ДИСЦИПЛИНА **Интеллектуальные системы и технологии**

(полное наименование дисциплины без сокращений)

ИНСТИТУТ **информационных технологий**

КАФЕДРА **корпоративных информационных систем**

полное наименование кафедры

ВИД УЧЕБНОГО МАТЕРИАЛА **Лекция**

(в соответствии с пп.1-11)

ПРЕПОДАВАТЕЛЬ **Демидова Лилия Анатольевна**

(фамилия, имя, отчество)

СЕМЕСТР **1 семестр (осенний), 2024 – 2025 учебный год**

(семестр обучения, учебный год)

ЛЕКЦИЯ 4
АЛГОРИТМЫ КЛАССИФИКАЦИИ. РЕГРЕССИОННЫЕ МОДЕЛИ

Алгоритм опорных векторов

Понятие оптимальной разделяющей гиперплоскости

Каждый классификатор (алгоритм классификации) реализует отображение $A: Z \to Y$. С «позиции классификатора» классы, как правило, «достаточно просто устроены»: имеют небольшое число компонент связности, часто выпуклы, имеют гладкую границу и т.д. Границу между классами называют *разделяющей поверхностью* или *гиперплоскостью*: эта поверхность разделяет объекты с разной классовой принадлежностью. Если рассмотреть простой случай, когда в задаче с двумя непересекающимися классами объекты заданы в двумерном пространстве характеристик, то *разделяющая поверхность* – это кривая на плоскости, которая отделяет один класс от другого (рисунок 1).

Построить гиперплоскость, корректно разделяющую объекты, можно разными способами (рисунок 1, слева). Под зазором между классом и гиперплоскостью понимают минимальное расстояние между гиперплоскостью и объектами класса, где d^+ и d^- – зазоры между гиперплоскостью и первым и вторым классом соответственно (рисунок 1, справа).

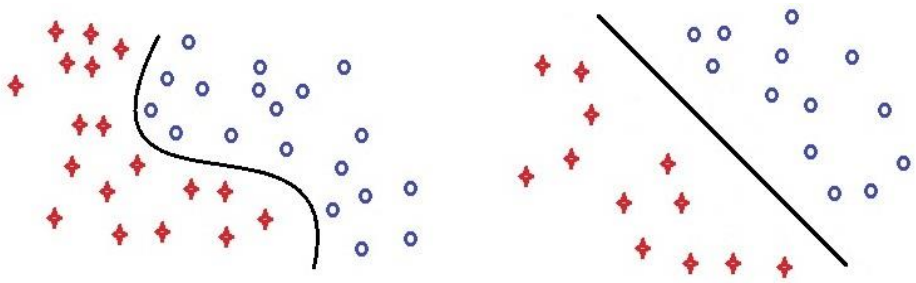

Рисунок 1 – Примеры разделяющих кривых в пространстве D-2

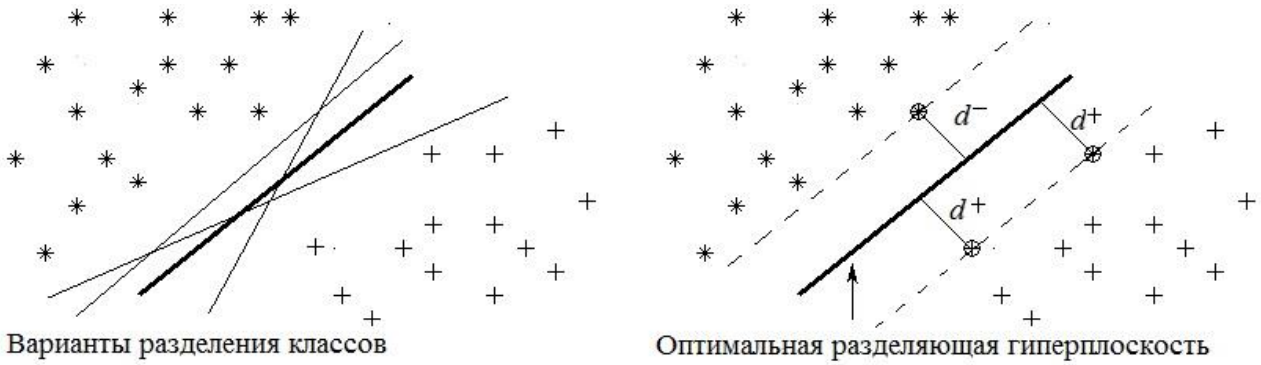

Рисунок 2 – Разделение классов: слева – варианты разделения классов, справа – оптимальная разделяющая гиперплоскость

Тогда *оптимальная гиперплоскость* – гиперплоскость, максимизирующая этот зазор. Максимизация зазора между классами должна способствовать более уверенной классификации.

Алгоритм опорных векторов (Support Vector Mashine, SVM-алгоритм), предложенный В.Н. Вапником, относится к группе граничных алгоритмов классификации. С его помощью решаются задачи бинарной классификации. Алгоритм определяет принадлежность объектов к классам с помощью границ областей. SVM-классификатор представляет собой разновидность классификатора «с широким зазором»: он относится к методам машинного обучения, основанным на модели векторного пространства, цель которых – найти разделяющие поверхности между классами, максимально удаленные от всех объектов обучающей выборки.

Задача бинарной классификации на основе SVM-алгоритма

В задаче бинарной классификации рассматривается множество объектов Z, подлежащих классификации, в котором каждому объекту ставится в соответствие q-мерный вещественный вектор характеристик $z_i = (z_i^1, z_i^2, \ldots, z_i^q)$,

где z_i^l – числовое значение l-й характеристики для i-го объекта ($l = \overline{1,q}$), нормированное значениями [0; 1]; $Y = \{-1,+1\}$ – множество ответов (метки классов); $f^* : Z \to Y$ – целевая зависимость, значения которой известны только на объектах учебного набора $U = \{< z_i, y_i^* >\}_{i=1}^s$, $y_i^* = f^*(z_i)$ – число (–1 или +1), характеризующее классовую принадлежность объекта $z_i \in Z^*$, $Z^* \subset Z$ ($i = \overline{1,s}$). Требуется построить классифицирующую функцию $A: Z \to Y$, аппроксимирующую целевую зависимость на всем пространстве Z. Для этого должна быть построена соответствующая разделяющая гиперплоскость.

Для разработки «наилучшего» классификатора необходимо реализовать многократное обучение и тестирование на различных случайным образом сформированных из учебного набора обучающей и тестовой выборках с последующим выбором «наилучшей» обучающей и тестовой выборок в смысле обеспечения максимально возможного качества классификации. В дальнейшем при удовлетворительном качестве обучения и тестирования на учебном наборе полученный классификатор может быть применен для классификации новых объектов из множества Z.

Пусть из учебного набора $U = \{< z_i, y_i^* >\}_{i=1}^s$ случайным образом выбраны S кортежей ($S<s$) и сформирована обучающая выборка $Train = \{< z_i, y_i^* > \mid z_i \in Z^*,\ y_i^* = f^*(z_i)\}_{i=1}^S$. В результате обучения SVM-классификатора определяется разделяющая гиперперплоскость, которая может быть задана уравнением:

$$w \bullet z + b = 0, \qquad (1)$$

где w − вектор-перпендикуляр к разделяющей гиперплоскости; $b \in R$ − параметр, задающий смещение гиперплоскости относительно начала координат (если значение параметра b равно нулю, то гиперплоскость проходит через начало координат); $w \bullet z$ − скалярное произведение векторов w и z.

Значения параметров классификатора определены с точностью до нормировки: алгоритм $A(z)$ не изменится, если w и b одновременно умножить на одну и ту же положительную константу. Удобно выбрать эту константу таким образом, чтобы для всех ближайших к разделяющей гиперплоскости объектов $z_i \in Z^*$ выполнялись условия: $w \bullet z_i + b = y_i$, т.е. расстояние от разделяющей гиперплоскости до пограничных объектов обоих классов было равно 1.

