БИОФИЗИКА, 2015, том 60, вып. 1, с. 18–31

МОЛЕКУЛЯРНАЯ БИОФИЗИКА

УДК 577.3

СРАВНИТЕЛЬНАЯ ЭФФЕКТИВНОСТЬ АЛГОРИТМОВ ПОСТРОЕНИЯ МАШИН ОПОРНЫХ ВЕКТОРОВ ДЛЯ ЗАДАЧИ БИНАРНОЙ КЛАССИФИКАЦИИ

© 2015 г. Н.О. Кадырова, Л.В. Павлова

Институт прикладной математики и механики Санкт-Петербургского государственного политехнического университета, 195251, Санкт-Петербург, ул. Политехническая, 29

E-mail: natalia.kadyrova@gmail.com

Поступила в редакцию 09.06.14 г.

Методы построения машин опорных векторов не требуют дополнительной априорной информации и позволяют обрабатывать большие объемы данных большой размерности, что особенно важно для многих проблем вычислительной биологии. Представлены основные алгоритмы построения машин опорных векторов для задачи бинарной классификации. Рассмотрен вопрос качества алгоритмов обучения. Приведено описание представленных алгоритмов, достаточное для их практической реализации. Исследована сравнительная эффективность ряда SV-классификаторов.

Ключевые слова: бинарная классификация, машины опорных векторов, ядерные машины, SVM-алгоритмы, сравнительная эффективность SV-классификаторов.

Основные принципы и идеи современного подхода к задачам бинарной классификации и восстановления регрессии, основанного на машинах опорных векторов, рассмотрены нами в работе [1]. SVM-методы относятся к технологии, названной интеллектуальным анализом данных, и замечательны тем, что практически не требуют дополнительной априорной информации и позволяют обрабатывать большие объемы данных большой размерности.

Новый индукционный принцип обучения по конечным выборкам – структурная минимизация риска – приводит к задаче минимизации регуляризованного риска [2]. Ряд SVM-алгоритмов непосредственно решает эту задачу, используя ядерные машины, например, алгоритм Naive Online Risk Minimization Algorithm (NORMA).

Стандартная постановка задачи бинарной классификации позволяет свести исходную задачу минимизации верхней границы ожидаемого риска к задаче квадратичного программирования с ограничениями типа равенств относительно двойственных переменных, полностью определяющих искомое решающее правило [2]. Непосредственное решение двойственной задачи квадратичного программирования обычно невозможно из-за ее громадной размерности. Основные подходы к построению непрямого решения этой задачи опираются на свойство разреженности, присущее SV-машинам. Ряд SVM-алгоритмов основан на декомпозиции, т.е. разбиении исходной задачи квадратичного программирования на серию подзадач существенно меньшей размерности, например, алгоритмы SVM_Light и Sequential minimal optimization (SMO).

В настоящей работе рассмотрены основные алгоритмы построения машин опорных векторов для задачи бинарной классификации и исследована их сравнительная эффективность.

КАЧЕСТВО АЛГОРИТМОВ ОБУЧЕНИЯ

Оценивание характеристики обобщения (generalization performance) различных алгоритмов, использующих некоторое выборочное множество, является существенным моментом SVM-подхода. При заданной тренировочной последовательности важно определить, насколько хорошо конкретный алгоритм обучения будет работать на новых данных, т.е. какова его способность к обобщению. Располагая подобной информацией, можно решать вопрос о выборе алгоритма, модели, значений параметров обучения.

На практике обычно используют методы оценивания ошибки обобщения, основанные на процедуре k-кратной перекрестной проверки (k-fold cross-validation (CV)). Однако теоретические аспекты этого подхода мало изучены. Наиболее популярной разновидностью CV-ошибки является так называемая leave-one-out (loo)

ошибка (скользящий контроль), полученная после применения l-кратной перекрестной проверки, где l – объем тренировочной последовательности. loo-Ошибка как оценка ошибки обобщения является важной статистической оценкой качества алгоритмов обучения. В работе [3] приведены результаты, позволяющие обосновать использование loo-ошибки при обучении машин.

Пусть X – пространство входных объектов $\boldsymbol{x}$ с фиксированным неизвестным распределением вероятностей $P(\boldsymbol{x})$; Y – пространство выходных объектов y с фиксированным неизвестным распределением вероятностей $P(y/\boldsymbol{x})$. D – тренировочная последовательность объема l, т.е. выборка независимых, одинаково распределенных согласно закону $P(\boldsymbol{x},y) = P(\boldsymbol{x}) \times P(y/\boldsymbol{x})$ наблюдений $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)$. D^i – тренировочная последовательность объема l–1, полученная из D путем удаления i-го образца $(\boldsymbol{x}_i, y_i)$. $f_D : X \to R$ – решение, полученное с использованием данного алгоритма обучения на основе выборки D. Пусть функция потерь L : $R \times Y \to R$, $L = L(f(\boldsymbol{x}),y) \equiv L(f,(\boldsymbol{x},y))$, штрафует отклонение оценок $f(\boldsymbol{x})$ от наблюдаемых y.

В качестве ошибки обобщения данного алгоритма обучения относительно функции потерь L рассматривают ожидаемый риск

$$R[f_D] = \int_{X \times Y} L(f_D,(\boldsymbol{x},y))dP(\boldsymbol{x},y).$$

В качестве эмпирической ошибки данного алгоритма обучения относительно функции потерь L можно использовать как эмпирический риск

$$R_{\text{emp}}[f_D] = \frac{1}{l}\sum_{i=1}^{l} L(f_D,(\boldsymbol{x}_i,y_i)),$$

так и leave-one-out-ошибку

$$R_{\text{loo}}[f_D] = \frac{1}{l}\sum_{i=1}^{l} L(f_{D^i},(\boldsymbol{x}_i,y_i)).$$

Заметим, что в последнем случае образец $(\boldsymbol{x}_i, y_i)$ не используется в процессе обучения, на нем тестируется решение, полученное на основе выборки D^i объема l–1.

В отличие от эмпирического риска loo-ошибка почти не смещена в следующем смысле: loo-ошибка, основанная на выборках объема l, дает несмещенную оценку ошибки обобщения после обучения по l–1 образцу.

Вычисление loo-ошибки требует больших временных затрат, поэтому многие исследователи пытались получить для нее простые (в отношении вычислений) границы.

ОЦЕНКА КАЧЕСТВА SVM-КЛАССИФИКАЦИИ

Одним из показателей качества алгоритма классификации является его способность правильно классифицировать новые образцы. При этом требуется, чтобы характеристика этой способности была эффективной и не требовала большого количества вычислений.

Для оценивания характеристики обобщения обычно используется процедура перекрестной проверки, в частности loo-оценка. Из тренировочной последовательности удаляется один образец. На оставшихся (l–1)-парах производится обучение, результат которого проверяется на удаленном образце. Число полученных ошибок, деленное на число членов тренировочной последовательности, и является в данном случае loo-оценкой ошибки обобщения.

Верхние границы для loo-ошибки классификации, предполагающие построение только одной машины опорных векторов на основе исходной тренировочной последовательности объема l, приведены в работах [3,4].

В работе [4] предложена верхняя граница leave-one-out-оценки ошибки обобщения для стандартного SV-классификатора:

$$J\_err^l = \frac{d}{l}, \text{ где } d = |\{i:(\rho\alpha_i R_\Delta^2 + \xi_i) \geq 1\}|. \quad (1)$$

Здесь: ρ – параметр, равный двум; α_i – множитель Лагранжа; ξ_i – ослабляющая переменная, соответствующая i-му образцу; R_Δ^2 – верхняя граница разности $K(\boldsymbol{x},\boldsymbol{x})$–$K(\boldsymbol{x},\boldsymbol{x}')$ для всех допустимых входных векторов $\boldsymbol{x}$, $\boldsymbol{x}'$, где K – ядерная функция (в случае линейного и гауссовского ядра $R_\Delta^2 = 1$). Доказано, что в среднем оценка (1) больше, чем истинная доля ошибок классификации [4].

