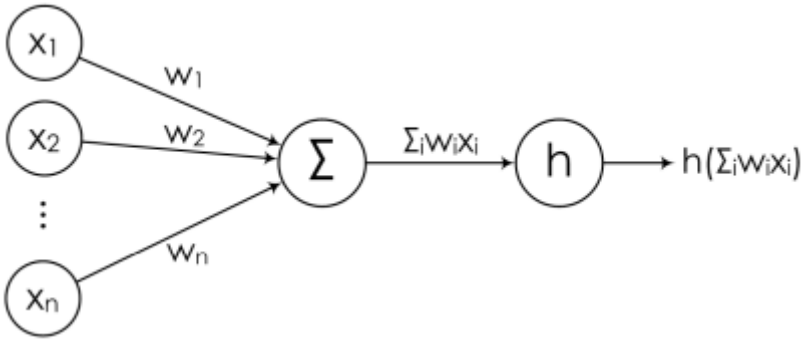

Рисунок 1.11

$X_1, X_2, \dots, X_n$ – входы (это могут быть и выходы других нейронов)
$w_1, w_2, \dots, w_n$ – веса/параметры.
$h(\sum_i w_i x_i)$ – нелинейная функция активации.

Функция активации обязательно должна быть нелинейной, т.к. в противном случае, сколько бы слоев мы не сделали, в итоге мы получим линейную функцию, из-за того, что линейная комбинация линейных отображений есть линейное отображение.

Если суммировать все вышесказанное мы получим такое изображение для нейронной сети, как показано на рисунке 1.12:

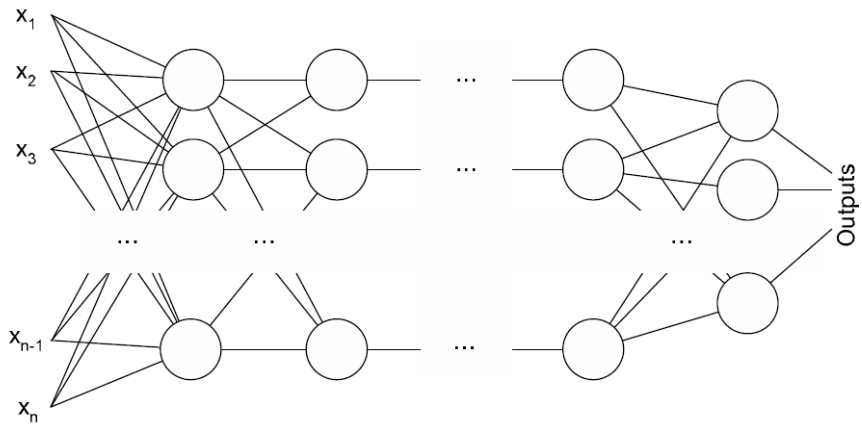

Рисунок 1.12

В задаче бинарной классификации, обычно, в выходном слое используется уже известная нам сигмоида. Благодаря этой функции мы получаем значения вероятностей принадлежности к классу 1.

Есть великое множество функций активации, но обычно используются такие, как показано на рисунке 1.13.

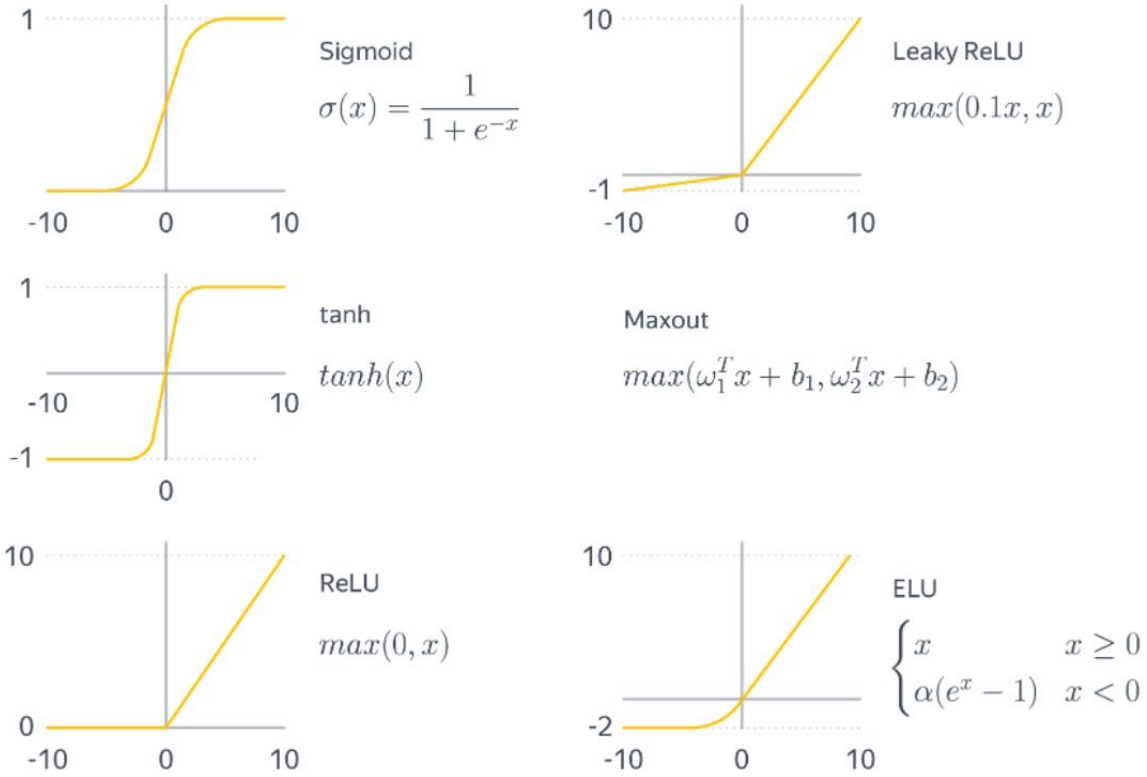

Рисунок 1.13

Обсудим только сигмоиду и ReLU, т.к. остальные являются либо их воплощениями под конкретные задачи, либо практически не используются.

Сигмоида:

Одна из первых функций активации.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

На практике часто используется в выходном нейроне, для решения задачи бинарной классификации и редко используется во внутренних слоях нейронных сетей из-за ряда недостатков:

- Вычислительная сложность.
- Близость производной к нулю на «хвостах» сигмоиды, что может привести к затуханию градиента.

- В целом малое максимальное значение производной, равное 0.25, что также может привести к затуханию градиента.

ReLU, Rectified linear unit:

$$ReLU(x) = \max(0, x)$$

ReLU является одной из наиболее популярных функций активаций внутри нейронных сетей из-за 2 основополагающих плюсов:

- Простота вычисления, что функции активации, что производной.
- Высокая эффективность данной функции, полученная практическим путем.

1.3.2. Обучение нейронных сетей

Нейронная сеть – это большая сложная функция, являющаяся композицией различных нелинейных преобразований (тех самых функций активаций). Вся суть обучения сводится к подбору наилучших параметров/весов нашей сети. Из-за того, что функция может быть не выпуклой, а чаше всего такой и является, самым действенным способом оптимизации параметров признан градиентный спуск или его младший собрат, стохастический градиентный спуск.

Рассмотрим проблему, которые возникают при этом подходе и способ ее решения, называемый, методом обратного распространения ошибки.