Пример построения разделяющей гиперплоскости *в идеальном случае* в пространстве D-2 приведен на рисунке 3: объект относится к первому классу, если он лежит с «положительной» стороны от гиперплоскости, и относится ко второму классу в противном случае. Условие $-1 < w \bullet z + b < 1$ задает полосу, которая разделяет классы. Сама разделяющая гиперплоскость проходит ровно посередине полосы.

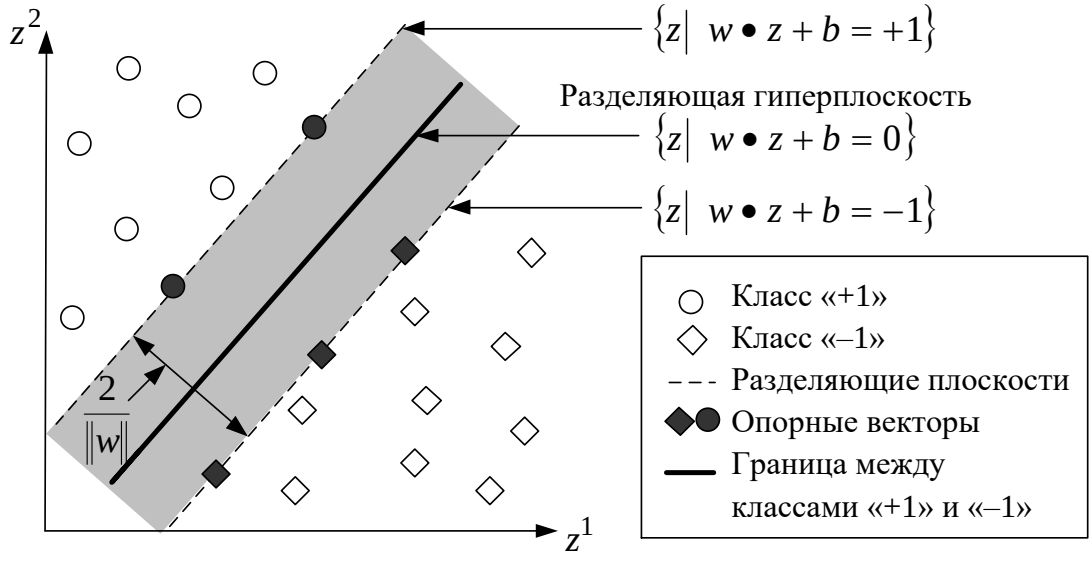

Рисунок 3 – Разделение классов гиперплоскостью (идеальный случай)

Границами полосы являются две параллельные гиперплоскости с направляющим вектором w. Ближайшие к параллельным гиперплоскостям объекты называются *опорными векторами*, они расположены точно на границах полосы разделяющей классы и несут всю информацию об их разделении.

Чем шире полоса, тем увереннее можно классифицировать объекты. Соответственно, в алгоритме опорных векторов считается, что самая широкая полоса является наилучшей. Ни один из объектов обучающей выборки не должен лежать внутри этой разделяющей полосы.

Так как для граничных объектов выполняется условие: $w \bullet z_i + b = y_i$, а остальные объекты находятся дальше, то для всех $z_i \in Z^*$ справедливы неравенства:

$$\begin{cases} w \bullet z_i + b \le -1, & \text{если} \quad y_i = -1 \\ w \bullet z_i + b \ge +1, & \text{если} \quad y_i = +1 \end{cases} \quad \text{или} \quad y_i\left(w \bullet z_i + b\right) \ge 1 \ (i = \overline{1,S}\,). \quad (2)$$

Искомое правило классификации может быть записано в виде:

$$A(z) = sign\ (w \bullet z + b). \quad (3)$$

Если предположить, что выборка линейно разделима, то существуют такие значения w и b, при которых функционал числа ошибок, записанный с учетом ограничений (2):

$$Q(w,b) = \frac{1}{s}\sum_{i=1}^{s}[y_i(w \bullet z_i + b) < 0] \quad (4)$$

принимает нулевое значение. Но тогда разделяющая гиперплоскость не единственна (рисунок 2, слева), поскольку существуют и другие положения разделяющей гиперплоскости, реализующие то же самое разбиение выборки. Идея решения заключается в оптимальном выборе значений w и b, чтобы разделяющая гиперплоскость максимально далеко отстояла от ближайших к ней объектов обоих классов (рисунок 2, справа).

Случай линейно разделимой выборки

Если обучающая выборка линейно разделима, то можно выбрать гиперплоскости таким образом, чтобы между ними не лежал ни один объект обучающей выборки и затем максимизировать расстояние между гиперплоско-

стями. Ширина полосы между гиперплоскостями равна $2/\|w\|$. Для более уверенной классификации ширину гиперполосы необходимо максимизировать, для этого необходимо минимизировать величину $\|w\|$ при условии (2). Эта задача соответствует задаче квадратичной оптимизации, которая имеет вид:

$$\begin{cases} \dfrac{1}{2}\|w\|^2 \to \min, \\ y_i(w \bullet z_i + b) \geq 1, \quad i = \overline{1,S} \end{cases}. \tag{5}$$

По теореме Каруша – Куна – Таккера задача (5) эквивалентна двойственной задаче поиска седловой точки функции Лагранжа:

$$\begin{cases} L(w,b;\lambda) = \dfrac{1}{2}\|w\|^2 - \sum\limits_{i=1}^{S} \lambda_i \left(y_i (w \bullet z_i + b) - 1 \right) \to \min\limits_{w,b} \quad \max\limits_{\lambda}, \\ \lambda_i \geq 0, \quad i = \overline{1,S}, \\ \lambda_i = 0 \quad \text{либо} \quad w \bullet z_i + b = y_i, \quad i = \overline{1,S}, \end{cases} \tag{6}$$

где λ_i – множитель Лагранжа ($i = \overline{1,S}$): $\lambda_i > 0$ для опорных векторов, для неопорных векторов $\lambda_i = 0$, поэтому суммирование производится только по опорным векторам. Коэффициент $1/2$ вводится для удобства дифференцирования. Последнее из трех условий называется *условием дополняющей нежесткости*.

Из задачи (6) видно, что для поиска седловой точки необходимо сначала минимизировать Лагранжиан $L(w,b;\lambda)$ по переменным w и b, а затем максимизировать его по множителям Лагранжа при условии $\lambda_i \geq 0$. Необходимым условием седловой точки является равенство нулю производных Лагранжиана $L(w,b;\lambda)$:

$$\frac{\partial L(w,b;\lambda)}{\partial w} = w - \sum_{i=1}^{S} \lambda_i y_i z_i = 0, \quad \Rightarrow \quad w = \sum_{i=1}^{S} \lambda_i y_i z_i ; \tag{7}$$

$$\frac{\partial L(w,b;\lambda)}{\partial b} = -\sum_{i=1}^{S} \lambda_i y_i = 0, \quad \Rightarrow \quad \sum_{i=1}^{S} \lambda_i y_i = 0 . \tag{8}$$

С учетом (7) и (8) задача (6) сводится к эквивалентной задаче квадратичного программирования, содержащей только множители Лагранжа:

$$\begin{cases} -L(\lambda) = -\sum\limits_{i=1}^{S}\lambda_i + \dfrac{1}{2}\sum\limits_{j=1}^{S}\sum\limits_{i=1}^{S}\lambda_i\lambda_j y_i y_j (z_i \bullet z_j) \to \min\limits_{\lambda}, \\ \lambda_i \geq 0, \quad i = \overline{1,S}, \\ \sum\limits_{i=1}^{S}\lambda_i z_i = 0, \end{cases} \tag{9}$$

где λ_i – множитель Лагранжа; z_i – объект из обучающей выборки; y_i – число (–1 или +1), характеризующее классовую принадлежность объекта z_i из обучающей выборки; $z_i \bullet z_j$ – скалярное произведение векторов характеристик объектов z_i и z_j (рассматриваемое в случае линейной разделимости объектов); S – число объектов в обучающей выборке, $i = \overline{1,S}$.