В работе [3] показано, что для задачи бинарной классификации с использованием функции потерь Хевисайда $\delta(-y\cdot f(\boldsymbol{x}))$ loo-оценка ядерной машины, которая минимизирует регуляризованный риск на основе тренировочной последовательности D, ограничена сверху:

$$R_{loo}[f_D] \equiv \frac{1}{l}\sum_{i=1}^{l} \delta(-y_i f_{D^i}(\boldsymbol{x}_i)) \le$$

$$\le \frac{1}{l}\sum_{i=1}^{l} \delta(|\alpha_i| K(\boldsymbol{x}_i, x_i) - y_i f_D(\boldsymbol{x}_i)).$$

ROC-анализ (Receiver Operator Characteristic) представляет собой графический метод для представления результатов бинарной классификации при машинном обучении, отражающий качество данного классификатора или сравнительную эффективность нескольких классификаторов. ROC-кривые весьма полезны в случае асимметричного распределения входных векторов и неодинаковой цены ошибок классификации. Эти показатели становятся особенно важными при несбалансированных классах.

Чтобы определять качество классификации отдельно для каждого класса (один класс – класс с положительными образцами, второй – с отрицательными образцами), используют показатели чувствительности и специфичности или показатели recall и precision, причем recall – то же самое, что чувствительность.

Чувствительность отражает эффективность работы бинарного классификатора и определяет долю истинно положительных наблюдений относительно количества фактически положительных. Классификатор, обладающий высокой чувствительностью, обеспечивает большую вероятность правильного распознавания положительных образцов.

Специфичность отражает точность работы бинарного классификатора и определяется как отношение истинно отрицательных наблюдений к числу фактически отрицательных. Модель, обладающая высокой специфичностью, обеспечивает большую вероятность правильного распознавания отрицательных образцов.

Показатель precision определяет долю истинно положительных наблюдений относительно количества наблюдений, классифицированных моделью как положительные.

Например, при классификации пациентов на больных (положительные образцы) и здоровых (отрицательные образцы) чувствительный классификатор максимально предотвращает пропуск больных, а специфичный классификатор диагностирует только настоящих больных.

Практическое руководство по корректному использованию ROC-кривых и основных показателей качества классификации можно найти в работе [5]. Взаимосвязанность ROC-кривых и precision-recall-кривых рассмотрены в работе [6].

ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ БИНАРНОЙ КЛАССИФИКАЦИИ

Рассмотрим формулировку двойственной задачи SVM-обучения для бинарной классификации:

$$\text{найти } \min_{\alpha} W(\boldsymbol{\alpha}) = -\sum_{i=1}^{l}\alpha_i + \frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) \quad (2)$$

$$\text{при ограничениях } 0 \le \alpha_i \le C, \sum_{i=1}^{l}\alpha_i y_i = 0.$$

Здесь $(\boldsymbol{x}_1,y_1), \ldots, (\boldsymbol{x}_l, y_l) \in R^n \times \{-1;+1\}$ – тренировочная последовательность объема l; $\alpha_i \ge 0$ – множители Лагранжа; $C > 0$ – параметр регуляризации, контролирующий ширину полосы; $K(\boldsymbol{x},\boldsymbol{x}')$ – ядерная функция, удовлетворяющая условию Мерсера [2].

Оптимальная решающая функция представляет собой линейную комбинацию значений ядра на тренировочных векторах:

$$f(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}_i,\boldsymbol{x}) + b.$$

Пусть $\boldsymbol{Q}$ – симметричная положительно определенная $l \times l$ матрица с элементами $Q_{ij} = y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)$. Процедура декомпозиции [7] позволяет разбить задачу квадратичного программирования (2) на ряд задач существенно меньшей размерности q ($q << l$) и таким образом избавиться от проблем, связанных с вычислением и хранением в полном объеме матрицы $\boldsymbol{Q}$.

АЛГОРИТМ SVMLight

Процедура декомпозиции эффективно реализована в алгоритме SVMLight, предложенном в работе [8]. На каждом шаге итерационного процесса обучения оптимизируется только часть множителей Лагранжа (рабочее множество) при фиксированных значениях остальных. Элементами рабочего множества являются «наихудшие» нарушители условий Каруша–Куна–Таккера, для выбора которых применяется эвристическая процедура. Объем рабочего множества не меняется от итерации к итерации. Когда не остается двойственных переменных, подходящих для рабочего множества, исходная задача квадратичного программирования оказывается решенной.

Условия оптимальности. Обозначим множитель Лагранжа для ограничения типа равенства задачи (2) через λ^{eq}, а векторы множителей Лагранжа для нижней и верхней границ компонент вектора α – λ^{lo} и λ^{up} соответственно. Лагранжиан для двойственной задачи (2) примет вид:

$$L_W = -\sum_i \alpha_i + \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j Q_{ij} + \lambda^{eq}\sum_i \alpha_i y_i - \\ - \sum_i \lambda_i^{lo}\alpha_i + \sum_i \lambda_i^{up}(\alpha_i - C).$$

Выпишем условия оптимальности для этой задачи:

$$\forall i \in \{1, \ldots l\}: g_i(\boldsymbol{\alpha}) + (\lambda^{eq} y_i - \lambda_i^{lo} + \lambda_i^{up}) = 0,$$

$$\lambda_i^{lo}(-\alpha_i) = 0,\ \lambda_i^{lo}(\alpha_i - C) = 0 \qquad (3)$$

и

$$\boldsymbol{\lambda}^{lo} \geq 0,\ \boldsymbol{\lambda}^{up} \geq 0,\ \boldsymbol{\alpha}^{T} y = 0,\ 0 \leq \boldsymbol{\alpha} \leq C\cdot\mathbf{1}. \qquad (4)$$

Здесь $g_i(\boldsymbol{\alpha})$ – i-й элемент градиента целевой функции $W(\alpha)$ по α:

$$\boldsymbol{g}(\boldsymbol{\alpha}) = -\mathbf{1} + \boldsymbol{Q}\boldsymbol{\alpha}. \qquad (5)$$

Декомпозиция задачи (2). Сходимость процесса декомпозиции гарантирована, если рабочее множество удовлетворяет некоторым минимальным требованиям [8]. Разделим переменные α_i исходной задачи (2) на две категории: свободные переменные, индексы которых составляют множество B, и фиксированные переменные, индексы которых составляют множество N,$N = \{1,2,\ldots,l\}\backslash B$. Проведем декомпозицию задачи (2) посредством разделения α, $\boldsymbol{y}$ и $\boldsymbol{Q}$ на части, соответствующие множествам индексов B и N:

$$\boldsymbol{\alpha} = \begin{vmatrix}\boldsymbol{\alpha}_B \\ \boldsymbol{\alpha}_N\end{vmatrix},\ \boldsymbol{y} = \begin{vmatrix}\boldsymbol{y}_B \\ \boldsymbol{y}_N\end{vmatrix},\ \boldsymbol{Q} = \begin{vmatrix}\boldsymbol{Q}_{BB}\ \boldsymbol{Q}_{BN} \\ \boldsymbol{Q}_{NB}\ \boldsymbol{Q}_{NN}\end{vmatrix}. \qquad (6)$$