Для примера возьмем простую функцию:

$$f(x, y) = x^2 + xy + (x + y)^2$$

Представим ее в виде композиции простых функций, также называемой вычислительным графом, продемонстрированном на рисунке 1.14:

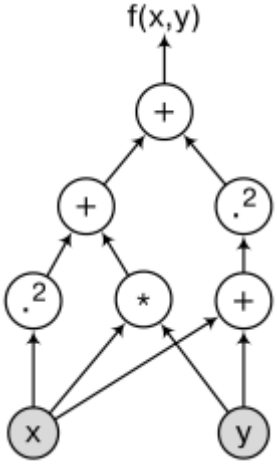

Рисунок 1.14

Что бы найти оптимальные значения для x, y мы итерационно должны находить градиент нашей функции по этим параметрам и приближаться к

истинному решению от начального приближения. Для вычисления градиента по параметрам, в нашем случае это переменные *x, y* существует 2 способа. Обсудим сначала неверный, он же метод прямого распространения (forward propagation, fprop). В этом методе производная сложной функции считается индукционно снизу графа, как это показано на рисунке 1.15:

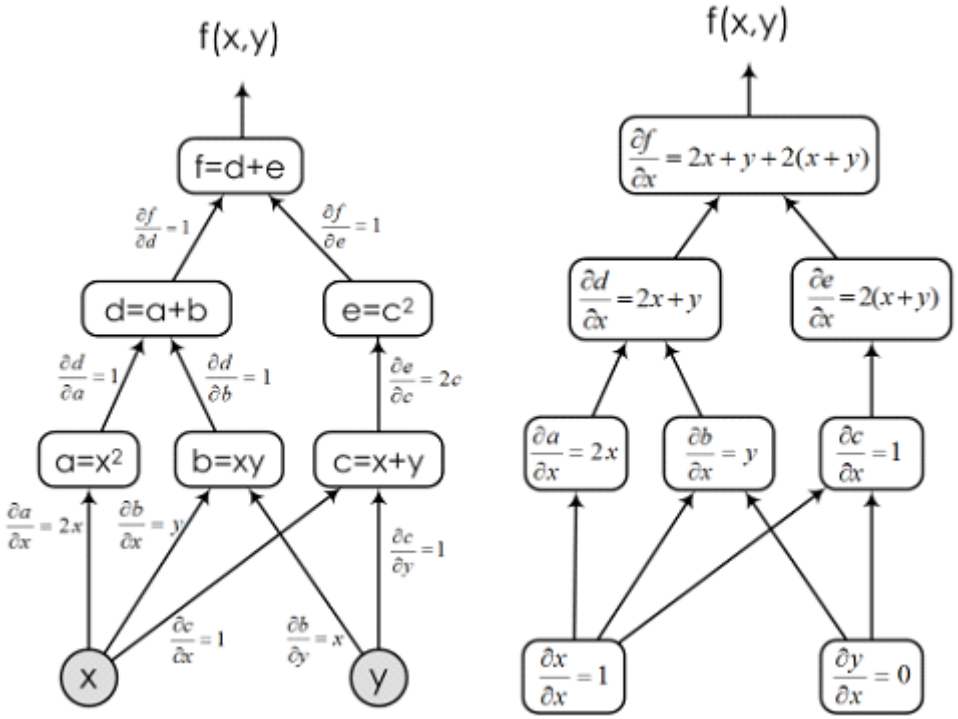

Рисунок 1.15

Основной проблемой данного метода является вычислительная сложность, ведь, чтобы рассчитать градиент в одной точке нам необходимо пройтись по графу n раз, где n – количество параметров. В реальных нейронных сетях количество параметров исчисляется тысячами и миллионами, так что такой способ неприемлем даже с текущими мощностями.

Правильный метод, называемый, методом обратного распространения ошибки, показан на рисунке 1.16 и заключается в обратном вычисление производных сложных функций, благодаря чему вычисление градиента производится за один проход по графу.

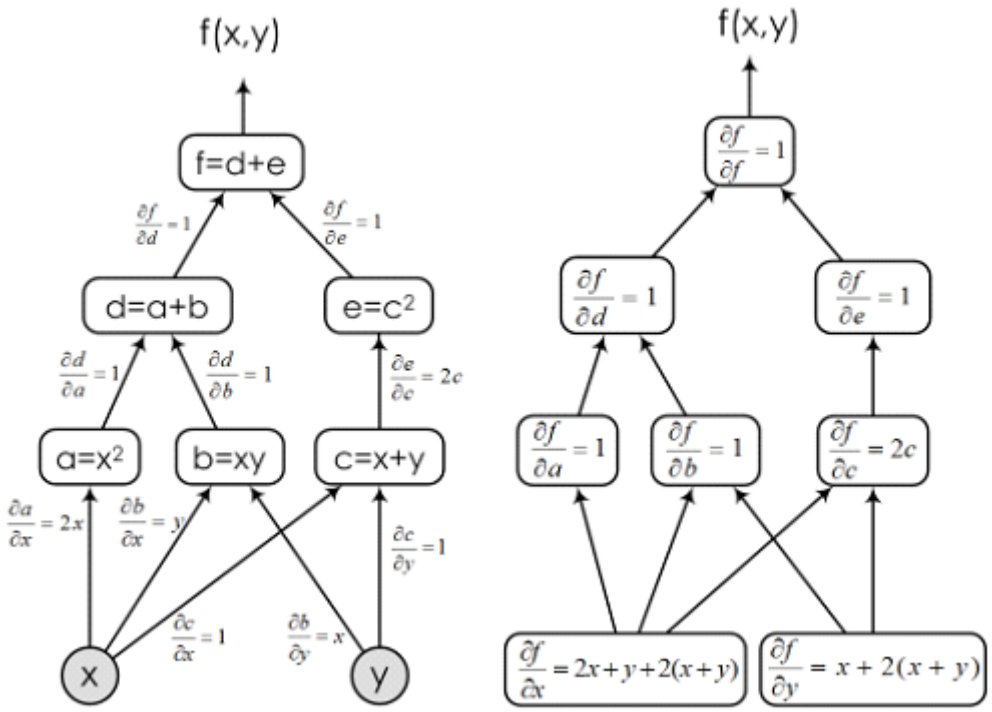

Рисунок 1.16

Как мы видим, при использовании этого метода все производные высчитываются за один проход по графу, что позволяет существенно ускорить процесс обучения нейронных сетей.

Обсудим плюсы и минусы нейронных сетей.

Плюсы:

- Высокая результативность, доказанная как теоретически, теоремой Цыбенко, так и практически.

Минусы:

- Необходимость в большом количестве данных для обучения.
- Не нулевая вероятность сходимости обучающего процесса к неоптимальным параметрам.

На данный момент времени, нейронные сети являются наиболее эффективным методом машинного обучения, хотя у них есть свои недостатки.

2. Сбор и предобработка информации

2.1. Данные

2.1.1. Получение данных

Для предсказания результатов матчей на профессиональной сцене в Counter-Strike: Global Offensive, нам необходимы данные об этих матчах. Именно на этих данных мы будем обучать наши модели. Данные собираются с сайта hltv.org, который является основным сайтам по сбору статистики на профессиональной сцене в CS:GO. Пример прошедшего матча находится на рисунке 2.1.