В задаче (25) минимизируется квадратичный функционал, имеющий неотрицательно определенную квадратичную форму, следовательно, выпуклый. Область, определяемая ограничениями-неравенствами и одним равенством, также выпуклая. Следовательно, задача (9) имеет единственное решение. В теории оптимизации существует много методов и алгоритмов решения этой задачи (например, градиентные методы, метод покоординатного спуска и т.д.). Если задача (9) решена и минимум достигается при $\lambda_i = \lambda_i^0$ ($i = \overline{1,S}$), то решение задачи поиска оптимальной гиперплоскости (то есть искомых оптимальных значений для w^0 и b^0) имеет вид:

$$w^0 = \sum_{i=1}^{S}\lambda_i^0 y_i z_i; \qquad b^0 = -\frac{1}{2}(\min_{y_i=+1}(w^0 \bullet z_i) + \max_{y_i=-1}(w^0 \bullet z_i)), \qquad \lambda_i^0 > 0. \tag{10}$$

В итоге правило классификации (19) может быть записано в виде:

$$A(z) = sign\left(\sum_{i=1}^{S}\lambda_i^0 y_i (z_i \bullet z) + b^0\right). \tag{11}$$

Следует отметить, что суммирование идет не по всей выборке объектов, а только по опорным векторам, для которых $\lambda_i \neq 0$. Именно это свойство разреженности является отличительной чертой SVM-алгоритма.

Случаи идеального разделения двух классов гиперплоскостью встречаются крайне редко. На практике линейная разделимость объектов двух различных классов обычно невозможна и применение SVM-алгоритма для линейно неразделимых наборов объектов приводит к неудовлетворительным результатам. В этом случае применяют подход, позволяющий алгоритму допускать ошибки на обучающих объектах, или подход, основанный на замене в приведенных формулах скалярного произведения нелинейной функцией ядра, т.е. скалярным произведением в пространстве с большей размерностью.

Случай линейно неразделимой выборки

Чтобы обобщить SVM-алгоритм на случай линейной неразделимости, ему позволяют допускать ошибки на обучающих объектах, но при этом требуют, чтобы ошибок было поменьше. Для этого вводят набор дополнительных переменных $\xi_i \geq 0$, характеризующих величину ошибки на объектах z_i ($i = \overline{1,S}$) (рисунок 4).

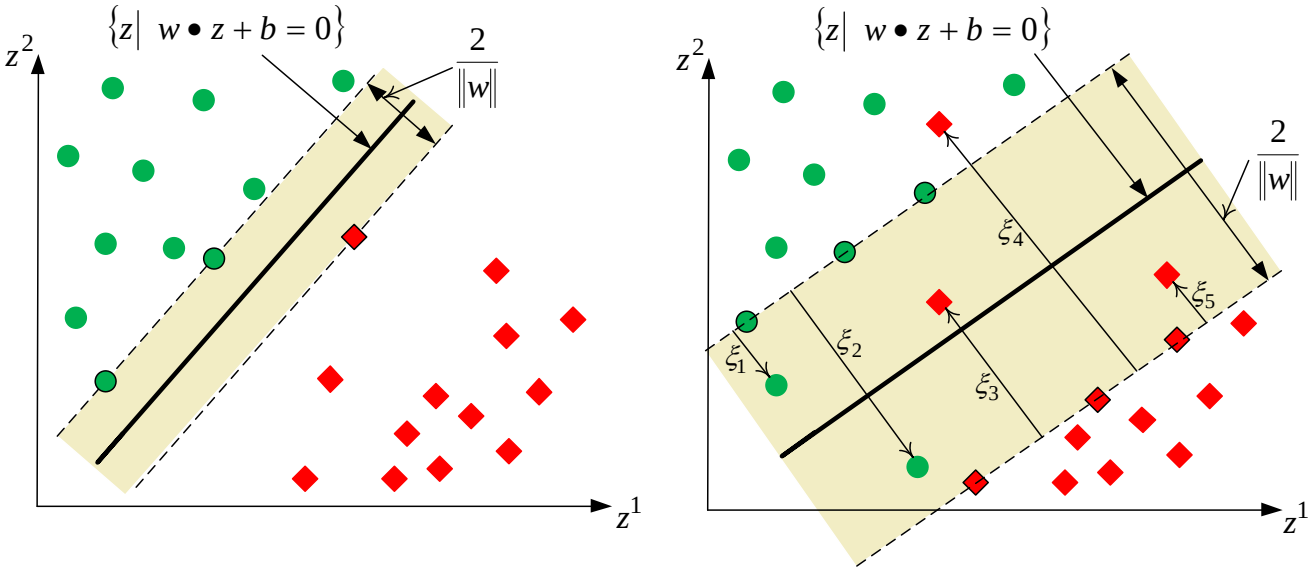

Рисунок 4 – Разделение классов: слева – случай линейной разделимости, справа – случай, когда объекты линейно неразделимы

Если взять за основу задачу (5), смягчив в ней второе ограничение-неравенство, и одновременно введя в минимизируемый функционал штраф за суммарную ошибку, то задача (5) может быть записана в виде:

$$\begin{cases} \dfrac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{S}\xi_i \to \min\limits_{w,b,\xi_i}, \\ y_i\left(w \bullet z_i + b\right) \geq 1 - \xi_i, \quad i = \overline{1,S}, \\ \xi_i \geq 0, \quad i = \overline{1,S}. \end{cases} \tag{12}$$

В задаче (12) минимизируемый функционал показывает, что среди векторов w предпочтительны векторы с минимальной нормой, при этом число ошибок при обучении должно быть минимально. Неравенство $y_i\left(w \bullet z_i + b\right) \geq 1 - \xi_i$ нарушает неравенство $y_i\left(w \bullet z_i + b\right) \geq 1$ в (2), при этом объект, нарушивший это неравенство, может оказаться: 1) все еще с нужной стороны гиперплоскости; 2) в «чужой» полуплоскости. Дополнительные переменные ξ_i, показывают, насколько объект вышел за пределы своей области. В задаче (28) C – параметр регуляризации ($C > 0$), который является управляющим параметром алгоритма, который позволяет находить компромисс между максимизацией разделяющей полосы и минимизацией суммарной ошибки.

При увеличении значения параметра C (то есть при уменьшении регуляризации) SVM-алгоритм определяет гиперплоскость таким образом, что объекты обучения становятся ближе к границам классификации и ширина гиперполосы уменьшается, что может приводить к неправильной классификации новых объектов вблизи границ (то есть к переобучению). При уменьшении значения параметра C SVM-алгоритм становится более консервативным, поскольку ширина гиперполосы увеличивается и ошибке классификации обучающих объектов присваивается меньшее значение. Таким образом, увеличение значения параметра C повышает точность на обучающих данных, но может привести к переобучению.

Параметр C регулирует, какое из двух слагаемых в минимизируемой функции требуется минимизировать: отвечающее за отступ или суммарную ошибку. Введение регуляризатора повышает устойчивость алгоритма по отношению к составу выборки и его обобщающую способность. В случаях, когда минимум функционала числа ошибок $Q(w,b)$ (4) достигается на множестве векторов w, регуляризация выбирает из них вектор с минимальной нормой. Тем самым устраняется проблема мультиколлинеарности, повышается устойчивость алгоритма, улучшается его обобщающая способность. Таким образом, принцип оптимальной разделяющей гиперплоскости или максимизации ширины разделяющей полосы тесно связан с регуляризацией некорректно поставленных задач по А.Н. Тихонову.

Отступом объекта z_i от границы классов называется величина $M_i(w,b) = y_i(w \bullet z_i + b)$. Искомое правило классификации (3) допускает ошибку на объекте z_i тогда и только тогда, когда отступ M_i отрицателен, при этом величина M_i определяет, насколько эта ошибка велика. Если $M_i \in (-1;+1)$, то объект z_i попадает внутрь разделяющей полосы. Если $M_i > 1$, то объект z_i классифицируется правильно и находится на некотором удалении от разделяющей полосы.