Учитывая симметричность матрицы $\boldsymbol{Q}$ и исключая из целевой функции постоянные слагаемые, приходим к задаче оптимизации:

$$\text{найти } \min_{\alpha_B} W(\boldsymbol{\alpha}_B) = \qquad (7)$$

$$= -\boldsymbol{\alpha}_B^T(\mathbf{1} - \boldsymbol{Q}_{BN}\boldsymbol{\alpha}_N) + \frac{1}{2}\boldsymbol{\alpha}_B^T\boldsymbol{Q}_{BB}\boldsymbol{\alpha}_B$$

$$\text{при ограничениях } \boldsymbol{\alpha}_B^T y_B + \boldsymbol{\alpha}_N^T y_N = 0, \qquad (8)$$

$$0 \leq \boldsymbol{\alpha}_B \leq C\cdot\mathbf{1}. \qquad (9)$$

Выбор рабочего множества. Эффективность от использования декомпозиции напрямую связана с процедурой выбора рабочего множества. Построение рабочего множества основано на методе Зойтендейка: определяется направление наискорейшего спуска, $\boldsymbol{d}$, которое имеет только q ненулевых элементов. Переменные, соответствующие этим элементам, и составляют рабочее множество. Для его выбора на текущей итерации t требуется решить следующую задачу:

$$\text{найти } \min_d V(\boldsymbol{d}) = \boldsymbol{g}(\boldsymbol{\alpha}^{(t)})^T\boldsymbol{d}$$

$$\text{при ограничениях } \boldsymbol{y}^T\boldsymbol{d} = 0,\ -\mathbf{1} \leq \boldsymbol{d} \leq \mathbf{1},$$

$$|\{d_i : d_i \neq 0\}| = q,$$

$$d_i \geq 0,\ i:\alpha_i = 0, \qquad (10а)$$

$$d_i \leq 0,\ i:\alpha_i = C. \qquad (10б)$$

Определим вектор ω: $\omega_i = y_i g_i(\alpha^{(t)})$ и проведем сортировку элементов α_i в порядке, соответствующем убыванию ω_i. Пусть q – четно. Выберем $q/2$ элементов из начала упорядоченного списка, для которых либо $0 < \alpha_i^{(t)} < C$, либо $d_i = -y_i$ удовлетворяет (10а) или (10б). Подобным же образом выберем $q/2$ элементов, начиная с конца списка, для которых либо $0 < \alpha_i^{(t)} < C$, либо $d_i = y_i$ удовлетворяет (10а) или (10б). Эти q переменных составляют рабочее множество.

Структура алгоритма SVMLight. Задают размер рабочего множества q, $q << l$.

Пока нарушены условия оптимальности:

– выбирают q переменных для рабочего множества; остальные l–q переменных фиксируют;

– проводят декомпозицию задачи (2) и решают подзадачу квадратичного программирования (7) при ограничениях (8), (9) размерности q.

Иначе процесс останавливают, так как оптимальное решение найдено.

Эффективная реализация алгоритма. Условия оптимальности (3), (4) удобно проверять, используя следующие соотношения:

$$\lambda^{\text{eq}} - \varepsilon \le y_i - \sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) \le \lambda^{\text{eq}} + \varepsilon,\ i\colon 0<\alpha_i<C,$$
$$y_i\left(\sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) + \lambda^{\text{eq}}\right) \ge 1-\varepsilon,\ i\colon \alpha_i = 0,$$
$$y_i\left(\sum_{j=1}^{l}\alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) + \lambda^{\text{eq}}\right) \le 1+\varepsilon,\ i\colon \alpha_i = C, \quad (11)$$
$$\sum_{j=1}^{l}\alpha_i y_i = 0,\ \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{y} = 0,$$

которые при ε = 0 эквивалентны (3)–(4). При решении большинства задач приемлемым является значение ε = 1e–3.

Определим на итерации t вектор $\boldsymbol{s}^{(t)}$:

$$s_i^{(t)} = \sum_{j=1}^{l}\alpha_j^{(t)} y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j),\ i\in\{1,2,\ldots l\}.$$

Тогда при изменении на каждой итерации t значений вектора $\boldsymbol{\alpha}$ от $\boldsymbol{\alpha}^{(t-1)}$ к $\boldsymbol{\alpha}^{(t)}$ по следующей формуле можно быстро обновлять суммы $s_i^{(t)}$, входящие в условия (11):

$$s_i^{(t)} = s_i^{(t-1)} + \sum_{j\in B}(\alpha_j^{(t)} - \alpha_j^{(t-1)}) y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j),$$

а следовательно, и значение градиента $\boldsymbol{g}(\boldsymbol{\alpha}^{(t)})$ (см. (5)).

В процессе итераций предусмотрена возможность уменьшения размерности («сжатия») исходной задачи (2) за счет удаления переменных α_i, которые, вероятно, окажутся равными 0 или C. Значения этих переменных фиксируются, и ни градиент $\boldsymbol{g}(\boldsymbol{\alpha}^{(t)})$ (5), ни условия оптимальности для них не пересчитываются, что приводит к существенному сокращению количества вычислений значений ядра. Детальное описание процедуры «сжатия» приведено в работе [8].

АЛГОРИТМ SEQUENTIAL MINIMAL OPTIMIZATION (SMO)

Алгоритм SMO, впервые предложенный для задач классификации в работах [9,10], реализует крайнюю форму декомпозиции задачи (2) при наименьшем возможном объеме рабочего множества, которое равно двум, т.к. вектор двойственных переменных $\boldsymbol{\alpha}$ должен подчиняться линейному ограничению типа равенств. Важнейшее преимущество SMO перед другими SVM-алгоритмами состоит в том, что решение задачи для двух переменных находится аналитически.

Вычислительные проблемы, которые могут возникать при реализации SMO, полностью связаны с ядерной матрицей $\{K(\boldsymbol{x}_i,\boldsymbol{x}_j)\}$. SMO особенно эффективен для случая разреженных данных (более 80% нулевых значений компонент входных векторов $\boldsymbol{x}$) и особенно быстро сходится для линейных SV-машин.

Здесь рассмотрена более эффективная по сравнению с первоначальным вариантом вторая модификация SMO-алгоритма Platt'a [9], предложенная в работе [11]. Исследование влияния методов выбора рабочего множества на эффективность алгоритма SMO-классификации проведено в работе [12]. Сходимость SMO-алгоритма для задач классификации доказана [13].

На каждом шаге SMO выбирает определенным образом двух нарушителей условий оптимальности $\{\alpha_i, \alpha_j\}$, т.е. рабочее множество, находит оптимальные значения этих переменных и соответственно обновляет SV-машину. Когда не остается подходящих для рабочего множества переменных, исходная задача (2) оказывается решенной.

Условия оптимальности и условия остановки SMO-алгоритма. Для получения подходящих условий остановки SMO-алгоритма, решающего двойственную задачу (2), выпишем условия оптимальности для этой задачи. Лагранжиан для двойственной задачи имеет вид:

$$L_W = \frac{1}{2}w(\boldsymbol{\alpha})w(\boldsymbol{\alpha}) - \sum_{i=1}^{l}\alpha_i - \sum_{i=1}^{l}\delta_i\alpha_i + \sum_{i=1}^{l}\mu_i(\alpha_i - C) - \beta\sum_{i=1}^{l}\alpha_i y_i,$$

где $w(\boldsymbol{\alpha}) = \sum_{i=1}^{l}\alpha_i y_i \boldsymbol{x}_i$.

Определим

$$F_i = w(\boldsymbol{\alpha})\boldsymbol{x}_i - y_i = \sum_j \alpha_j y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j) - y_i. \quad (12)$$

Тогда условия Каруша–Куна–Таккера для двойственной задачи для любого i примут вид:

$$\frac{\partial L_W}{\partial \alpha_i} = (F_i - \beta)y_i - \delta_i + \mu_i = 0,$$
$$\delta_i \ge 0,\ \delta_i\alpha_i = 0,\ \mu_i \ge 0,\ \mu_i(\alpha_i - C) = 0.$$

Рассмотрим три возможных положения переменной α_i. Для каждого случая условия Каруша–Куна–Таккера приобретают простой вид:

$$1).\ \alpha_i = 0, \text{тогда } \delta_i \geq 0, \mu_i = 0 \Leftrightarrow (F_i - \beta) y_i \geq 0,$$

$$2).\ 0 < \alpha_i < C, \text{тогда } \delta_i = 0, \mu_i = 0 \Leftrightarrow (F_i - \beta) y_i = 0, \quad (13)$$

$$3).\ \alpha_i = C, \text{тогда } \delta_i = 0, \mu_i \geq 0 \Leftrightarrow (F_i - \beta) y_i \leq 0.$$