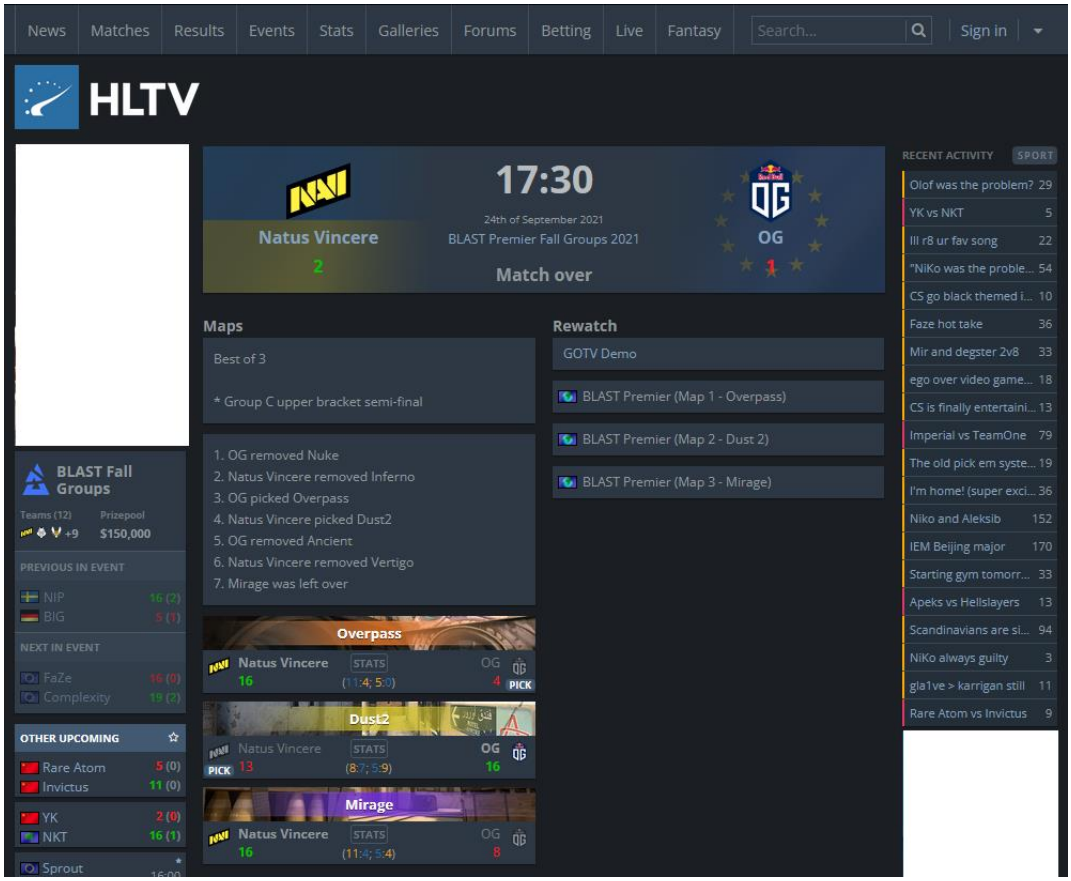

Рисунок 2.1

Информация с сайта собирается каждый день. С утра собирается информация о матчах предстоящего дня, а вечером о результатах этих матчей. Данный способ выбран из-за общего неудобства использования данного сайта. HLTV не предоставляет никакого API, а также не позволяет делать слишком частые запросы. Из-за этих причин, а также из-за общего неудобства в доступе к информации и используются ежедневные запросы к сайту вместо получения исторических данных.

Для сбора данных используются программные библиотеки Python3, такие как requests, для получения интересующей нас html-страницы, и BeautifulSoup4 для сбора необходимых данных (парсинга) с данного документа. В некоторых местах используются регулярные выражения и библиотека re, для работы с ними. Представленные библиотеки предоставляют все необходимые инструменты для решения поставленной задачи, и являются наиболее распространенным способом решения задачи сбора информации с определенного сайта на Python3.

2.1.2. Структура собираемых данных

Для решения задачи предсказания мы используем большое количество специфических данных, для понимания которых необходимо разбираться в соревновательном Couter-Strike. Рассмотрим подробнее данные, получаемые с сайта. Мы собираем несколько типов данных. Первым типом являются общие данные о командах, представленные в таблице 2.1.

Таблица 2.1

Название	Пояснение
Url	Ссылка на страницу матча на hltv.org
Number of confrontation	Номер игры для её последующей индексации
Datetime	Дата и время игры
Best of	Количество карт, на которых происходит игра, обычно это 1,3 или 5 карт
Name 1	Имя левой команды, используется в последствии для составления запросов к сайту
Number 1	Номер левой команды, используется аналогично
Name 2	Имя правой команды, используется в последствии для составления запросов к сайту
Number 2	Номер правой команды, используется аналогично
Team 1 world ranking	Мировой рейтинг левой команды
Team 2 world ranking	Мировой рейтинг правой команды
Data maps 1	Данные по картам для левой команды
Data maps 2	Данные по картам для правой команды

Вторым типом данных являются данные по картам для каждой из команд. Всего в списке карт, играемых на турнирах, используются 7 карт. Для каждой команды мы собираем информацию по каждой карте за последние 3 месяца, причем, разделяем ее для различных уровней оппонентов. Рассматриваются 6

уровней оппонентов: топ 5, топ 10, топ 20, топ 30, топ 50, любые оппоненты. То есть для каждой команды мы делаем 42 запроса, на каждый из которых получаем в ответ данные, приведенные в таблице 2.2.

Таблица 2.2

Название	Пояснение
Times played	Сколько игр на карте сыграно
Wins	Сколько игр выиграно
Draws	Сколько игр сыграно в ничью
Losses	Сколько игр проиграно
Total rounds played	Сколько раундов сыграно
Rounds won	Сколько раундов выиграно
Win percent	Процент выигрыша игр на карте
Pistol rounds	Количество пистолетных раундов
Pistol rounds won	Количество выигранных пистолетных раундов
Pistol rounds won percent	Процент выигрыша пистолетных раундов
CT round win percent	Процент выигрыша раундов за CT сторону
T round win percent	Процент выигрыша раундов за T сторону

Для примера рассмотрим данные об игре команды FaZe на карте Mirage за последние 3 месяца, приведенные на рисунке 2.2. Против любых оппонентов (слева) и сравним с их же игрой за тот же промежуток против оппонентов из топ 5 рейтинга(справа):

Mirage raw stats	
Times played	10
Wins / draws / losses	7 / 0 / 3
Total rounds played	307
Rounds won	157
Win percent	70.0%
Pistol rounds	20
Pistol rounds won	9
Pistol round win percent	45.0%
CT round win percent	51.9%
T round win percent	50.3%

Mirage raw stats	
Times played	5
Wins / draws / losses	3 / 0 / 2
Total rounds played	154
Rounds won	76
Win percent	60.0%
Pistol rounds	10
Pistol rounds won	6
Pistol round win percent	60.0%
CT round win percent	51.2%
T round win percent	47.1%

Рисунок 2.2

Как мы видим, на данном примере, процент выигрыша против сильных команд уменьшился, но при этом, допустим количество выигранных пистолетных раундов увеличилось. Так же из данного сравнения можно сделать эвристический вывод о том, что против оппонентов из топ 5 гораздо сложнее играть за T сторону, которая обычно является слабой стороной.

Все эти данные характеризуют игру команды на каждой карте против соперников определенного уровня. Следуя из всего вышесказанного для одной команды, в одной игре, мы собираем вектор длиной $42 * 12 = 504$. Тогда вспомнив про все наши общие значения для этих команд вектор значений для одной игры равен $10 + 504 * 2 = 1018$. В дальнейшем, когда мы приступим к обработке данных, мы будем использовать, как различные эвристические подходы, основанные на различных предположениях о сути игры, так и более математические, которые будут обрабатывать данные, не опираясь на их область применения.

2.2. Предварительная обработка данных

Предобработка данных крайне важна для наилучшего обучения моделей, из-за того, что даже самая наилучшая модель может оказаться плоха, если для обучения будут взяты данные, с большими шумами, ненормализованные или с большим количеством не значащих переменных. В данной дипломной работе, мы

будем применять различные способы обработки данных, после которых будем сравнивать различные модели, обученные на них.

Предобработку данных мы будем совершать в 2 этапа. Первым этапом является создание выборки данных, путем выбора определенных характеристик объектов. Вторым этапом является улучшение полученных данных, для наилучшего обучения моделей.

2.2.1. Способы создания выборки данных

Выборка из всех данных.

В качестве первого способа создания выборки данных мы рассмотрим способ, при котором все наши данные будут включены в итоговую таблицу. Данные, собранные таким способом, можно разбить на несколько частей:

- Общие данные, не используемые при обучении. В эту категорию входят такие данные, как: время матча, названия команд, ссылка на матч и т.п. Эти данные не используются для обучения, а служат для идентификации матчей.
- Общие данные, используемые при обучении. В этой категории находятся данные, используемые при обучении, например, количество карт, их список, мировой рейтинг команд.
- Данные по картам. В этой категории находятся данные, характеризующие игру команды на определенной карте, против оппонентов, определенного уровня. Это самая обширная категория, т.к. на каждой карте, которых 7, берется информация для 6 уровней оппонентов, т.е. мы получаем 42 группы данных, в каждой из которых 12 значений.
- Целевая переменная. Результат игры.

Перед моделями, обученными на таких данных, не ставиться цели хорошо предсказывать результаты игр. Такие модели будут являться исходной точкой нашего исследования (baseline), от которой можно будет оттолкнуться, для дальнейшего улучшения наших моделей.

Выборка из всех данных, для одной карты.