Как и в случае задачи (5), условия Каруша – Куна − Таккера сводят задачу (12) к поиску седловой точки функции Лагранжа:

$$\begin{cases} L(w,b,\xi;\lambda,\eta) = \dfrac{1}{2}\|w\|^2 - \sum\limits_{i=1}^{S}\lambda_i\left(M_i(w,b)-1\right) - \sum\limits_{i=1}^{S}\xi_i\left(\lambda_i+\eta_i-C\right) \to \min\limits_{w,b,\xi_i} \ \max\limits_{\lambda,\eta}; \\ \xi_i \ge 0, \quad \lambda_i \ge 0, \quad \eta_i \ge 0, \quad i=\overline{1,S}; \\ \lambda_i = 0 \quad \text{либо} \quad M_i(w,b) = 1-\xi_i, \quad i=\overline{1,S}; \\ \eta_i = 0 \quad \text{либо} \quad \xi_i = 0, \quad i=\overline{1,S}; \end{cases} \tag{13}$$

где λ_i − переменные, двойственные к ограничениям на отступы $M_i \ge 1-\xi_i$, η_i − переменные, двойственные к ограничениям $\xi_i \ge 0$, $i=\overline{1,S}$.

Необходимым условием седловой точки функции Лагранжа является равенство нулю ее частных производных:

$$\frac{\partial L(w,b,\xi;\lambda,\eta)}{\partial w} = w - \sum_{i=1}^{S} \lambda_i y_i z_i = 0, \quad \Rightarrow \quad w = \sum_{i=1}^{S} \lambda_i y_i z_i \; ; \qquad (14)$$

$$\frac{\partial L(w,b,\xi;\lambda,\eta)}{\partial b} = -\sum_{i=1}^{S} \lambda_i y_i = 0, \quad \Rightarrow \quad \sum_{i=1}^{S} \lambda_i y_i = 0 \; ; \qquad (15)$$

$$\frac{\partial L(w,b,\xi;\lambda,\eta)}{\partial \xi_i} = -\lambda_i - \eta_i + C = 0, \quad \Rightarrow \quad \eta_i + \lambda_i = C, \quad i = \overline{1,S} \; . \qquad (16)$$

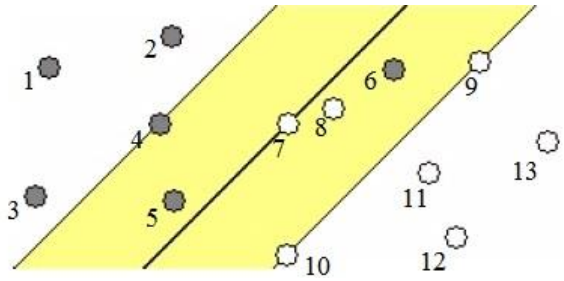

Рисунок 5 – Виды объектов: периферийные (1 – 3, 11 – 13), опорные (4, 9, 10), нарушители (5 – 8)

Из (16) и неравенства $\eta_i \geq 0$ можно получить, что: $0 \leq \lambda_i \leq C$. С учетом этого неравенства и из двух последних строк системы (13), в которых записаны условия дополняющей нежесткости, вытекает, что возможны только три сочетания значений переменных ξ_i, λ_i, η_i и отступов M_i. Соответственно, все объекты z_i ($i = \overline{1,S}$) делятся на три типа (рисунок 5).

1. $\lambda_i = 0$; $\eta_i = C$; $\xi_i = 0$ и $M_i \geq 1$. Объект z_i находится далеко от разделяющей полосы, классифицируется правильно и не влияет на решение. Такие объекты называют *периферийными*.

2. $0 < \lambda_i < C$; $0 < \eta_i < C$; $\xi_i = 0$ и $M_i = 1$. Объект z_i классифицируется правильно и лежит в точности на границе разделяющей полосы. Такие объекты, как и раньше, называют *опорными*. Только опорные объекты определяют

положение оптимальной разделяющей гиперплоскости, и если из выборки удалить все неопорные объекты, то ее положение не изменится.

3. $\lambda_i = C$; $\eta_i = 0$; $\xi_i > 0$ и $M_i < 1$. Объект z_i либо лежит внутри разделяющей полосы, но классифицируется правильно ($0 < \xi_i < 1$, $0 < M_i < 1$), либо попадает на границу классов ($\xi_i = 1$, $M_i = 0$), либо вообще относится к чужому классу ($\xi_i > 1$, $M_i < 0$). Во всех этих случаях объект z_i называют *нарушителем*.

Таким образом, задача (12) снова сводится к задаче квадратичного программирования относительно двойственных переменных λ_i. Единственное отличие от задачи с линейной разделимостью (9) состоит в появлении ограничения сверху для λ_i: $\lambda_i \le C$:

$$\begin{cases} -L(\lambda) = -\sum\limits_{i=1}^{S} \lambda_i + \frac{1}{2} \sum\limits_{j=1}^{S} \sum\limits_{i=1}^{S} \lambda_i \lambda_j y_i y_j (z_i \bullet z_j) \to \min\limits_{\lambda}, \\ 0 \le \lambda_i \le C, \quad i = \overline{1, S}, \\ \sum\limits_{i=1}^{S} \lambda_i y_i = 0. \end{cases} \tag{17}$$

Здесь минимизируется квадратичный функционал, имеющий неотрицательно определенную квадратичную форму, следовательно, выпуклый. Область, определяемая ограничениями неравенствами и одним равенством, также выпуклая. Следовательно, данная двойственная задача имеет единственное решение:

$$\begin{cases} w = \sum\limits_{i=1}^{S} \lambda_i y_i z_i; \\ b = y_i - w \bullet z_i, \ \text{для } i: \ \lambda_i > 0, M_i = 1. \end{cases} \tag{18}$$

На практике при реализации SVM-алгоритма решают именно задачу (17), а не (9), так как гарантировать линейную разделимость объектов на два класса в общем случае не представляется возможным. Этот вариант SVM-

алгоритма называют *алгоритмом с мягким зазором* (soft-margin SVM-алгоритм), тогда как в случае с линейной разделимостью говорят о жестком зазоре (hard-margin SVM-алгоритм).

Для правила классификации сохраняется формула (11) с той лишь разницей, что теперь ненулевыми λ_i ($\lambda_i \neq 0$) обладают не только опорные объекты, но и объекты-нарушители. В определенном смысле это недостаток, поскольку нарушителями часто оказываются шумовые выбросы, и построенное на них решающее правило, по сути дела, опирается на шум.

Ядра и спрямляющие пространства

SVM-алгоритм не ограничивается случаем линейного разделения объектов. На практике, расположение объектов относительно друг друга неизвестно и очень редко их можно разделить прямой линией или плоскостью. Но даже если, например, объекты располагаются в пространстве D-2 так, как на рисунке 6, слева, то и в этом случае возможно применение SVM-алгоритма.

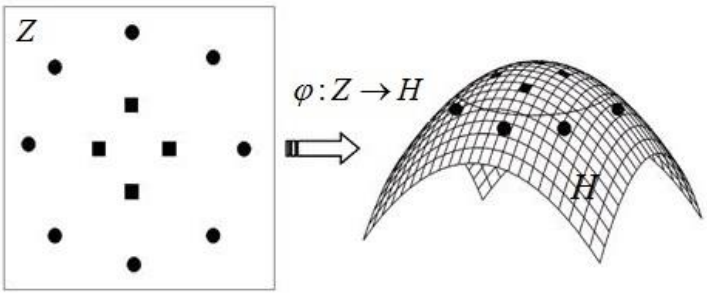

Рисунок 6 – Пример перехода к пространству более высокой размерности в случае линейной неразделимости объектов

Еще один подход к решению проблемы линейной неразделимости основан на переходе от исходного пространства характеристик объектов Z к новому пространству H с помощью некоторого преобразования $\varphi : Z \to H$. Если пространство H имеет достаточно высокую размерность, то можно надеяться, что в нем объекты удастся линейно разделить. Пространство H

называют *спрямляющим*. Если предположить, что характеристиками объектов z_i $(i=\overline{1,S})$ являются векторы $\varphi(z_i)=(\varphi(z_i^1),\varphi(z_i^2),\ldots,\varphi(z_i^q))$, а не векторы $z_i=(z_i^1,z_i^2,\ldots,z_i^q)$, то SVM-алгоритм реализуется точно так же, как и ранее. Единственное отличие состоит в том, что скалярное произведение $z\bullet z'$ в пространстве Z всюду заменяется скалярным произведением $\varphi(z)\bullet\varphi(z')$ в пространстве H. Отсюда вытекает требование: пространство должно быть наделено скалярным произведением.