Определим следующие множества индексов при фиксированном векторе $\boldsymbol{\alpha}$:

$$I_0 = \{i : 0 < \alpha_i < C\};$$
$$I_1 = \{i : y_i = 1, \alpha_i = 0\};$$
$$I_2 = \{i : y_i = -1, \alpha_i = C\};$$
$$I_3 = \{i : y_i = 1, \alpha_i = C\};$$
$$I_4 = \{i : y_i = -1, \alpha_i = 0\}.$$

Введем величины

$$b_{\text{up}} = \min\{F_i : i \in I_0 \cup I_1 \cup I_2\}, \quad (14)$$

$$b_{\text{low}} = \max\{F_i : i \in I_0 \cup I_3 \cup I_4\}. \quad (15)$$

Тогда условия оптимальности будут выполнены для такого значения вектора $\boldsymbol{\alpha}$, при котором

$$b_{\text{low}} \leq b_{\text{up}}. \quad (16)$$

Введем положительный параметр толерантности τ и запишем в соответствии с (13) приближенные условия оптимальности, которые и будем использовать в качестве правила остановки SMO-алгоритма.

$$\begin{aligned} &(F_i - \beta) y_i \geq -\tau, \text{ если } \alpha_i = 0 \\ &|F_i - \beta| \leq \tau, \text{ если } 0 < \alpha_i < C \\ &(F_i - \beta) y_i \leq \tau, \text{ если } \alpha_i = C. \end{aligned} \quad (17)$$

Говорят, что пара индексов $\{i, j\}$ определяет нарушение при заданном $\boldsymbol{\alpha}$, если выполнено одно из условий:

$$i \in I_0 \cup I_3 \cup I_4 \text{ и } j \in I_0 \cup I_1 \cup I_2 \text{ и } F_i > F_j \quad (18а)$$

$$i \in I_0 \cup I_1 \cup I_2 \text{ и } j \in I_0 \cup I_3 \cup I_4 \text{ и } F_i < F_j. \quad (18б)$$

Заметим, что условия оптимальности будут выполнены для $\boldsymbol{\alpha}$ тогда и только тогда, когда не существует ни одной пары индексов $\{i, j\}$, которая определяет нарушение при этом $\boldsymbol{\alpha}$.

При практических вычислениях используем параметр толерантности τ. Условие (16) примет вид

$$b_{\text{low}} \leq b_{up} + 2\tau. \quad (19)$$

А для определения нарушения заменим соответственно условия (18а) и (18б) на

$$i \in I_0 \cup I_3 \cup I_4 \text{ и } j \in I_0 \cup I_1 \cup I_2 \text{ и } F_i > F_j + 2\tau, \quad (20а)$$

$$i \in I_0 \cup I_1 \cup I_2 \text{ и } j \in I_0 \cup I_3 \cup I_4 \text{ и } F_i < F_j - 2\tau. \quad (20б)$$

SMO-алгоритм использует условие (19) (или (20)) для проверки оптимальности текущего вектора $\boldsymbol{\alpha}$. Заметим, что при этом не требуется знать текущее значение b. Кроме того, благодаря такому способу проверки условий Каруша–Куна–Таккера при заданном первом члене рабочего множества второй его член находится автоматически.

Выбор рабочего множества. На каждом шаге SMO-алгоритм выбирает две переменные двойственной задачи (2) для совместной оптимизации:

$$\{\alpha_{i_1}, \alpha_{i_2}\}.$$

Начальное приближение искомого вектора $\boldsymbol{\alpha}$ обычно представляет собой нулевой вектор. Начальное значение $b_{\text{up}} = -1$, $b_{\text{low}} = 1$; i_up полагают равным любому индексу из первого класса, i_low – любому индексу из второго класса. Сначала перебираются все члены тренировочной последовательности для выявления нарушителей условий оптимальности. При выполнении цикла, включающего индексы только из множества I_0, SMO-алгоритм всегда работает с наихудшей нарушающей парой, т.е. выбирает множество $B = \{i_1, i_2\} \subset \{1,2,\ldots,l\}$ следующим образом: $i_2 = i\_\text{low}$, а $i_1 = i\_\text{up}$, где индексы i_low и i_up находятся из условий: $F_{i\_\text{low}} = b_{\text{low}}$, $F_{i\_\text{up}} = b_{\text{up}}$.

Таким образом, выбор первого и второго члена рабочего множества происходит одновременно (в отличие от первоначального варианта SMO [10], где выбор второго члена рабочего множества производится на основе известной эвристической процедуры в предположении, что первый член уже выбран).

После удачного шага алгоритма с использованием пары индексов $\{i_1, i_2\}$ образуем множество $I^* = I_0 \cup \{i_1, i_2\}$. Заметим, что оба множества $I^* \cap \{I_0 \cup I_1 \cup I_2\}$ и $I^* \cap \{I_0 \cup I_3 \cup I_4\}$ не пусты. Следовательно, можно частично вычислить b_{low} и b_{up}, используя множество $I^* = I_0 \cup \{i_1, i_2\}$. Нам пришлось отказаться от этой привлекательной рекомендации, которая приводила к «недообученной» машине, т.е. оптимальная гиперплоскость строилась правильно, но полоса была слишком широкой. В нашей реализации SMO-алгоритма не используется идея, связанная с множеством $I^* = I_0 \cup \{i_1, i_2\}$,

b_{low} и b_{up} на каждой итерации определяются согласно (14) и (15) соответственно.

Когда устанавливается оптимальность (согласно (19)) на множестве I_0, SMO-алгоритм возвращается к рассмотрению всей (исходной) тренировочной последовательности для проверки оптимальности на всех индексах. Поскольку (b_{low}, i_low) и (b_{up}, i_up) были вычислены только по множеству I_0, для каждого текущего индекса эти величины обновляются: сначала вычисляется F_i, а затем проверяется оптимальность с использованием текущих (b_{low}, i_low) (согласно (20)). Если в этом цикле по i нет нарушений, это означает, что условия оптимальности выполнены для всех элементов вектора α, т.е. оптимальное решение найдено.

Решение задачи квадратичного программирования для выбранного рабочего множества. Рабочее множество состоит из двух переменных $\{\alpha_i, \alpha_j\}$. Ограничение типа равенств (см. (8)) можно использовать для исключения одной из них. Для удобства изложения все величины, имеющие отношение к первой переменной рабочего множества, будут иметь индекс 1, ко второй – индекс 2.

Тогда, если соответствующие метки классов различны ($y_1 \neq y_2$), то $\alpha_1 - \alpha_2 = \text{const}$. Если соответствующие метки одинаковы ($y_1 = y_2$), то $\alpha_1 + \alpha_2 = \text{const}$. Таким образом, на каждом шаге алгоритма получаем задачу квадратичного программирования только для одной переменной (пусть α_2). Эта задача имеет аналитическое решение (подробный вывод см. в Appendix работы [10]):

$$\alpha_2^* = \alpha_2 + \frac{y_2(F_2 - F_1)}{\eta},$$

где $\eta = K(\boldsymbol{x}_1,\boldsymbol{x}_1) + K(\boldsymbol{x}_2,\boldsymbol{x}_2) - 2K(\boldsymbol{x}_1,\boldsymbol{x}_2)$ – вторая производная целевой функции (7). С учетом ограничений типа неравенств и в предположении, что $\eta > 0$, новое значение переменной α_2 будет иметь вид:

$$\alpha_2^{\text{new}} = \begin{cases} H, \text{ если } \alpha_2^* \geq H \\ \alpha_2^*, \text{ если } L < \alpha_2^* < H, \\ L, \text{ если } \alpha_2^* \leq L \end{cases}$$

где L и H – границы для α_2: при $y_1 \neq y_2$ $L = \max(0, \alpha_1 - \alpha_2)$, $H = \min(C, C + \alpha_1 - \alpha_2)$; при $y_1 = y_2$ $L = \max(0, \alpha_2 + \alpha_1 - C)$, $H = \min(C, \alpha_1 + \alpha_2)$.