Как понятно из название данного параграфа, мы будем использовать те же данные, что и в прошлом параграфе, но мы будем пытаться решить часть от общей задачи и предсказать не результат всего матча, а только результат игры на

одной карте. От предыдущего способа отличается только целевой переменной и отсутствием всего списка карт, на которых происходит игра.

Выборка части данных, для одной карты.

Как и в предыдущем случае, здесь, в качестве целевой переменной, используется результат команды на определенной карте, но при этом сильно уменьшены объемы информации. Обсудим имеющиеся данные:

- Общие данные, не используемые при обучении. В данной категории все без изменений.
- Общие данные, используемые при обучении. Тут есть изменения, т.к. мы больше не используем информацию, о количестве карт и их названиях, но берем информацию о текущей карте.
- Данные по карте. Мы существенно уменьшаем количество информации. Мы берем данные только по текущей карте, причем не против всех 6 уровней оппонентов, а только против уровня оппонента текущего матча и для оппонентов из любого уровня.
- Целевая переменная. Целевой переменной все также является итог матча на данной карте.

Данный способ получения данных является самым эвристическим из всех, т.к. предположения, на которых он основан, подкреплены только общими наблюдениями.

К каждой из полученных выборок мы будем применять различные методы улучшения данных, для построения наилучших моделей. Обсудим способы улучшения данных.

2.2.2. Способы улучшения данных

Стандартизация данных.

Стандартизация выборки – процедура, преобразующая все переменные к определенному виду. Обычно, такая трансформация данных, либо приводит все данные в определенный диапазон, либо задает одинаковые средние и дисперсии. Это необходимо, т.к. многие алгоритмы машинного обучения плохо обучаются, при отсутствие этих свойств. Рассмотрим самые популярные способы стандартизации данных.

Standard Scaler – один из самых популярных алгоритмов стандартизации данных, задающий всем данным одинаковое среднее и дисперсию.

$$z = \frac{x - M(X)}{std(X)}$$

где M(X) – выборочное среднее по выборке X, а std(X) – выборочное стандартное отклонение. Выборка, после такого преобразования имеет $M(Z) = 0$ и $D(Z) = 1$, при этом форма распределения не изменяется.

Другим популярным способом стандартизации данных является преобразование MinMax Scaler.

$$z = \frac{x - \min(X)}{\max(X) - \min(X)}$$

Данные, к которым было применено такое преобразование, оказываются заключенными в интервал от 0 до 1.

Благодаря таким преобразованиям наши данные оказываются в одном диапазоне, что положительно влияет на способность алгоритмов машинного обучения к, собственно, обучению.

Приведение данных к нормальному виду.

Также иногда, для еще лучшей обучаемости используют преобразование данных к нормальному виду при помощи степенных преобразований. Степенные преобразования – это семейство параметрических монотонных преобразований, целью которых является преобразование данных из любого распределения в максимально близкое к распределению Гаусса, чтобы стабилизировать дисперсию и минимизировать асимметрию.

Самым часто используемым степенным преобразованием является преобразование Yeo-Johnson:

$$x_i^{(\lambda)} = \begin{cases} [(x_i + 1)^\lambda - 1]/\lambda & \text{if } \lambda \neq 0, x_i \geq 0, \\ \ln(x_i + 1) & \text{if } \lambda = 0, x_i \geq 0 \\ -[(-x_i + 1)^{2-\lambda} - 1]/(2 - \lambda) & \text{if } \lambda \neq 2, x_i < 0, \\ -\ln(-x_i + 1) & \text{if } \lambda = 2, x_i < 0 \end{cases}$$

Вторым же по частоте использования является трансформация Box-Cox:

$$x_i^{(\lambda)} = \begin{cases} \dfrac{x_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(x_i) & \text{if } \lambda = 0, \end{cases}$$

Параметр λ определяется методом максимального правдоподобия.

Основным различием этих преобразований является область допустимых значений: для Box-Cox преобразования допустимы только положительные значения, в то время как для Yeo-Johnson любые. Пример работы данных преобразований над экспоненциально распределенной случайной величиной можно увидеть на рисунке 2.3.

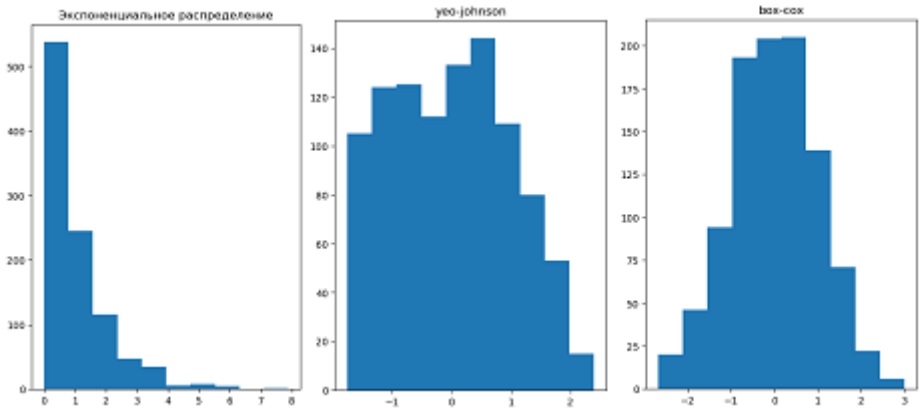

Рисунок 2.3

Уменьшение размерности данных.

В нашей задаче, как и во многих других задачах машинного обучения, наблюдается большое число признаков. Одним из способов улучшения данных является уменьшение размерности данных.

Объекты описаны признаками $F = (f_1, f_2, \dots, f_n)$

Задача:

Построить множество признаков $G = (g_1, g_2, \dots, g_k)$: $k < n$ (иногда $k \ll n$), таким образом, чтобы сохранить наибольшее количество информации.

Основными плюсами такого преобразования данных являются:

- Ускорение обработки данных, обучения и применения алгоритмов машинного обучения.
- Уменьшения шума в данных, за счет того, что оставляем только самые значимые признаки.
- Борьба с мультиколлинеарностью, за счет уменьшения размерности.

Также одной из основных проблем, с которыми позволяет справиться уменьшение размерности, является «проклятие размерности». «Проклятие размерности» – это совокупность затруднений, связанных с большой размерностью признакового описания данных. Эта совокупность включает в себя такие составляющие:

- Высокие требования к памяти и вычислительной мощности устройства
- Разрежённость данных
- Легкость в подтверждении гипотез, не соответствующих реальности.

Есть различные способы уменьшения размерности данных. Мы обсудим только самый популярный из них.

Метод главных компонент.

Метод главных компонент (PCA) – наиболее известный метод для уменьшения размерности данных. Рассмотрим одномерный случай. Предположим, что нам необходимо описать множество точек из $\mathbb{R}^2$ одной переменной, таким образом, чтобы сохранить максимальное количество информации о точках. Для этого будем проецировать точки на прямую, такую что, дисперсия проекций на прямой будет максимальна. Данный случай рассмотрен на рисунке 2.4.

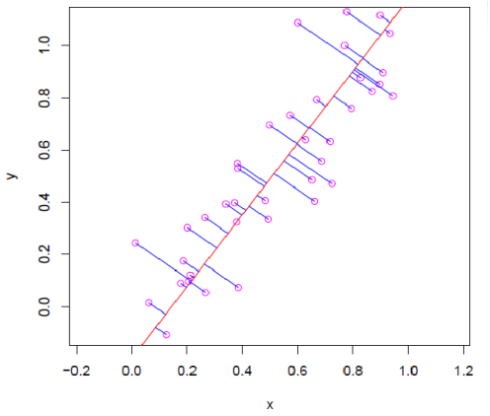

Рисунок 2.4

Рассмотрим требования к задаче в общем случае. Необходимо приблизить данные линейным многообразием меньшего размера таким образом, чтобы:

- Максимизировать дисперсию проекций.
- Корреляция между осями проекций равна нулю.

Математическая постановка задачи:

$f_1(x), \dots, f_n(x)$ – исходные числовые признаки.

$g_1(x), \dots, g_m(x)$ – новые числовые признаки, $m \leq n$.