Функция $\kappa: Z\times Z\to R$ называется *ядром* (kernel function), если она представима в виде $\kappa(z,z')=\varphi(z)\bullet\varphi(z')$ при некотором отображении $\varphi: Z\to H$, где H – пространство со скалярным произведением. Замена скалярного произведения ядром позволяет перевести задачу в пространство более высокой размерности, в котором объекты могут оказаться разделимыми.

На рисунке 6 приведен пример перехода к пространству более высокой размерности. Как видно, круглые и квадратные фигуры на плоскости (в пространстве D-2) невозможно разделить прямой линией. Но если «изогнуть» пространство, перейдя в пространство D-3, то эти фигуры можно разделить плоскостью, которая отрезает часть поверхности с квадратными фигурами. Таким образом, выгнув пространство с помощью отображения $\varphi(z)$, можно найти разделяющую гиперплоскость. Следовательно, SVM-алгоритм позволяет работать не только с линейно разделяемыми объектами, но и с объектами, которые линейно неразделимы.

На практике обычно выбирают не отображение $\varphi(z)$, а сразу функцию $\kappa(z,z')$, которая могла бы быть скалярным произведением при некотором отображении $\varphi(z)$. В качестве функции ядра $\kappa(z,z')$, позволяющей разделить объекты разных классов, обычно используется одна из следующих функций:

– полиномиальная (однородная):

$$\kappa(z,z')=(z\bullet z')^d; \qquad (19)$$

– полиномиальная (неоднородная):

$$\kappa(z, z') = (z \bullet z' + 1)^d; \qquad (20)$$

– радиальная базисная:

$$\kappa(z,z') = \exp\left(-\gamma \|z - z'\|^2\right); \qquad (21)$$

– радиальная базисная функция Гаусса:

$$\kappa(z,z') = \exp\left(-\frac{\|z - z'\|^2}{2 \cdot \sigma^2}\right); \qquad (22)$$

– сигмоидная:

$$\kappa(z, z') = th(k_2 + k_1(z \bullet z')), \qquad (23)$$

где $z \bullet z'$ – скалярное произведение векторов z и z'; d [$d \in N$ (по умолчанию $d = 3$)], γ [$\gamma > 0$ (по умолчанию $\gamma = 1$)], σ [$\sigma > 0$ (по умолчанию $\sigma^2 = 1$)], k_2 [$k_2 < 0$ (по умолчанию $k_2 = -1$)] и k_1 [$k_1 > 0$ (по умолчанию $k_1 = 1$)] – некоторые параметры; th – гиперболический тангенс.

Результирующий SVM-алгоритм очень похож на алгоритм линейной классификации с той лишь разницей, что каждое скалярное произведение в приведенных выше формулах заменяется нелинейной функцией ядра (скалярным произведением в пространстве с большей размерностью). Так, задача (17) принимает вид:

$$\begin{cases} -L(\lambda) = -\sum\limits_{i=1}^{S} \lambda_i + \frac{1}{2} \sum\limits_{j=1}^{S} \sum\limits_{i=1}^{S} \lambda_i \lambda_j y_i y_j \kappa(z_i, z_j) \to \min\limits_{\lambda}, \\ \sum\limits_{i=1}^{S} \lambda_i y_i = 0, \\ 0 \le \lambda_i \le C, \quad i = \overline{1, S}. \end{cases} \qquad (24)$$

В итоге правило классификации (19) будет записано в виде:

$$A(z) = sign\ (\sum_{i=1}^{S} \lambda_i^0 y_i \kappa(z_i, z)\ +\ b^0). \qquad (25)$$

В результате удается построить поверхность, которая разделяет объекты разных классов в новом пространстве более высокой размерности H.

Алгоритмы ближайшего соседа и *k*-ближайших соседей

Простейшим метрическим классификатором, основанном на оценивании сходства объектов, является *k*NN-классификатор, разрабатываемый на основе *k*NN-алгоритма (k Nearest Neighbors Algorithm), согласно которому классифицируемый объект относится к тому классу, к которому принадлежат ближайшие к нему k объектов обучающей выборки.

При разработке *k*NN-классификатора на основе *k*NN-алгоритма, как и при разработке SVM-классификатора на основе SVM-алгоритма, используется учебный набор данных: $U = \{< z_1, y_1 >, ..., < z_s, y_s >\}$, который также случайным образом разбивается на обучающую и тестовую выборки, состоящие соответственно из S и $s - S$ кортежей ($s > S$) с целью реализации многократного обучения и тестирования формируемых *k*NN-классификаторов с последующим определением лучшего классификатора в смысле обеспечения максимально возможной точности классификации. При этом для каждого *k*NN-классификатора определяется значение числа соседей k, при котором ошибка классификации минимальна. Класс принадлежности $y_i \in Y$ объекта $z_i \in Z$ определяется классом принадлежности большинства объектов из числа k ближайших соседей объекта $z_i \in Z$.

Реализация *k*NN-алгоритма для определения класса принадлежности произвольного объекта z при фиксированном числе k ближайших соседей предполагает выполнение следующей последовательности шагов.

1. Вычислить расстояние $d(z, z_i)$ от объекта z до каждого из объектов z_i, классовая принадлежность которых известна.

2. Выполнить упорядочение вычисленных расстояний по возрастанию их значений.

3. Выбрать k объектов z_i (k ближайших соседей), наиболее близко расположенных к объекту z.

4. Выявить классовую принадлежность каждого из k ближайших соседей объекта z.

5. Установить для произвольного объекта z в качестве его класса принадлежности класс, наиболее характерный для его k ближайших соседей.

Для оценки расстояния между объектами в kNN-алгоритме могут использоваться различные меры расстояния, такие как евклидова мера, манхэттенская мера, косинусная мера и др. При этом наиболее часто используется евклидова мера расстояния:

$$d(z,\ z_i)=(\sum_{l=1}^{q}(z^l - z_i^l)^2)^{1/2}, \qquad (26)$$

где q – число характеристик объектов z и z_i; z_i^l – числовое значение l-й характеристики для i-го объекта ($l=\overline{1,q}$) нормированное значениями [0; 1].

При реализации kNN-алгоритма в качестве способов голосования могут применяться простое невзвешенное голосование и взвешенное голосование.

При использовании простого невзвешенного голосования расстояние от объекта z до каждого из k объектов-ближайших соседей z_i ($i=\overline{1,k}$) не играет роли: все k объектов-ближайших соседей z_i ($i=\overline{1,k}$) имеют равные права в определении класса объекта z. Каждый из k объектов-ближайших соседей z_i ($i=\overline{1,k}$) объекта z голосует за его отнесение к своему классу y_{z,z_i}. В результате реализации kNN-алгоритма объект z будет отнесен к тому классу, который наберет большее число голосов:

$$\alpha = \arg\max_{y\in Y}\sum_{i=1}^{k}|\ y_{z,z_i} = y\ |. \qquad (27)$$

При использовании взвешенного голосования учитывается расстояние от объекта z до каждого из k объектов-ближайших соседей z_i ($i=\overline{1,k}$): чем меньше расстояние, тем более значимый вклад в оценку принадлежности объекта z к некоторому классу вносит голос объекта-соседа z_i ($i=\overline{1,k}$).