Если в тренировочной последовательности имеются хотя бы два одинаковых вектора $\boldsymbol{x}$, то η может стать равной нулю. SMO-алгоритм будет работать и в этом случае, вычисляя W_L и W_H – значения целевой функции (см. (7)) на концах сегмента прямой, которую определяет линейная зависимость между α_1 и α_2:

$$f_1 = y_1F_1 - \alpha_1K(\boldsymbol{x}_1,\boldsymbol{x}_1) - s\alpha_2K(\boldsymbol{x}_1,\boldsymbol{x}_2),$$
$$f_2 = y_2F_2 - s\alpha_1K(\boldsymbol{x}_1,\boldsymbol{x}_2) - \alpha_2K(\boldsymbol{x}_2,\boldsymbol{x}_2),$$
$$L_1 = \alpha_1 + S(\alpha_2 - L),$$
$$H_1 = \alpha_1 + S(\alpha_2 - H),$$
$$W_L = L_1f_1 + Lf_2 + \frac{1}{2}L_1^2K(\boldsymbol{x}_1,\boldsymbol{x}_1) +$$
$$+ \frac{1}{2}L^2K(\boldsymbol{x}_2,\boldsymbol{x}_2) + sLL_1K(\boldsymbol{x}_1,\boldsymbol{x}_2),$$
$$W_H = H_1f_1 + Hf_2 + \frac{1}{2}H_1^2K(\boldsymbol{x}_1,\boldsymbol{x}_1) +$$
$$+ \frac{1}{2}H^2K(\boldsymbol{x}_2,\boldsymbol{x}_2) + sHH_1K(\boldsymbol{x}_1,\boldsymbol{x}_2),$$

где $s = y_1y_2$.

Если $W_L < W_H - \tau$, то α_2^{new} получает значение L. Если $W_L > W_H + \tau$, то α_2^{new} получает значение H. Если значения целевой функции W_L и W_H отличаются друг от друга меньше, чем на τ, то текущее значение α_2 не меняется, и выбирается новое рабочее множество. Значение переменной α_1^{new} вычисляем, опираясь на полученное в процессе оптимизации значение переменной α_2^{new}: $\alpha_1^{\text{new}} = \alpha_1 + s(\alpha_2 - \alpha_2^{\text{new}})$.

Краткое описание SMO-алгоритма. SMO-алгоритм представляет собой процесс, каждая итерация которого состоит из двух шагов: 1) выбор рабочего множества объема два $\{\alpha_i,\alpha_j\}$ (эвристический метод); 2) решение задачи квадратичного программирования для выбранного рабочего множества (аналитический метод). Когда не остается двух подходящих для рабочего множества переменных, исходная задача (2) оказывается решенной.

Основные этапы SMO-алгоритма:

1. Возьмем в качестве начального приближения искомого вектора $\boldsymbol{\alpha}$ некоторое $\boldsymbol{\alpha}^1$ (обычно берут нулевой вектор). Положим $k = 1$.

2. Если $\boldsymbol{\alpha}^k$ удовлетворяет условиям оптимальности (19), то процесс решения закончен. В противном случае находим рабочее множество из двух элементов, индексы которых образуют множество $B = \{i,j\} \subset \{1,2,\ldots,l\}$, где l – объем тренировочной последовательности. Определим множество индексов $N = \{1,2,\ldots,l\}\backslash B$ и векторы $\boldsymbol{\alpha}_B^k$ и $\boldsymbol{\alpha}_N^k$ как соответствующие B и N части вектора α. Множества B и N меняются от итерации к итерации, т.е. зависят от k.

3. Решаем задачу квадратичного программирования для текущих $\{\alpha_i,\alpha_j\}$ (см. (7)–(9)). Положим $\boldsymbol{\alpha}_B{}^{k+1}$ равным оптимальному решению этой задачи.

4. Положим: $\boldsymbol{\alpha}_N{}^{k+1} \equiv \boldsymbol{\alpha}_N{}^{k}$; $k \leftarrow k+1$ и перейдем к п. 2.

АЛГОРИТМ SUCCESSIVE OVERRELAXATION (SOR)

Если добавить к целевой функции стандартной SVM-задачи классификации переменную $b^2/2$ и минимизировать полученную функцию по $\boldsymbol{w}$, ξ и b, то двойственная задача минимизации лагранжиана, будучи по-прежнему задачей квадратичного программирования, не будет иметь ограничений типа равенств:

$$\text{найти } \min_{\alpha} \frac{1}{2}\boldsymbol{\alpha}^T(\boldsymbol{Q}+\boldsymbol{y}\boldsymbol{y}^T)\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}, \qquad (21)$$

$$\text{при ограничениях } 0 \le \boldsymbol{\alpha} \le C\boldsymbol{e},$$

где $\boldsymbol{Q}$ – симметричная положительно определенная матрица $l \times l$ с элементами $Q_{ij} = y_i y_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)$;

$\boldsymbol{y}$ – вектор-столбец, элементами которого являются метки классов $y_i = \pm 1$;

$\boldsymbol{e}$ – единичный вектор из e компонент;

$\boldsymbol{\alpha}$ – вектор множителей Лагранжа из e компонент.

Решение задачи (21) эквивалентно решению системы линейных алгебраических уравнений (СЛАУ)

$$\boldsymbol{M}\boldsymbol{\alpha} = \boldsymbol{e}, \qquad (22)$$

где $\boldsymbol{M} = \boldsymbol{Q} + \boldsymbol{y}\boldsymbol{y}^T$ – симметричная положительно определенная матрица.

Алгоритм SOR [14,15] решает СЛАУ (22) методом последовательной верхней релаксации, учитывая на каждом шаге ограничения типа неравенств задачи (21). На $(k+1)$-й итерации компоненты искомого вектора $\boldsymbol{\alpha}^{k+1}$ вычисляются следующим образом:

$$\alpha_i^{(k+1)} = \qquad (23)$$

$$= \left(\omega \left(e_i - \sum_{j=1}^{i-1} m_{ij}\alpha_j^{(k+1)} - \sum_{j=i+1}^{l} m_{ij}\alpha_j^{(k)} \right) / m_{ii} + (1-\omega)\alpha_i^{(k)} \right), \quad i = \overline{1,l},$$

где $()_{\#}$ обозначает:

$$(\alpha_i)_{\#} = \begin{cases} 0, \text{ если } \alpha_i \le 0 \\ \alpha_i, \text{ если } 0 \le \alpha_i \le C. \\ C, \text{ если } \alpha_i \ge C \end{cases}$$

Параметр релаксации $\omega \in (0,2)$. При $\omega = 1$ итерации (23) являются итерациями Гаусса–Зейделя. Варьируя значения ω, можно добиться увеличения скорости сходимости алгоритма. Итерационный процесс сходится при любом начальном приближении α^0.

Алгоритм SOR состоит из следующих шагов:

1. Выберем $\omega \in (0,2)$. Можно начать с $\omega = 1$, а в случае медленной сходимости алгоритма подобрать другое значение ω.

2. Выберем начальное приближение $\boldsymbol{\alpha}^0 \in R^l$ (обычно нулевой вектор). Положим $k = 0$.

3. Зная $\boldsymbol{\alpha}^k$, вычислим $\boldsymbol{\alpha}^{k+1}$ по формуле (23). Вычисления проводим, пока $\|\boldsymbol{\alpha}^{k+1} - \boldsymbol{\alpha}^k\| > \tau$, где τ – заданный уровень толерантности (обычно $\tau = 10^{-3}$).

АЛГОРИТМ INCREMENTAL UPDATING (IU)

Алгоритм IU [16,17] предполагает online режим поступления образцов. За конечное число шагов алгоритм приводит к выполнению условий ККТ как для всех предшествующих данных, так и для вновь поступившего образца. Это достигается путем изменения ключевых переменных системы за счет наибольшего возможного приращения двойственной переменной, соответствующей новому образцу.