Исходные числовые признаки $f_j(x)$ должны линейно восстанавливаться по новым числовым признакам $g_s(x)$:

$$\hat{f}_j(x) = \sum_{s=1}^{m} g_s(x) * u_{js}, j = 1, \dots, n,$$

как можно точнее на обучающей выборке $x_1, \dots, x_l$:

$$\sum_{i=1}^{l}\sum_{j=1}^{n}(\hat{f}_j(x)-f_j(x))^2 \to min \text{ по } \{g_s(x_i)\},\{u_{js}\}$$

Найти: новые признаки $g_s(x)$ и преобразования u_{js}.

Запишем в матричном виде:
Матрицы «объекты-признаки», старая и новая:

$$\underset{\ell\times n}{F}=\begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_\ell) & \dots & f_n(x_\ell) \end{pmatrix}; \quad \underset{\ell\times m}{G}=\begin{pmatrix} g_1(x_1) & \dots & g_m(x_1) \\ \dots & \dots & \dots \\ g_1(x_\ell) & \dots & g_m(x_\ell) \end{pmatrix}.$$

Матрица линейного преобразования новых признаков в старые:

$$\underset{n\times m}{U}=\begin{pmatrix} u_{11} & \dots & u_{1m} \\ \dots & \dots & \dots \\ u_{n1} & \dots & u_{nm} \end{pmatrix};$$

Запись задачи в матричном виде:

$$\|GU^T-F\|^2 \to min \text{ по } G,U$$

Для решения этой задачи используется «Основная теорема метода главных компонент»:

Если k $\leq$ rank F, то минимум достигается, когда столбцы U являются собственными векторами матрицы F^TF, соответствующим m максимальным собственным значениям, а матрица $G=FU$.
Следствия:

- Матрица U ортонормирована: $U^TU=E_m$;
- Матрица G ортогональна: $G^TG=\Lambda=diag(\lambda_1,\dots,\lambda_m)$;
- $U\Lambda=F^TFU$; $G\Lambda=F^TFG$
- $\|GU^T-F\|^2=\ \|F\|^2-tr\Lambda=\sum_{j=m+1}^{n}\lambda_j$;

После исследования этой теоремы решение задачи становится тривиально:
G — линейное многообразие
Оси G — главные компоненты
Итеративный поиск главных компонент:

- найти прямую c_1, расстояние до которой минимально.
- Повторять: найти прямую c_i, ортогональную $\{c_j\}_{j=1}^{i-1}$, расстояние до которой минимально.

Также важным вопросом является выбор количества новых переменных m. Эту проблему решают при помощи E(m), характеризующего долю информации, теряемой при проекции.

$$E_m = \frac{\|GU^T - F\|^2}{\|F\|^2} = \frac{\lambda_{m+1} + \cdots + \lambda_n}{\lambda_1 + \cdots + \lambda_n}$$

Следую всему вышеперечисленному мы можем подобрать такое число m, что бы нас устраивала доля теряемой информации.

Метод главных компонент был разработан известным английским математиком и основателем математической статистики Карлом Пирсоном.

3. Проведение исследований и анализ результатов

В этой главе мы будем обучать и сравнивать результаты для различных моделей машинного обучения, примененных к данным из разных выборок, обработанных различными способами.

Рассмотрим выборки, используемые в нашем исследование. Подробнее эти выборки были разобраны во второй главе в параграфе 2.2.1.

Выборки:

- Выборка из всех данных. В этой выборке используются все, собранные нами данные. Целевой переменной является результат матча.
- Выборка из всех данных, для одной карты. В этой выборке используются все, собранные нами данные. Целевой переменной является результат игрой на одной карте.
- Выборка части данных, для одной карты. В этой выборке используются только часть данных собранных нами. Эти данные характеризуют игру команды на одной карте. Целевой переменной является результат игрой на одной карте.

Способы обработки данных:

- Без обработки. В этом случае вся обработка данных сводится к исключению или преобразованию в численный тип данных строкового типа.
- Стандартизация. В этом случае данные преобразуются таким образом, что среднее у всех данных становится равным 0, а дисперсия 1.
- Приведения к нормальному типу. В этом случае данные преобразуются к нормальному виду при помощи метода Yeo-Johnson.
- PCA. В этом случае данные преобразуются алгоритмом PCA, при этом будем всегда пытаться сохранить максимальную дисперсию данных, при значительном уменьшение их размерности.

Модели машинного обучения:

- Dummy. Не совсем модель машинного обучения. Просто запоминает наиболее частый класс на обучающей выборке и предсказывает результат на тестовой выборке.
- Perceptron. Линейный классификатор, построенный на ошибке перцептрона.
- SVM. Метод опорных векторов.
- Логистическая регрессия.
- Решающее дерево.

- Случайный лес.
- Градиентный бустинг.
- Нейронная сеть.

Таким образом мы будем сравнивать 8 моделей на каждой из 12 выборок данных, полученных и обработанных различными способами.

Для оценки методов мы будем использовать среднее значение точности на кросс-валидации.

3.1. Способы сравнения результатов

3.1.1. Метрика

Для сравнения методов машинного обучения необходимо понять, по каким критериям их сравнивать. Из-за того, что в задаче присутствует человеческий фактор, а некоторые данные либо трудно получить, либо невозможно оценить, например, моральное состояние игроков перед игрой, мы можем утверждать, что невозможно идеально предсказывать предстоящие матчи. Но, мы можем создать алгоритм, который будет делать это лучше остальных. Каким же образом мы будем оценивать наши алгоритмы?

Очевидным способом сравнения для нашей задачи является метрика точности (Accuracy) – доля правильно предсказанных объектов:

$$Accuracy(y_{true}, y_{pred}) = \frac{1}{N}\sum_{i=1}^{N} |y_{true}^{i} - y_{pred}^{i}|$$

Так как, в нашей задаче нет верного и неверного класса (для нас нет менее важной команды: правой или левой), эта метрика неплохо нам подходит, для решения задачи классификации.

3.1.2. Валидация данных

Одной из основных проблем машинного обучения является проблема переобучения. Модель называется переобученной, когда вместо выявления необходимой закономерности в обучающей выборке, она просто ее запоминает, из-за чего на тестовой выборке показывает плохой результат. Для недопуска переобучения обычно используется один из способов. Первым способом является разбиение выборки на обучающую, валидационную и тестовую выборки. На обучающей выборке модели обучаются, сравниваются модели на валидационной

выборке, а тестовая используется, как последний шаг, на котором можно удостоверится, что модель не подогнана по какие-либо данные.

Вторым способом является использование кросс-валидации. При этом способе из данных снова выделяется тестовая выборка, которую модель увидит только в самый последний момент, и обучающая выборка. Обучающая выборка разбивается на *n* не пересекающихся частей. За один раз обучаются *n* моделей, на *n-1* различных частях. Среднее значение предсказаний моделей на неиспользованных частях обучающей выборки используется для валидации. Пример разбиения данных на кросс-валидации можно увидеть на рисунке 3.1.

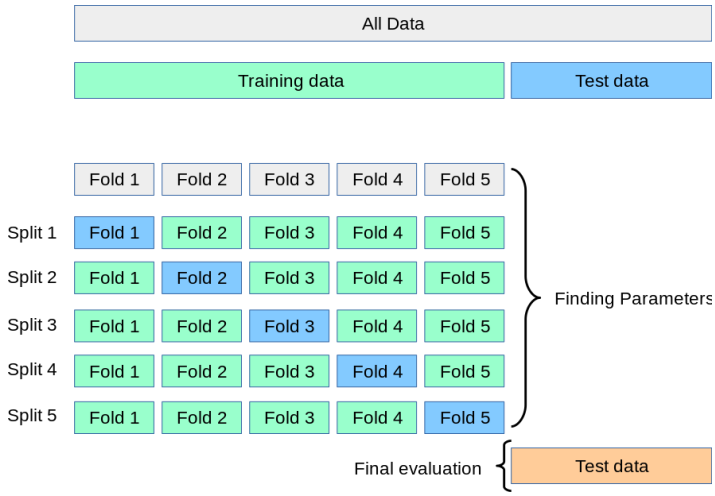

Рисунок 3.1

3.2. Проведение исследований

3.2.1. Выборка из всех данных

Без обработки.