Оценка суммарного вклада голосов объектов-соседей z_i ($i=\overline{1,k}$) за принадлежность объекта z классу $y \in Y$ при взвешенном голосовании может быть рассчитана как:

$$\alpha = \sum_{i=1}^{k} \frac{1}{d^2(z,\ z_i)} \cdot \rho_i , \qquad (28)$$

где $\rho_i = 0$, если $y_{z,z_i} \neq y$ и $\rho_i = 1$, если $y_{z,z_i} = y$.

Класс, которому соответствует наибольшее значение оценки (28), назначается рассматриваемому объекту z.

При невзвешенном голосовании расстояние между объектами z_i и z может быть вычислено на основе функции ядра:

$$d^2(z,\ z_i) = \kappa(z,\ z) - 2 \cdot \kappa(z,\ z_i) + \kappa(z_i,\ z_i) . \qquad (29)$$

Достоинства *k*NN-алгоритма:

– устойчивость к аномальным выбросам;

– простота программной реализации;

– легкая интерпретируемость;

– модифицируемость алгоритма на основе использования наиболее подходящих метрик.

Недостатки *k*NN-алгоритма:

– требования к репрезентативности набора данных;

– модель нельзя «отделить» от данных: для классификации нового объекта необходимо использовать все объекты (эта особенность сильно ограничивает использование алгоритма).

Регрессионные модели

Термину регрессионная модель, используемому в регрессионном анализе, можно сопоставить синонимы: «теория», «гипотеза». Эти термины пришли из

статистики, в частности из раздела «проверка статистических гипотез». Регрессионная модель есть прежде всего гипотеза, которая должна быть подвергнута статистической проверке, после чего она принимается или отвергается.

Различают математическую модель и регрессионную модель. Математическая модель предполагает участие аналитика в конструировании функции, которая описывает некоторую известную закономерность. Математическая модель является интерпретируемой – объясняемой в рамках исследуемой закономерности. При построении математической модели сначала создаётся параметрическое семейство функций, затем с помощью измеряемых данных выполняется идентификация модели – нахождение её параметров. Известная функциональная зависимость объясняющей переменной и переменной отклика – основное отличие математического моделирования от регрессионного анализа. Недостаток математического моделирования состоит в том, что измеряемые данные используются для верификации, но не для построения модели, вследствие чего можно получить неадекватную модель. Также затруднительно получить модель сложного явления, в котором взаимосвязано большое число различных факторов.

Регрессионная модель объединяет широкий класс универсальных функций, которые описывают некоторую закономерность. При этом для построения модели в основном используются измеряемые данные, а не знание свойств исследуемой закономерности. Такая модель часто не интерпретируема, но более точна. Это объясняется либо большим числом моделей-претендентов, которые используются для построения оптимальной модели, либо большой сложностью модели. Нахождение параметров регрессионной модели называется обучением модели.

Недостатки регрессионного анализа: модели, имеющие слишком малую сложность, могут оказаться неточными, а модели, имеющие избыточную сложность, могут оказаться переобученными.

С использованием регрессионных моделей может выполняться прогнозирование значений зависимых показателей.

Примеры регрессионных моделей: линейные функции, алгебраические полиномы, ряды Чебышёва, нейронные сети без обратной связи, например, однослойный персептрон Розенблатта, радиальные базисные функции и прочее.

Регрессионные модели могут быть реализованы и с применением, например, SVM-алгоритма, RF-алгоритма.

И регрессионная, и математическая модель, как правило, задают непрерывное отображение. Требование непрерывности обусловлено классом решаемых задач: чаще всего это описание физических, химических и других явлений, где требование непрерывности выставляется естественным образом. Иногда на *отображение* накладываться ограничения монотонности, гладкости, измеримости, и некоторые другие. Теоретически, никто не запрещает работать с функциями произвольного вида, и допускать в моделях существование не только точек разрыва, но и задавать конечное, неупорядоченное множество значений свободной переменной, то есть, превращать задачи регрессии в задачи классификации.

При решении задач регрессионного анализа встают следующие вопросы.

1. Как выбрать тип и структуру модели, какому именно семейству она должна принадлежать?
2. Какова гипотеза порождения данных, каково распределение случайной переменной?
3. Какой целевой функцией оценить качество аппроксимации?
4. Каким способом отыскать параметры модели, каков должен быть алгоритм оптимизации параметров?

Логистическая регрессия (Logistic Regression)

Логистическая регрессия – это разновидность множественной регрессии, предназначенная для классификации записей на основании значений входных полей.

Линейную регрессию можно представить в виде уравнения, которое описывает прямую, наиболее точно показывающую взаимосвязь между входными переменными и выходными переменными. Для составления этого уравнения нужно найти определённые коэффициенты для входных переменных. Если уравнение прямой представить уравнением $y = b_0 + b_1 * z$, где y – зависимая переменная, а z – независимая переменная, тогда базовые теории исчисления применяются для определения значений для b_0 и b_1 с использованием заданного набора данных.

Функцию линейной регрессии в общем виде можно представить через следующую зависимость:

$$f(b, x) = b_0 + b_1 * z_1 + ... + b_n * z_n, \quad (30)$$

где b_n – параметры (коэффициенты) регрессии, z_n – регрессоры (факторы модели), n – количество факторов модели. Параметр b_0, называемый константой, формально отражает значение функции при нулевом значении всех факторов.

Существует два типа линейной регрессии: простая линейная регрессия с одной независимой переменной и множественная линейная регрессия, где используется несколько независимых переменных.

Логистическая регрессия (логит-регрессия, *Logit model*) представляет собой статистическую модель, используемую для прогнозирования вероятности возникновения некоторого события путём подгонки данных к логистической кривой.

В отличие от линейной регрессии, в логистической регрессии значение функции отражает вероятность принадлежности исходного значения числовой переменной к определенному классу. Таким образом, результат логистической регрессии всегда находится в интервале [0, 1].

Основной задачей логистической регрессии является прогнозирование вероятности возникновения некоторого события по значениям имеющегося набора характеристик. При этом определяется зависимая переменная y (категориальная или бинарная), принимающая лишь одно из двух значений («0» – событие не произошло и «1» – событие произошло), а также множество независимых переменных (предикторов или регрессоров), принимающих вещественные значения $z_1, z_{2,\dots,} z_n$. На основе значений регрессоров проводится вычисление вероятности принятия того или иного значения зависимой переменной.

Вероятность наступления события $y = 1$ может быть выражена по формуле:

$$P\{y = 1 \mid z\} = f(\gamma) = \frac{1}{1 + e^{-\gamma}}, \tag{31}$$

где $\gamma = \theta^T = \theta_0 + \theta_1 * z_1 + \dots + \theta_n * z_n$ – логистическая функция (сигмоид или логит-функция), θ_n – параметры (коэффициенты) регрессии, z_n – регрессоры (факторы модели), n – количество факторов модели.

Соответственно вероятность принятия значения $y = 0$:

$$P\{y = 0 \mid z\} = 1 - f(\gamma). \tag{32}$$

Логистическая регрессия используется для решения задачи классификации, а не регрессии в отличие от линейной регрессии. Обучается с использованием методов оптимизации, в частности, методом градиентного спуска.

Достоинства логистической регрессии:

– на основе логистической регрессии рассчитываются вероятности отнесения к разным классам (что является ценным в решении ряда прикладных задач, например, в задаче кредитного скоринга);

– подход логистической регрессии хорошо изучен;

– характеризуется быстродействием и работоспособностью на достаточно больших выборках.

Недостатки логистической регрессии:

– качество решения низкое в задачах, в которых зависимость ответов от характеристик сложная, нелинейная;

– наследование недостатков метода стохастического градиента в градиентном методе обучения логистической регрессии.

Наивный байесовский классификатор (Naive Bayes, NB)

Наивный байесовский алгоритм (NB-алгоритм) – это алгоритм классификации, основанный на теореме Байеса с допущением о независимости характеристик объекта. Другими словами, NB-алгоритм предполагает, что наличие какой-либо характеристики в классе не связано с наличием какой-либо другой характеристики.