Условия оптимальности. Если ограничение типа равенств непосредственно включить в целевую функцию двойственной задачи (2), добавив к $W(\boldsymbol{\alpha})$ слагаемое

$$b\sum_{i=1}^{l}\alpha_i y_i,$$

то условия оптимальности примут вид:

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_{j=1}^{l} Q_{ij}\alpha_j + y_i b - 1 = \qquad (24)$$

$$= y_i f(\boldsymbol{x}_i) - 1 \begin{cases} \ge 0; \ \alpha_i = 0, \\ = 0; \ 0 < \alpha_i < C, \ i = 1,\ldots,l,, \\ \le 0; \ \alpha_i = C \end{cases}$$

$$\frac{\partial W}{\partial b} = \sum_{j=1}^{l} y_j \alpha_j = 0. \qquad (25)$$

До поступления нового образца условия Каруша–Куна–Таккера (24), (25) выполнены для всех образцов тренировочной последовательности, т.е. имеется SV-машина, построенная по l образцам. Согласно условиям ККТ, векторы тренировочной последовательности можно разделить на три категории:

$S = \{\boldsymbol{x}_i: 0 < \alpha_i < C\}$ – множество опорных векторов, лежащих на оптимальных поддерживающих границах, для них $g_i = 0$;

$E = \{\boldsymbol{x}_i: \alpha_i = C\}$ – множество опорных векторов, нарушающих эти границы, для них $g_i \le 0$;

$O = \{\boldsymbol{x}_i: \alpha_i = 0\}$ – множество векторов, не являющихся опорными, для них $g_i \ge 0$.

Введем также множество векторов $R = E \cup O$.

Соответствующие этим множествам векторов множества индексов (будем обозначать их строчными буквами s,e,o,r) индуцируют некоторое разбиение матрицы $\boldsymbol{Q}$, вектора меток $\boldsymbol{y}$, вектора коэффициентов $\boldsymbol{\alpha}$ и градиента целевой функции $\boldsymbol{g}$. Будем использовать для таких разбиений нижние индексы.

Соотношения чувствительности. При поступлении нового вектора $\boldsymbol{x}_c$ соответствующий ему коэффициент α_c первоначально полагают равным нулю. Если при этом расширенный вектор коэффициентов (размерности $l+1$) не является оптимальным (это означает, что $\boldsymbol{x}_c$ должен стать опорным вектором), то все коэффициенты α_i и сдвиг b должны быть обновлены для получения оптимального решения, соответствующего увеличенной тренировочной последовательности (объема $l+1$).

Коэффициенты α_i, соответствующие векторам множества S, меняют свои значения в течение каждого шага IU-алгоритма таким образом, чтобы удерживать все элементы тренировочной последовательности в равновесии, т.е. поддерживать выполнение для них условий Каруша–Куна–Таккера.

Выписывая условия Каруша–Куна–Таккера до и после обновления $\boldsymbol{\alpha}$ в предположении, что состав множеств S и R не менялся, получим следующие соотношения, которые должны быть выполнены после обновления:

$$\begin{bmatrix} \Delta g_c \\ \Delta \boldsymbol{g}_s \\ \Delta \boldsymbol{g}_r \\ 0 \end{bmatrix} = \begin{bmatrix} y_c \boldsymbol{Q}_{cs} \\ \boldsymbol{y}_s \boldsymbol{Q}_{ss} \\ \boldsymbol{y}_r \boldsymbol{Q}_{rs} \\ 0 \;\; \boldsymbol{y}_s^T \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta \boldsymbol{\alpha} \end{bmatrix} + \Delta \alpha_c \begin{bmatrix} Q_{cc}^T \\ \boldsymbol{Q}_{cs}^T \\ \boldsymbol{Q}_{cr}^T \\ y_c \end{bmatrix} \qquad (26)$$

Из условий (24) следует, что $\Delta \boldsymbol{g}_s = 0$, поэтому 2-я и 4-я строки системы (26) приводят к системе линейных уравнений относительно $\Delta\alpha_j$ (по всем индексам из множества s) и Δb:

$$\Delta b = \beta_0 \Delta \alpha_c, \;\; \Delta \alpha_j = \beta_j \Delta \alpha_c. \qquad (27)$$

где β – вектор чувствительностей коэффициентов α_j, $j \in s$, и сдвига b к изменению α_c имеет структуру $[\beta_0 \;\; \boldsymbol{\beta}_s^T]$ и определяется следующим образом:

$$\beta = -\begin{bmatrix} 0 \;\; \boldsymbol{y}_s^T \\ \boldsymbol{y}_s \boldsymbol{Q}_{ss} \end{bmatrix}^{-1} \begin{bmatrix} y_c \\ \boldsymbol{Q}_{cs}^T \end{bmatrix}. \qquad (28)$$

Обозначим первый матричный сомножитель в (28) через $\Re$.

Подставив (27) в 1-ю и 3-ю строки системы (26), получим

$$\begin{bmatrix} \Delta g_c \\ \Delta \boldsymbol{g}_r \end{bmatrix} = \boldsymbol{\gamma} \Delta \alpha_c, \qquad (29)$$

где γ – вектор margin-чувствительностей, определяется следующим образом:

$$\gamma = \begin{bmatrix} y_c \boldsymbol{Q}_{cs} \\ \boldsymbol{y}_r \boldsymbol{Q}_{rs} \end{bmatrix} \boldsymbol{\beta} + \begin{bmatrix} Q_{cc} \\ \boldsymbol{Q}_{cr}^T \end{bmatrix}. \qquad (30)$$

Итак, обновление SV-машины, вызванное новым членом тренировочной последовательности, должно контролироваться соотношениями чувствительности (27) и (29).

Верхний предел величины $\Delta\alpha_c$. Поскольку состав множеств S и R меняется с изменением значений коэффициентов α_j и b, соотношения (26) не могут использоваться непосредственно для получения нового состояния SV-машины. IU-алгоритм определяет наибольшее возможное приращение $\Delta\alpha_c$, при котором некоторый вектор может перемещаться из своего множества в соседние. Чтобы учесть подобные изменения, рассмотрим следующие ситуации:

1. Некоторый коэффициент α_i, являющийся элементом вектора α_s, достигает границы (0 или C). Пусть $\varepsilon > 0$ – маленькое число.

Определим множества индексов

$$I_S = \{I_+^S \cup I_-^S\},\ I_+^S = \{i \in s{:}\beta_i > \varepsilon\},$$
$$I_-^S = \{i \in s{:}\beta_i < -\varepsilon\}$$

и положим

$$\Delta\alpha_i^{\max} = \begin{cases} C - \alpha_i,\ i \in I_+^S \\ -\alpha_i,\ i \in I_-^S. \end{cases}$$

Тогда наибольшее возможное приращение α_c, прежде чем некоторый вектор из S перейдет в R, таково:

$$\Delta\alpha_c^S = \min_{i \in I^S} \left| \frac{\Delta\alpha_i^{\max}}{\beta_i} \right| \operatorname{sign}\left(\frac{\Delta\alpha_p^{\max}}{\beta_p} \right),$$

где p – индекс минимального значения величины $|\Delta\alpha_i^{\max}/\beta_i|$. Если минимальное значение величины $|\Delta\alpha_i^{\max}/\ \beta_i|$ достигается на нескольких индексах, то индекс

$$p = \arg\min_{i \in I^S} \left| \frac{\Delta\alpha_i^{\max}}{\beta_i} \right|.$$

2. Некоторое g_i из множества R достигает нуля.

Тогда наибольшее возможное приращение α_c, прежде чем некоторый вектор из множества R перейдет в множество S, вычисляется следующим образом

$$\Delta\alpha_c^R = \min_{i \in I^R} \frac{-g_i}{\gamma_i},$$

где $I^R = \{I_+^R \cup I_-^R\}$, $I_+^R = \{i \in \text{e} : \gamma_i > \varepsilon\}$, $I_-^R = \{i \in \text{o} : \gamma_i < -\varepsilon\}$.

3. g_c становится нулем.