Рассмотрим эффективность различных моделей на выборке из всех данных без обработки, где целевой переменной является результат матча. Как выглядят данные после базовой обработки, мы можем увидеть в таблице 3.1.

Таблица 3.1

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	28	46	0	0	0	0	0	0
1	1	223	2	2	0	0	54	32
2	14	101	0	0	0	0	0	0
3	3	46	0	0	0	0	0	0
4	7	1	0	0	0	0	0	0

Суммарное число колонок (характеристик матча): 1017.

Суммарное число строк (матчей): 842.

Результаты, различных моделей находятся в таблице 3.2:

Таблица 3.2

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	56.6%	-
Perceptron	56.5%	58.5%
SVM	60.6%	60.6%
Логистическая регрессия	59.6%	61.4%
Решающее дерево	56.6%	60.6%
Случайный лес	61.1%	63%
Градиентный бустинг	60.8%	62.4%
Нейронная сеть	-	63.9%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 3 скрытых слоя, в каждом из которых 5 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

Стандартизация.

Рассмотрим эффективность различных моделей на выборке из всех данных со стандартизацией, где целевой переменной является результат матча. Пример части данных после базовой обработки и стандартизации, мы можем увидеть в таблице 3.3.

Таблица 3.3

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-0,64	-0,56	-0,18	-0,13	0	-0,14	-0,18	-0,17
1	-1,08	2,11	6,21	7,76	0	-0,14	5,89	6,37
2	-0,87	0,26	-0,18	-0,13	0	-0,14	-0,18	-0,17
3	-1,04	-0,56	-0,18	-0,13	0	-0,14	-0,18	-0,17
4	-0,98	-1,25	-0,18	-0,13	0	-0,14	-0,18	-0,17

Суммарное число колонок(характеристик матча): 1017.

Суммарное число строк (матчей): 842.

Результаты, различных моделей находятся в таблице 3.4:

Таблица 3.4

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	56.6%	-
Perceptron	57.9%	59.7%
SVM	60.8%	61.5%
Логистическая регрессия	57.8%	62.2%
Решающее дерево	53.8%	57.9%
Случайный лес	61.1%	62.9%
Градиентный бустинг	60.8%	62.3%

Нейронная сеть	-	62.9%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 4 скрытых слоя, в каждом из которых 4 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1. Также хорошо себя показывает Случайный лес.

Нормальный вид.

Рассмотрим эффективность различных моделей на выборке из всех данных с приведением этих данных к нормальному виду, где целевой переменной является результат матча. Пример части данных после базовой обработки и приведения к нормальному виду, мы можем увидеть в таблице 3.5.

Таблица 3.5

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-0,45	-0,35	-0,20	-0,14	0	-0,14	-0,20	-0,20
1	-2,14	1,64	4,95	6,76	0	-0,14	4,95	4,95
2	-0,97	0,50	-0,20	-0,14	0	-0,14	-0,20	-0,20
3	-1,80	-0,35	-0,20	-0,14	0	-0,14	-0,20	-0,20
4	-1,39	-2,26	-0,20	-0,14	0	-0,14	-0,20	-0,20

Суммарное число колонок(характеристик матча): 1017.
Суммарное число строк (матчей): 842.
Результаты, различных моделей находятся в таблице 3.6:

Таблица 3.6

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	56.6%	-
Perceptron	60.3%	60.3%
SVM	61.2%	61.8%
Логистическая регрессия	58.6%	61.8%
Решающее дерево	55.4%	58.3%
Случайный лес	62.3%	63.4%
Градиентный бустинг	60.4%	61.9%
Нейронная сеть	-	64.4%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 4 скрытых слоя, в каждом из которых 5 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

Метод главных компонент.

Рассмотрим эффективность различных моделей на выборке из всех данных после предобработки методом главных компонент, где целевой переменной является результат матча. Пример части данных после базовой обработки применения метода главных компонент, мы можем увидеть в таблице 3.7.

Таблица 3.7

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-440,12	-134,30	35,15	72,35	20,23	-93,18	-141,53	66,09
1	262,58	-18,82	575,08	-341,58	-31,82	-300,33	-51,87	228,38
2	-56,46	442,48	416,71	-103,74	-157,42	-82,76	-170,41	82,92
3	158,49	236,34	391,15	-293,57	-208,45	-204,54	-342,43	143,84
4	365,53	-163,56	578,73	-415,11	156,74	-194,84	126,34	195,72

Суммарное число колонок(характеристик матча): 50.

Суммарное число строк (матчей): 842.

Результаты, различных моделей находятся в таблице 3.8:

Таблица 3.8

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	56.6%	-
Perceptron	56.8%	59.2%
SVM	60.5%	61.7%
Логистическая регрессия	61.7%	62.3%
Решающее дерево	52.6%	59.8%
Случайный лес	60.9%	62.9%
Градиентный бустинг	57.7%	59.9%
Нейронная сеть	-	63.1%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 4 скрытых слоя, в каждом из которых 4 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.2.

3.2.2. Выборка из всех данных для одной карты

Без обработки.

Рассмотрим эффективность различных моделей на выборке из всех данных без обработки, где целевой переменной является результат карты. Как выглядят данные после базовой обработки, мы можем увидеть в таблице 3.9.

Таблица 3.9

	0	1	2	3	4	5	6	7
0	28	46	0	0	0	0	0	0
1	28	46	0	0	0	0	0	0
2	1	223	2	2	0	0	54	32
3	1	223	2	2	0	0	54	32
4	14	101	0	0	0	0	0	0

Суммарное число колонок(характеристик матча): 1012.
Суммарное число строк (карт): 1813.
Результаты, различных моделей находятся в таблице 3.10:

Таблица 3.10

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	55.1%	56.7%
SVM	59%	59%
Логистическая регрессия	56.7%	57.7%
Решающее дерево	58.4%	60%
Случайный лес	60.6%	63%
Градиентный бустинг	59.6%	60.8%
Нейронная сеть	-	61.9%

Вывод: Наилучшей моделью является случайный лес, состоящий из 1000 деревьев, с максимальной высотой дерева 10 и энтропией в качестве критерия ветвления.

Стандартизация.

Рассмотрим эффективность различных моделей на выборке из всех данных со стандартизацией, где целевой переменной является результат карты. Пример части данных после базовой обработки и стандартизации, мы можем увидеть в таблице 3.11.

Таблица 3.11

	0	1	2	3	4	5	6	7
0	-0,64	-0,54	-0,19	-0,14	0	-0,14	-0,19	-0,18
1	-0,64	-0,54	-0,19	-0,14	0	-0,14	-0,19	-0,18
2	-1,09	2,22	6,05	7,46	0	-0,14	5,74	6,17
3	-1,09	2,22	6,05	7,46	0	-0,14	5,74	6,17
4	-0,88	0,31	-0,19	-0,14	0	-0,14	-0,19	-0,18

Суммарное число колонок(характеристик матча): 1012.
Суммарное число строк (карт): 1813.

Результаты, различных моделей находятся в таблице 3.12:

Таблица 3.12

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	55.9%	56%
SVM	59.1%	61%
Логистическая регрессия	58.6%	60.7%
Решающее дерево	57.9%	60.3%
Случайный лес	60.7%	63.2%
Градиентный бустинг	59.7%	60.8%
Нейронная сеть	-	60%

Вывод: Наилучшей моделью является случайный лес, состоящий из 100 деревьев, с максимальной высотой дерева 11 и энтропией в качестве критерия ветвления.

Нормальный вид.

Рассмотрим эффективность различных моделей на выборке из всех данных с приведением этих данных к нормальному виду, где целевой переменной является результат карты. Пример части данных после базовой обработки и приведения к нормальному виду, мы можем увидеть в таблице 3.13.

Таблица 3.13

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-0,46	-0,33	-0,21	-0,16	0,00	-0,15	-0,21	-0,21
1	-0,46	-0,33	-0,21	-0,16	0,00	-0,15	-0,21	-0,21
2	-2,18	1,70	4,81	6,42	0,00	-0,15	4,81	4,81
3	-2,18	1,70	4,81	6,42	0,00	-0,15	4,81	4,81
4	-0,99	0,55	-0,21	-0,16	0,00	-0,15	-0,21	-0,21

Суммарное число колонок(характеристик матча): 1012.