Теорема Байеса позволяет рассчитывать апостериорные вероятности классов. Объект относится к тому классу, для которого апостериорная вероятность максимальна. Идея наивного байесовского классификатора описана в следующем формульном описании:

$$P(y \mid z_1, z_{2,}..., z_n) = \frac{P(y)P(z_1, z_2,..., z_n \mid y)}{P(z_1, z_2,..., z_n)}, \quad (33)$$

$$P(y \mid z_1, z_{2,}..., z_n) = P(y)P(z_1 \mid y)P(z_2 \mid y)P(z_3 \mid y) \times ... \times P(z_n \mid y), \quad (34)$$

$$P(y \mid z_1, z_{2,}..., z_n) = P(y)\prod_{i=1}^{n} P(z_i \mid y), \quad (35)$$

где $P(y \mid z_1, z_2, ..., z_n)$ – апостериорная вероятность принадлежности объекта $z_i = (z_1, .z_{2,} ..., z_n)$ (n – количество характеристик объекта) к классу y; $P(y)$ – вероятность отнесения объекта к классу y, априорная вероятность данного класса; $P(z_1, z_2, ..., z_n \mid y)$ – правдоподобие, т.е. вероятность возникновения в классе y объекта z_i с набором характеристик $z_i = (z_1, .z_{2,} ..., z_n)$; $P(z_1, z_2, ..., z_n)$ – априорная вероятность возникновения объекта z_i с набором характеристик $z_i = (z_1, .z_{2,} ..., z_n)$.

Достоинства NB-алгоритма:

– простота и высокая скорость классификации, в том числе многоклассовой;

– NB-алгоритм превосходит другие алгоритмы, такие как логистическая регрессия, и при этом требует меньший объем обучающих данных (при условии выполнения допущения о независимости);

– NB-алгоритм лучше работает с категорийными характеристиками объектов, чем с непрерывными; для непрерывных признаков предполагается нормальное распределение, что является достаточно сильным допущением.

Недостатки NB-алгоритма:

– алгоритму присуще явление под названием «нулевая частота» (zero frequency): при условии, что в тестовом наборе данных присутствует некоторое значение категорийной характеристики объекта, которое не встречалось в обучающем наборе данных, модель присвоит нулевую вероятность этому значению и не сможет сделать прогноз (данную проблему можно решить с помощью сглаживания);

– значения спрогнозированных вероятностей не всегда являются достаточно точными;

– допущение о независимости характеристик (в реальности наборы полностью независимых признаков встречаются крайне редко).

Широкое применение NB-алгоритм находит в задачах фильтрации спама (идентификация спама в электронных письмах) и анализа тональных характеристик текста (идентификация позитивных и негативных мнений клиентов).

Алгоритм *k*-ближайших соседей находит применение в ряде практических задач, в частности:

– обнаружение мошенничества (распознавание новых случаев мошенничества для проведения дальнейшей экспертизы);

– предсказание отклика клиентов (прогноз отклика новых клиентов по историческим данным);

– медицина (классификация пациентов по разным характеристикам на основании данных прошлых периодов).

Алгоритм случайного леса (Random Forest, RF)

Решающее дерево (Decision Tree, DT) представляет собой логический алгоритм в решении задач классификации, выполняющий поиск конъюнктивных закономерностей.

Деревья решений являются способом представления правил в иерархической структуре, состоящей в нелистовых вершинах (узел) и некотором заключении о целевой функции в листовых вершинах (прогноз). Решающее правило представляет функцию для определения, в какую из дочерних вершин необходимо поместить рассматриваемый объект. В листовых вершинах могут находиться разные объекты: класс, который нужно присвоить попавшему туда объекту (в задаче классификации), вероятности классов (в задаче классификации), непосредственно значение целевой функции (задача регрессии).

Несмотря на определенные преимущества DT-алгоритма (легкую интерпретируемость; отсутствия требования к подготовке данных, в частности, нормализации, удаления пропущенных данных; способности работать с разными

типами данных: например, категориальными, интервальными значениями характеристик объектов), данный алгоритм имеет существенные недостатки:

– единственно оптимальное решение выбирается локально в каждом узле, что не обеспечивает оптимальность всего дерева в целом;

– создание достаточно сложных структур в процессе построения дерева, что является следствием переобучения алгоритма (для решения данной проблемы используется регулировка глубины дерева).

На сегодняшний день решающие деревья на практике нечасто используются как отдельные методы классификации (или регрессии). В то же время, как оказалось, они очень хорошо объединяются в композиции – решающие леса, которые являются одними из наиболее сильных и универсальных моделей.

Алгоритм случайного леса, или RF-алгоритм, реализует использование комитета (ансамбля) решающих деревьев для принятия решения о классовой принадлежности объекта. В RF-алгоритме за счет объединения большого числа простых (в том числе «слабых») классификаторов – решающих деревьев – достигается высокое качество классификации. Итоговый результат классификации получается на основе агрегирования ответов множества деревьев. Ансамбль решающих деревьев позволяет (по сравнению с одним решающим деревом) ослабить проблему переобучения и повысить точность классификации. RF-алгоритм сочетает в себе идеи метода бэггинга (bagging, bootstrap aggregating) и метода случайных подпространств (RSM, random subspace method). В RF-алгоритме, как и в методе бэггинга, обучение классификаторов происходит независимо друг от друга на разных подмножествах обучающей выборки, что решает проблему построения одинаковых деревьев на одном и том же множестве данных. В результате классификации объекту будет присвоен тот класс, за который проголосовало большинство деревьев, при условии, что одно дерево обладает одним голосом. Как и в методе случайных подпространств, в RF-алгоритме базовые классификаторы – решающие деревья – обучаются по случайным подмножествам характеристик.

При реализации RF-алгоритма важным является определение значений его параметров. К основным параметрам RF-алгоритма относятся: число деревьев r, число характеристик для выбора расщепления $\max\_ fs$, максимальная глубина деревьев dp_t, критерий расщепления cr. При этом в качестве критерия расщепления вершины дерева при решении задачи классификации может быть использован критерий Джини:

$$Gini_t = 1 - \sum_{j=1}^{v} P^2(Y_h), \qquad (36)$$

где $P(Y_h)$ – доля объектов Y_h-го класса в подвыборке, связанной с вершиной дерева t; $h = \overline{1, v}$ (при бинарной классификации $v = 2$, $Y = \{-1, +1\}$); $P(Y_h) = l^{Y_h} / N$; l^{Y_h} – число объектов Y_h-го класса в подвыборке, N – число объектов в подвыборке.

Построение решающего дерева в RF-алгоритме может быть описано следующей последовательностью шагов.

Шаг 1. Для каждого j-го дерева ($j = \overline{1, r}$) из обучающей выборки выбирается подвыборка z_{rf}^{j}, содержащая S_{rf} $(S_{rf} < S)$ объектов. При этом формирование подвыборки z_{rf}^{j} может проводиться с применением метода бутстрэпа (*bootstrap*) – метода имитации статистического выбора). Суть метода бутсрэпа заключается в формировании множества подвыборок на основе случайного выбора с повторениями объектов.

При генерации случайных подвыборок z_{rf}^{j} с повторениями часть объектов из обучающей выборки не попадёт в подвыборку z_{rf}^{j}. Эти объекты будут определять множество «неотобранных» объектов $(out - of - bag)$, число которых в среднем будет составлять S/e, где S $(s > S)$ – число объектов в обучающей выборке; $i = \overline{1, S}$; $e \approx 2{,}71828$ – основание натурального логарифма.

В результате выполнения шага 1 для каждого j-го дерева ($j = \overline{1,r}$) формируется собственная подвыборка объектов z_{rf}^{j}.

Шаг 2. Выполняется расщепление r построенных деревьев. Для каждого расщепления в дереве рассматривается max_ fs характеристик (при этом рекомендуется выбирать max_ fs так, что max_ $fs = \sqrt{n}$), случайным образом выбранных из числа характеристик n, где n – исходное число характеристик объектов обучающей выборки. Выбирается «лучшая» характеристика, по которой на текущем шаге происходит оптимальное расщепление вершины j-го дерева в соответствии с критерием cr. В случае применения индекса Джини в качестве критерия cr оптимальным считается то расщепление вершины дерева, для которого значение критерия минимально. При бинарной классификации показатель качества расщепления оценивается следующим образом:

$$Gini_t^{split} = \frac{N_1}{N} Gini_{t1} + \frac{N_2}{N} Gini_{t2} \to \min, \qquad (37)$$

где N – число объектов в текущей вершине t дерева (вершине-предке); N_1, N_2 – число объектов в вершинах t_1, t_2, соответствующих левому и правому вершинам-потомкам в случае построения бинарного дерева.