Если обновление возможно, т.е. $\gamma_c > \varepsilon$, то наибольшее возможное приращение α_c имеет вид

$$\Delta\alpha_c^g = \frac{-g_c}{\gamma_c}. \qquad (31)$$

4. α_c достигает верхней границы C.

Понятно, что в этом случае наибольшее возможное приращение α_c таково:

$$\Delta\alpha_c^\alpha = C - \alpha_c.$$

Наименьшее из перечисленных четырех значений

$$\Delta\alpha_c^{\max} = \min(\Delta\alpha_c^S, \Delta\alpha_c^R, \Delta\alpha_c^g, \Delta\alpha_c^a) \qquad (32)$$

и будет окончательным наибольшим возможным приращением α_c. Важно, на каком множестве индексов достигается минимум при определении наибольшего возможного приращения $\Delta\alpha_c$, а также само значение этого индекса.

Рекурсивное обновление матрицы $\Re$. После того, как наибольшее возможное приращение α_c определено, производится вычисление приращений ключевых переменных системы $\Delta\boldsymbol{\alpha}_s$, Δb, Δg согласно соотношениям чувствительности (27) и (29). Матрица $\Re$ тоже должна быть пересчитана для того, чтобы учесть новый состав множества S. Рассмотрим эффективное обновление этой матрицы.

1. Некоторый вектор $\boldsymbol{x}_k$ добавляется к множеству S.

Применение формулы Шермана–Моррисона_Вудбери для обращения блочной матрицы [18] приводит к соотношению

$$\Re \leftarrow \begin{bmatrix} \Re & \boldsymbol{\eta}_k \\ \boldsymbol{\eta}_k^T & Q_{kk} \end{bmatrix}^{-1} = \begin{bmatrix} \Re & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{Q_{kk} - \boldsymbol{\eta}_k^T \Re \boldsymbol{\eta}_k} \begin{bmatrix} \boldsymbol{\beta}_k \\ 1 \end{bmatrix} [\boldsymbol{\beta}_k^T 1],$$

где $\boldsymbol{\eta}_k = [y_k\ Q_{ks}{}^T]^T$, $\boldsymbol{\beta}_k = -\Re\eta_k$.

2. Некоторый вектор $\boldsymbol{x}_k$ удаляется из множества S.

Тогда матрица $\Re$ конструируется следующим образом: $\Re_{ij} \leftarrow \Re_{ij} - \Re_{kk}^{-1}\ \Re_{ik}\ \Re_{kj}$, $\forall \Re_{ij} \in \Re$; $i \in s \cup \{0\}$, $i,j \neq k$, где индекс 0 соответствует b.

Структура IU-алгоритма. Основной блок IU-алгоритма представляет собой процедуру для обновления уже существующего оптимального решения после добавления одного нового образца. Имеем SV-машину, построенную на основе тренировочной последовательности объема l. Добавим к имеющимся данным один образец $\{\boldsymbol{x}_c, y_c\}$. Используем имеющееся решение $\{\alpha_i^l, b^l\}$, матрицу $\Re$ и новый образец для получения нового оптимального решения $\{\alpha_i^{l+1}, b^{l+1}\}$, $i = 1,\ldots,l+1$, применяя IU-алгоритм.

Этапы IU-алгоритма ($l \leftarrow l+1$):

1. Положим $\alpha_c = 0$.

2. Если $g_c > 0$, то решение остается прежним. Конец процедуры.

3. Пока $g_c < 0$ и $\alpha_c < C$, выполняем:

вычисляем β и γ согласно формулам (28) и (30);

вычисляем $\Delta\alpha_c{}^{\max}$ согласно (32), пусть k – индекс, на котором достигается минимум в (32);

обновляем: $\alpha_c \leftarrow \alpha_c + \Delta\alpha_c{}^{\max}$;

$$\begin{bmatrix}\Delta b\\ \Delta\boldsymbol{\alpha}_s\end{bmatrix} \leftarrow \beta\Delta\alpha_c^{\max},\ \begin{bmatrix} b\\ \boldsymbol{\alpha}_s\end{bmatrix} \leftarrow \begin{bmatrix} b\\ \boldsymbol{\alpha}_s\end{bmatrix} + \begin{bmatrix}\Delta b\\ \Delta\boldsymbol{\alpha}_s\end{bmatrix};$$
$$\begin{bmatrix}\Delta g_c\\ \Delta\boldsymbol{g}_r\end{bmatrix} \leftarrow \gamma\Delta\alpha_c^{\max},\ \begin{bmatrix} g_c\\ \boldsymbol{g}_r\end{bmatrix} \leftarrow \begin{bmatrix} g_c\\ \boldsymbol{g}_r\end{bmatrix} + \begin{bmatrix}\Delta g_c\\ \Delta\boldsymbol{g}_r\end{bmatrix};$$

если $k \in s$, то {$\boldsymbol{x}_k$ переходит из S в R}, иначе, {если $k \in r$, то $\boldsymbol{x}_k$ переходит из R в S}, иначе {$k = c$: ничего не делаем, конец процедуры}. (Заметим, что при продолжении процесса поступления образцов выход на конец процедуры означает переход к п. 1. При этом, если $k = c$ и минимум в (32) дает величина (31), то {$\boldsymbol{x}_c$ добавляем к S и рекурсивно обновляем матрицу $\Re$}, иначе {$\boldsymbol{x}_c$ добавляем к E}. Далее $l \leftarrow l + 1$ и переходим к п. 1.);

рекурсивно обновляем матрицу $\Re$; возвращаемся к началу п.3.

NAIVE ONLINE RISK MINIMIZATION ALGORITHM (NORMA)

В работе [19] рассмотрено online обучение ядерных машин. Разработан простой и эффективный алгоритм NORMA для широкого круга проблем, таких как классификация, восстановление регрессии и классификация при отсутствии меток (novelty detection), на основе стохастического градиентного спуска в гильбертовом пространстве аппроксимирующих функций.

Цель любого алгоритма обучения состоит в получении машины $f : X \to R$ на основе тренировочной последовательности D объема l образцов $(\boldsymbol{x}_1, y_1),\ldots,(\boldsymbol{x}_l, y_l) \in X \times Y$. Пусть функция потерь $L : R \times Y \to R$, $L = L(f(\boldsymbol{x}),y)$ штрафует отклонение оценок $f(\boldsymbol{x})$ от наблюдаемых меток y. В случае классификации, когда $Y = \{-1,+1\}$, $\mathrm{sign}(f(\boldsymbol{x}))$ интерпретируется как предсказание класса $\boldsymbol{x}$, а $|f(\boldsymbol{x})|$ отражает уровень доверия к такой классификации.

Результат f работы алгоритма обучения называют гипотезой. Множество всех возможных гипотез обозначают через H. Пусть H представляет собой гильбертово пространство с репродуктивным ядром, т.е. в пространстве H существует ядерная функция $K : X \times X \to R$ и скалярное произведение $\langle\cdot,\cdot\rangle_H$, такие, что:

1) K обладает репродуктивным свойством:

$$\langle f,K(\boldsymbol{x},\cdot)\rangle_H = f(\boldsymbol{x}),\ x \in X,\ \forall f \in H; \qquad (33)$$

2) пространство H является замыканием линейной оболочки всех функций $K(\boldsymbol{x},\cdot)$, $\boldsymbol{x} \in X$.

Скалярное произведение в пространстве H вводится согласно (33):

$$K(\boldsymbol{x},\boldsymbol{x}') = \langle K(\boldsymbol{x},\cdot),K(\boldsymbol{x}',\cdot)\rangle_H$$

и порождает норму, которая является мерой сложности функций. Для любого элемента

$$f \in H: \|f\|_H = \langle f,f\rangle_H^{1/2}.$$

Различные классы функций могут быть обучены путем использования различных ядер. Функция, минимизирующая регуляризованный риск,

$$R_{\mathrm{reg}}[f,D] = R_{\mathrm{emp}}[f,D] + \lambda\|f\|_H^2$$

единственна и представляет собой ядерную машину, т.е. имеет вид:

$$f(\boldsymbol{x}) = \sum_{i=1}^{l}\beta_i K(\boldsymbol{x}_i,\boldsymbol{x}),$$

где вектор весов признаков $\beta \in R^l$. Положительный параметр регуляризации λ согласовывает малую эмпирическую ошибку со степенью гладкости (сложности) решающей функции. Параметр регуляризации C для стандартного SVM-обучения связан с λ соотношением $C = 1/(2l\lambda)$.