Суммарное число строк (карт): 1813.

Результаты, различных моделей находятся в таблице 3.14:

Таблица 3.14

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	54.6%	55.4%
SVM	59.7%	60.6%
Логистическая регрессия	58.4%	60.3%
Решающее дерево	58.2%	59.8%

Случайный лес	60.6%	63.3%
Градиентный бустинг	60.2%	60.6%
Нейронная сеть	-	59%

Вывод: Наилучшей моделью является случайный лес, состоящий из 200 деревьев, с максимальной высотой дерева 11 и энтропией в качестве критерия ветвления.

Метод главных компонент.

Рассмотрим эффективность различных моделей на выборке из всех данных после предобработки методом главных компонент, где целевой переменной является результат карты. Пример части данных после базовой обработки применения метода главных компонент, мы можем увидеть в таблице 3.15.

Таблица 3.15

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	-456,29	-138,35	64,62	60,49	34,73	28,33	-101,62	-147,25
1	-456,29	-138,35	64,62	60,49	34,73	28,33	-101,62	-147,25
2	251,07	36,43	588,57	-358,07	-34,44	-151,03	-68,19	-317,30
3	251,08	36,43	588,57	-358,07	-34,44	-151,03	-68,19	-317,30
4	-64,44	472,30	389,59	-91,67	-142,22	61,84	-134,11	-148,61

Суммарное число колонок(характеристик матча): 50.

Суммарное число строк (карт): 1813.

Результаты, различных моделей находятся в таблице 3.16:

Таблица 3.16

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	54.2%	55.3%
SVM	59%	59.2%
Логистическая регрессия	58.8%	58.8%
Решающее дерево	59.4%	60.6%
Случайный лес	60.5%	61.8%
Градиентный бустинг	60.5%	61.4%
Нейронная сеть	-	62.4%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 2 скрытых слоя, в каждом из которых 100 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

3.2.3. Выборка части данных, для одной карты

Без обработки.

Рассмотрим эффективность различных моделей на выборке из части данных без обработки, где целевой переменной является результат карты. Как выглядят данные после базовой обработки, мы можем увидеть в таблице 3.17.

Таблица 3.17

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	4	28	46	3	0	0	3	80
1	0	28	46	1	0	0	1	27
2	3	1	223	7	4	0	3	182
3	0	1	223	2	2	0	0	54
4	5	14	101	4	2	0	2	106

Суммарное число колонок(характеристик матча): 52.

Суммарное число строк (карт):1813.

Результаты, различных моделей находятся в таблице 3.18:

Таблица 3.18

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	51.8%	55%
SVM	58.4%	58.5%
Логистическая регрессия	60.5%	60.5%
Решающее дерево	53.2%	59.1%
Случайный лес	56%	58.5%
Градиентный бустинг	56%	58.5%
Нейронная сеть	-	61.5%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 3 скрытых слоя, в каждом из которых 8 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

Стандартизация.

Рассмотрим эффективность различных моделей на выборке из части данных со стандартизацией, где целевой переменной является результат карты. Пример части данных после базовой обработки и стандартизации, мы можем увидеть в таблице 3.19.

Таблица 3.19

	0	1	2	3	4	5	6	7
0	0,51	-0,64	-0,54	-0,54	-0,96	0	0,17	-0,54
1	-1,58	-0,64	-0,54	-0,94	-0,96	0	-0,64	-0,93
2	-0,01	-1,09	2,21	0,25	0,26	0	0,17	0,21
3	-1,58	-1,09	2,21	-0,74	-0,34	0	-1,06	-0,73
4	1,04	-0,87	0,31	-0,34	-0,34	0	-0,23	-0,35

Суммарное число колонок(характеристик матча): 52.

Суммарное число строк (карт):1813.

Результаты, различных моделей находятся в таблице 3.20:

Таблица 3.20

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	53.7%	56%
SVM	58.5%	58.9%
Логистическая регрессия	58.7%	59.2%
Решающее дерево	53.5%	59.1%
Случайный лес	56%	59.7%
Градиентный бустинг	56.5%	58.1%
Нейронная сеть	-	60.8%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 2 скрытых слоя, в каждом из которых 10 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

Нормальный вид.

Рассмотрим эффективность различных моделей на выборке из части данных с приведением этих данных к нормальному виду, где целевой переменной является результат карты. Пример части данных после базовой обработки и приведения к нормальному виду, мы можем увидеть в таблице 3.21.

Таблица 3.21

	0	1	2	3	4	5	6	7
0	0,54	-0,46	-0,32	-0,32	-1,37	0	0,46	-0,22
1	-1,69	-0,46	-0,32	-1,01	-1,37	0	-0,51	-0,85
2	0,048	-2,18	1,70	0,50	0,61	0	0,46	0,47
3	-1,69	-2,18	1,70	-0,63	-0,02	0	-1,41	-0,48
4	1,02	-0,99	0,55	-0,07	-0,01	0	0,05	-0,01

Суммарное число колонок(характеристик матча): 52.

Суммарное число строк (матчей): 1813.

Результаты, различных моделей находятся в таблице 3.22:

Таблица 3.22

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	52.8%	57.6%
SVM	58.6%	58.6%
Логистическая регрессия	59.3%	59.7%
Решающее дерево	53.4%	59%
Случайный лес	56.9%	58.5%
Градиентный бустинг	56.6%	58.3%
Нейронная сеть	-	60.5%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 2 скрытых слоя, в каждом из которых 5 нейронов с функцией активации relu. Так же используется Dropout с параметром 0.1.

Метод главных компонент.

Рассмотрим эффективность различных моделей на выборке из всех данных после предобработки методом главных компонент, где целевой переменной является результат карты. Пример части данных после базовой обработки применения метода главных компонент, мы можем увидеть в таблице 3.23.

Таблица 3.23

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	-154,04	-65,92	117,94	8,39	-37,88	-1,07	12,12
1	-231,52	-49,61	130,89	12,53	-39,88	3,81	3,42
2	12,90	-18,77	-76,80	-49,18	-8,16	143,03	-13,78
3	-234,50	-202,42	-0,23	-4,07	-6,19	138,50	-4,06
4	-146,41	-202,56	18,49	49,59	-50,16	42,87	-5,03

Суммарное число колонок(характеристик матча): 6.

Суммарное число строк (матчей): 1813.

Результаты, различных моделей находятся в таблице 3.24:

Таблица 3.24

Название модели	Точность со стандартными гиперпараметрами	Точность с подобранными гиперпараметрами
Dummy	54.9%	-
Perceptron	55.8%	56.3%
SVM	58.4%	58.6%
Логистическая регрессия	60%	60%
Решающее дерево	53.4%	56.5%
Случайный лес	57.2%	58.7%

Градиентный бустинг	57.8%	57.8%
Нейронная сеть	-	60.5%

Вывод: Наилучшей моделью является обученная нейросеть, содержащая 2 скрытых слоя, в каждом из которых 8 нейронов с функцией активации relu.

3.3. Анализ результатов исследований

После проведения всех исследований можно сделать определенные выводы о эффективности различных моделей машинного обучения, примененных к различным выборкам данных.

Анализ точности моделей на выборке из всех данных.

На выборке из всех данных наилучшим образом себя показывают многослойные нейронные сети и случайный лес, результаты которого немногим хуже. Линейные модели показывают себя не столь хорошо, но их результаты все равно лучше случайного угадывания. Градиентный бустинг проигрывает случайному лесу, но выигрывает у линейных моделей. Различные методы обработки данных несущественно влияют на результаты.

Анализ точности моделей на выборке из всех данных для одной карты.

На выборке из всех данных для одной карты наилучшим образом себя показывает случайный лес. Линейные модели, нейронные сети и градиентный бустинг показывают приблизительно равную точность, которая ниже, точности случайного леса, но выше случайного угадывания. Все методы обработки данных, кроме метода главных компонент незначительно влияют на точность моделей машинного обучения. После применения к данным метода главных компонент значительно ухудшилась точность случайного леса, в то время как точность многослойной нейросети возросла.