Шаг 3. Проводится построение j-го дерева до исчерпания подвыборки z_{rf}^{j} (пока в листьях не останутся представители только одного класса) или до достижения заданного значения глубины дерева dp_t.

Итоговый RF-классификатор (комитет) $a(z_{rf})$ выбирает решение по большинству голосов построенных решающих деревьев:

$$a(z_{rf}) = sign \sum_{j=1}^{r} b(z_{rf}^{j}), \qquad (38)$$

где $b(z_{rf}^{j})$ – решение базового классификатора j-го дерева ($j = \overline{1,r}$).

Достоинства RF-алгоритма:

– случайные леса обеспечивают существенное повышение точности, так как деревья в ансамбле слабо коррелированы вследствие двойной инъекции случайности в индуктивный алгоритм – посредством бэггинга и использования метода случайных подпространств при расщеплении каждой вершины;

– отсутствует проблема переподгонки (даже при количестве характеристик, превышающем количество наблюдений обучающей выборки и большом количестве деревьев), тем самым снимается сложная проблема отбора характеристик, необходимая для других ансамблевых классификаторов;

– способность эффективно обрабатывать данные с большим числом характеристик и классов;

– наличие методов оценивания значимости отдельных характеристик в модели;

– возможность оценки обобщенной способности модели, которая учитывает ошибку RF-классификатора при построении на «неотобранных» объектах $(out-of-bag)$ (при этом минимизация оценки ошибки $out-of-bag$ проводится при определении оптимального числа деревьев).

Недостатки RF-алгоритма:

– при использовании больших наборов данных требуется значительные временные затраты на построение ансамбля деревьев;

– ограничение на глубину отрицательно влияет на общую точности классификации (при этом существует линейная зависимость времени обучения деревьев от их числа).

Ансамбли классификаторов. Каскадные классификаторы. Регрессионные модели на основе ансамблей

Ансамбль использует несколько обучающих алгоритмов с целью получения лучшей эффективности при классификации (прогнозировании), чем

могли бы получить от каждого обучающего алгоритма по отдельности. Ансамбль алгоритмов состоит из конкретного конечного множества альтернативных моделей, но, обычно, позволяет существовать существенно более гибким структурам.

Алгоритмы обучения с учителем наиболее часто описываются как осуществление задачи поиска в пространстве гипотез для нахождения подходящей гипотезы, которая делает хорошие предсказания для конкретной задачи. Даже если пространство гипотез содержит гипотезы, которые очень хорошо подходят для конкретной задачи, может оказаться трудной задачей найти хорошую гипотезу. Ансамбль методов комбинирует несколько гипотез в надежде образовать гипотезу лучше. Термин ансамбль обычно резервируется для методов, которые генерируют несколько гипотез с помощью одного и того же базового учителя. Более широкое понятие системы множественных классификаторов также использует несколько гипотез, но сгенерированных не с помощью одного и того же учителя.

Вычисление предсказания ансамбля обычно требует больше вычислений, чем предсказание одной модели, так что ансамбли можно рассматривать как способ компенсации плохого алгоритма обучения путём дополнительных вычислений. В ансамбле методов обычно используются быстрые алгоритмы, такие как деревья решений (например, случайные леса), хотя медленные алгоритмы могут получить также преимущества от техники сборки в ансамбль.

Ансамбль сам по себе является алгоритмом обучения с учителем, поскольку он может быть тренирован и затем использован для осуществления предсказания. Тренированный ансамбль, поэтому, представляет одну гипотезу. Эта гипотеза, однако, не обязательно лежит в пространстве гипотез моделей, из которых она построена. Таким образом, ансамбли могут иметь большую гибкость в функциях, которые они могут представлять. Эта гибкость мо-

жет, в теории, быстрее привести их к переобучению по тренировочным данным, чем могло быть в случае отдельной модели, но, на практике, некоторые техники сборки в ансамбль (особенно бэггинг) склонны уменьшить проблемы, связанные с переобучением на тренировочных данных.

Эмпирически, ансамбли склонны давать результаты лучше, если имеется существенное отличие моделей. Многие ансамбли методов, поэтому, стараются повысить различие в моделях, которые они комбинируют. Хотя, возможно, неинтуитивные, более случайные алгоритмы (подобные случайным деревьям решений) могут быть использованы для получения более строгих ансамблей, чем продуманные алгоритмы (такие как деревья решений с уменьшением энтропии). Использование различных алгоритмов строгого обучения, однако, как было показано, более эффективно, чем использование техник, которые пытаются упростить модели с целью обеспечить большее различие.

В то время как число классификаторов в ансамбле имеют большое влияние на точность предсказания, имеется лишь ограниченное число статей, изучающих эту проблему. Определение априори размера ансамбля и размеров скорости больших потоков данных делает этот фактор даже более критичным для онлайновых ансамблей классификаторов. Большинство статистических тестов были использованы для определения подходящего числа компонент. Относительно недавно теоретический фреймворк дал повод предположить, что имеется идеальное число классификаторов ансамбля, такое, что число классификаторов больше или меньше этого идеального числа приводит к ухудшению точности. Это называется «законом убывания отдачи в построении ансамбля». Этот теоретический фреймворк показывает, что использование числа независимых классификаторов, равного числу меток класса, даёт наибольшую точность.

Байесовский оптимальный классификатор – это техника классификации. Он является ансамблем всех гипотез из пространства гипотез. В среднем ни

один из ансамблей не может превосходить его. Простой байесовский оптимальный классификатор – это версия, которая предполагает, что данные условно независимы от класса, и выполняет вычисления за более реальное время. Каждой гипотезе даётся голос, пропорциональный вероятности того, что тренировочные данные будут выбраны из системы, если гипотеза была бы верна. Для получения тренировочных данных конечного размера голос каждой гипотезы умножается на априорную вероятность такой гипотезы.

Бэггинг

Бутстрэп-агрегирование, часто сокращаемое до бэггинг, даёт каждой модели в ансамбле одинаковый вес (голос). Чтобы поддерживать вариантность, бэггинг тренирует каждую модель в ансамбле с помощью случайно отобранного подмножества из тренировочного множества. Как пример, алгоритм «случайного леса» комбинирует случайные деревья решений с бэггингом, чтобы получить высокую точность классификации.

Бустинг

Бустинг строит ансамбль последовательными приращениями путём тренировки каждой новой модели, чтобы выделить тренировочные экземпляры, которые предыдущие модели классифицировали ошибочно. В некоторых случаях бустинг, как было показано, даёт лучшие результаты, чем бэггинг, но имеет тенденцию к переобучению на тренировочных данных. Наиболее частой реализацией бустинга является алгоритм AdaBoost, хотя есть утверждения, что некоторые более новые алгоритмы дают лучшие результаты.

Каскадные классификаторы могут быть легко реализованы технологий бустинга. Также при формировании каскадных ансамблей могут быть реализованы и другие подходы. В частности, при использовании SVM-алгоритма можно сначала на первом уровне каскада минимизировать число анализируемых объектов в обучающей выборке (например, посредством отбора опорных

векторов нескольких SVM-классификаторов), а затем на втором уровне каскада выполнить разработку SVM-классификатора с использованием набора данных, содержащего только объекты, которые потенциально могут быть объектами, лежащими на границе классов). В этом случае можно не только минимизировать время разработки классификатора, но и обеспечить высокое качество принимаемых классификационных решений.

При ансамблировании регрессионных моделей искомое решение может быть найдено в результате простого или взвешенного усреднения решений, предложенных разным моделями.