Анализ точности моделей на выборке из части данных для одной карты.

На выборке из части данных для одной карты наилучшим образом себя показывает многослойная нейросеть, но её точность несущественно отличается от других моделей. Различные методы обработки данных несущественно влияют на результаты.

Наивысшую точность в 64.4% продемонстрировала многослойная нейросеть, обученная на выборке из всех данных, прошедшей приведение к нормальному виду. Этой точности недостаточно для решения реальных задач, но этого достаточно, что бы утверждать, что данный результат не является результатом случайного угадывания.

Приложение

Листинг программы

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, RobustScaler, StandardScaler
from sklearn.preprocessing import PowerTransformer

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['figure.dpi'] = 100

from sklearn.dummy import DummyClassifier
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('../data/df_games.csv')
df = df.drop(df[(df.best_of == 5) & (df['5_map'] == 'did not play')].index.values, axis=0)
df.info()

df = df.drop(['version', 'url', 'number_of_confrontation', 'datetime1', 'ratio1', 'ratio2', 'name1', 'number1',
'name2', 'number2',
       'team1_stat_url', 'team2_stat_url'], axis=1)
df.head()

maps_decoder = LabelEncoder()
maps_decoder.fit(df['3_map'])
for num_map in range(1,6):
   df[f'{num_map}_map'] = maps_decoder.transform(df[f'{num_map}_map'])

cv = KFold(shuffle=True, random_state=1)
```

```
X = df.drop('result', axis=1)
y = df['result']

X_norm = pd.DataFrame()
transform = PowerTransformer()
for i in X.columns:
    X_norm[i] = transform.fit_transform(np.array(X[i]).reshape(-1, 1)).reshape(1, -1)[0]
X_norm.head()

model_dummy = DummyClassifier(strategy="most_frequent")
result_dummy = cross_val_score(model_dummy, X_norm, y, cv=cv)
print(f'name: Dummy, score: {result_dummy}, mean: {result_dummy.mean()}')

model_perceptron = Perceptron()
result_perceptron = cross_val_score(model_perceptron, X_norm, y, cv=cv)
print(f'name: Perceptron, score: {result_perceptron}, mean: {result_perceptron.mean()}')

param = {'penalty': ['l2', 'l1','none'],
        'alpha': [10**-6, 10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 10**0, 10**1, 10**2]}

%%time
grid_search = GridSearchCV(Perceptron(), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

pd.DataFrame(grid_search.cv_results_).sort_values('mean_test_score',
ascending=False).iloc[:,4:].head()

%%time
model_svm = SVC()
result_svm = cross_val_score(model_svm, X_norm, y, cv=cv, n_jobs=-1)
print(f'name: SVM, score: {result_svm}, mean: {result_svm.mean()}')

param = {'C': [10**-3, 10**-2, 10**-1, 10**0, 10**1],
        'kernel': ['rbf', 'poly', 'sigmoid']}
%%time
grid_search = GridSearchCV(SVC(), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

param = {'C': [1],
        'kernel': ['poly'],
        'degree': range(1,11)}
```

```
%%time
grid_search = GridSearchCV(SVC(), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

%%time
model_tree = DecisionTreeClassifier()
result_tree = cross_val_score(model_tree, X_norm, y, cv=cv, n_jobs=-1)
print(f'name: Tree, score: {result_tree}, mean: {result_tree.mean()}')

param = {'criterion': ['gini', 'entropy'],
        'max_depth': range(2,50)}

%%time
grid_search = GridSearchCV(DecisionTreeClassifier(), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

%%time
model_log = LogisticRegression()
result_log = cross_val_score(model_log, X_norm, y, cv=cv, n_jobs=-1)
print(f'name: LogisticRegression, score: {result_log}, mean: {result_log.mean()}')

param = {'C': [10**-3, 10**-2, 10**-1, 10**0, 10**1]}

%%time
grid_search = GridSearchCV(LogisticRegression(), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

pd.DataFrame(grid_search.cv_results_).sort_values('mean_test_score',
ascending=False).iloc[:,4:].head(10)

%%time
model_rf = RandomForestClassifier()
result_fr = cross_val_score(model_rf, X_norm, y, cv=cv, n_jobs=-1)
print(f'name: RandomForestClassifier, score: {result_fr}, mean: {result_fr.mean()}')

param = {'n_estimators': [100,200,500,1000,2000],
        'criterion': ['gini', 'entropy'],
        'max_depth': range(1,51,10)}

%%time
grid_search = GridSearchCV(RandomForestClassifier(n_jobs=-1), param,cv=cv, n_jobs=-1)
```

```
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

param = {'n_estimators': [100,200,500,1000],
        'criterion': ['gini'],
        'max_depth': range(10,41,5),
        'min_samples_split': range(2,20,2),
        'min_samples_leaf': range(1,10,2)}

%%time
grid_search = GridSearchCV(RandomForestClassifier(n_jobs=-1), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)

%%time
model_xgb = XGBClassifier()
result_xgb = cross_val_score(model_xgb, X_norm, y, cv=cv, n_jobs=-1)
print(f'name: XGBClassifier, score: {result_xgb}, mean: {result_xgb.mean()}')

param = {'n_estimators': [100,200],
        'max_depth': range(10,51,2)}

%%time
grid_search = GridSearchCV(XGBClassifier(n_jobs=-1), param,cv=cv, n_jobs=-1)
grid_search.fit(X_norm,y)
(grid_search.best_params_, grid_search.best_score_)
```

Заключение

В работе рассматривалась задача прогнозирования результатов киберспортивных матчей на профессиональной сцене в онлайн игре Counter-Strike: Global Offensive.В ходе работы были изучены виды классических бинарных классификаторов, а также классические многослойные нейронные сети, применяемые для бинарной классификации.

Были рассмотрены преимущества и недостатки используемых моделей, а также проведены исследования, по применению данных моделей к практической задаче прогнозирования результатов киберспортивных матчей на профессиональной сцене в онлайн игре Counter-Strike: Global Offensive.

В ходе проделанной работы был сделан вывод о том, что наиболее подходящими для нашей задачи являются многослойные нейронные сети и случайный лес.

Данные методы демонстрируют результат предсказывания, заметно лучший, чем случайное угадывание.

Библиографический список

1. Онлайн-учебник по машинному обучению от ШАД [Электронный ресурс] – Режим доступа: https://ml-handbook.ru. – (Дата обращения: 23.06.22).
2. Воронцов, К.В. Машинное обучение (курс лекций) [Электронный ресурс] / К.В. Воронцов – Режим доступа: http://www.machinelearning.ru – (Дата обращения: 23.06.22).
3. Николенко, С.И. Курс лекций [Электронный ресурс] / С.И. Николенко – Режим доступа: https://logic.pdmi.ras.ru/~sergey/teaching/mlhse2020.html – (Дата обращения: 23.06.22).
4. Николенко, С.И. Глубокое обучение / С.И. Николенко, А. А. Кадурин – С-П: Питер, 2017. – 480 с.
5. Рашка, С. Python и машинное обучение. Машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow. С. Рашка, В. Мирджалили – М: Вильямс, 2019. – 659 с.
6. Вьюгин, В.В. Математические основы теории машинного обучения и прогнозирования / Вьюгин В.В. – М.: 2013. - 387 с.
7. Бринк Х. Машинное обучение / Бринк Х., Ричардс Д., Феверолф М. – С-П: Питер, 2017. – 336 с.
8. Open Machine Learning Course [Электронный ресурс] – Режим доступа: https://mlcourse.ai – (Дата обращения: 23.06.22).
9. Траск, Э. Грокаем глубокое обучение / Э. Траск – С-П: Питер, 2019. – 352 с.
10. Элбон, К. Машинное обучение с использованием Python. Сборник рецептов / Э. Крис. – М: БХВ-Петербург, 2019 – 384 с.
11. Hastie, T. The Elements of Statistical Learning. Data Mining, Inference, and Prediction / Hastie T., Tibshirani R., Friedman J. Stanford, California: Springer, 2008. – 764 с.

Содержание