NORMA вырабатывает последовательность гипотез $f_1,\ldots, f_{l+1}$, где f_1 – некоторая произвольная гипотеза, а f_i для $i > 1$ – гипотеза, построенная после поступления $(i–1)$-го образца. $L(f_k(\boldsymbol{x}_k),y_k)$ – значение функции потерь алгоритма обучения при попытке предсказать y_k на основании $\boldsymbol{x}_k$ и предшествующих образцов $(\boldsymbol{x}_1,y_1),\ldots,(\boldsymbol{x}_{k-1},y_{k-1})$. Подходящей мерой точности алгоритма, использующего тренировочную последовательность D объема l, является накопленная функция потерь

$$L_{\mathrm{cum}}[f,D] = \sum_{k=1}^{l}L(f_k(\boldsymbol{x}_k),y_k).$$

Заметим, что здесь f_k тестируется на образце $(\boldsymbol{x}_k,\ y_k)$, который не участвует в обучении f_k. Таким образом, если можно гарантировать небольшое значение накопленной функции потерь, то тем самым можно уберечься от переобучения.

Поскольку речь идет об online-алгоритме, определим наряду с регуляризованным риском мгновенный регуляризованный риск (instantaneous regularized risk) для одного образца $(\boldsymbol{x}_k,\ y_k)$:

$$R_{\mathrm{inst}}[f,x,y] = R_{\mathrm{reg}}[f,(\boldsymbol{x},y)].$$

NORMA производит градиентный спуск в направлении минимума мгновенного риска. Общая форма правила обновления имеет вид

$$f_{k+1} = f_k - \eta_k \frac{\partial R_{\text{inst}}[f, \boldsymbol{x}_k, y_k]}{\partial f}\Big|_{f=f_k},$$

где $f_i \in H$, $\partial/\partial f$ – градиент по f, $\eta_k > 0$ – скорость обучения.

Воспользовавшись репродуктивным свойством ядра, получим

$$f_{k+1} = (1 - \eta_k\lambda)f_k - \eta_k L'(f_k(\boldsymbol{x}_k), y_k)K(\boldsymbol{x}_k, \cdot),$$

где $L'(z,y) = \partial L(z,y)/\partial z$.

Ясно, что для любого k должно выполняться: $\eta_k < 1/\lambda$.

Выберем нулевую начальную гипотезу $f_1 = 0$. Гипотезу f_k представим в виде ядерного разложения

$$f_k(\boldsymbol{x}) = \sum_{i=1}^{k-1} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}), \boldsymbol{x} \in X,$$

где коэффициенты на шаге k обновляются следующим образом:

$$\alpha_k = -\eta_k L'(f_k(\boldsymbol{x}_k) y_k), \quad i = k;$$
$$\alpha_i = (1 - \eta_k\lambda)\alpha_i, \quad i < k.$$

Добавим к решающей функции f сдвиг b, $b \in R$. На каждом шаге алгоритма обновляем b следующим образом:

$$b_{k+1} = b_k - \eta_k \frac{\partial R_{\text{inst}}[f + b, \boldsymbol{x}_k, y_k]}{\partial b}\Big|_{f+b=f_k+b_k}.$$

NORMA для задачи классификации. Для задачи классификации используется так называемая soft margin функция потерь $L_\rho(f(\boldsymbol{x}), y) = \max(0, \rho - yf(\boldsymbol{x}))$, $\rho \geq 0$ – margin-параметр. Функция потерь $L_\rho(f(\boldsymbol{x}), y)$ положительна, если f при $(\boldsymbol{x}, y)$ не превышает ширину полосы (margin) по крайней мере на величину ρ. В этом случае говорят, что f сделала margin-ошибку. Если f действительно сделала ошибку, то $L_\rho(f(\boldsymbol{x}), y) \geq \rho$.

Пусть σ_k – индикатор того, сделала ли f margin-ошибку при $(\boldsymbol{x}_k, y_k)$. Тогда

$$L'_\rho(f_k(\boldsymbol{x}_k), y_k) = -\sigma_k y_k = \begin{cases} 0, & y_k f_k(\boldsymbol{x}_k) > \rho \\ -y_k, & y_k f_k(\boldsymbol{x}_k) \leq \rho, \end{cases}$$

и правило обновления примет вид:

$$f_{k+1} = (1 - \eta_k\lambda)f_k + \eta_k\sigma_k y_k K(\boldsymbol{x}_k, \cdot),$$
$$b_{k+1} = b_k + \eta_k\sigma_k y_k.$$

Обновление в терминах коэффициентов ядерного разложения α_i, $i = 1,\ldots,k-1$, будет выглядеть так:

$$(\alpha_i, \alpha_k) \leftarrow ((1 - \eta_k\lambda)\alpha_i, \eta_k\sigma_k y_k).$$

Ключевым моментом алгоритмов рассматриваемого типа является выбор величины шага (скорости обучения) η_k. Обычно поступают следующим образом: либо назначают постоянный размер шага $\eta_k = \eta < 1/\lambda$, либо выбирают некоторое правило, согласно которому η_k убывает с ростом k. Метод автоматической настройки размера шага, так называемый Stochastic Meta-Descent-алгоритм, предложен в работе [20].

В работе [19] показано, что при достаточно мягких ограничениях на функцию потерь средний мгновенный риск

$$\frac{1}{l}\sum_{k=1}^{l} R_{\text{inst}}[f_k, \boldsymbol{x}_k, y_k]$$

гипотез, вырабатываемых NORM-алгоритмом, стремится к минимуму регуляризованного риска

$$\min_{f+b} R_{\text{reg}}[f + b, D]$$

со скоростью $O(l^{-1/2})$. Если при этом образцы тренировочной последовательности независимы и одинаково распределены, то с большой вероятностью регуляризованный риск усредненной гипотезы подобным же образом стремится к минимуму ожидаемого риска.

РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

Исследована сравнительная эффективность четырех алгоритмов решения задачи классификации: SVM_Light, SMO, SOR, IU. При построении SV-машин на реальных данных соблюдались общепринятые рекомендации:

масштабирование данных;

использование нелинейного (в данном случае гауссового) ядра;

подбор параметров регуляризации (C) и ядра (σ) по выборке небольшого объема с использованием процедуры поиска на грубой решетке с последующим тщательным уточнением на участке, соответствующем лучшему значению принятых оценок качества;

Таблица 1. Зависимость характеристик SMO-алгоритма от параметра C

C	CPU time	SV	BSV	(SV – BSV)/SV
1	2,203	161	135	0,161
10	4,547	82	38	0,293
100	10,343	53	5	0,906
250	**9,500**	**51**	**2**	**0,961**
500	13,391	49	0	1

Эффективность алгоритмов оценивалась по следующим характеристикам:

время, затраченное на построение SV-машины (CPU-time, с);

общее число опорных векторов (SV);

число связанных опорных векторов, для которых $\alpha_i = C$ (BSV);

доля несвязанных опорных векторов, (SV–BSV)/SV, характеризующая устойчивость SV-машины. (Под устойчивой SV-машиной понимается такая машина, для которой существует хотя бы один опорный вектор с $\alpha_i \neq C$. Такие опорные векторы называются несвязанными);

оценка качества алгоритма, полученная с использованием процедуры 10-кратной перекрестной проверки (CV_err);

верхняя граница оценки ошибки обобщения SV-классификатора (1) (J_err);

доля неправильно классифицированных образцов тестовой последовательности (Test_err).

Исследование SVM-классификаторов проводилось на реальных данных. Данные первой выборки (объем l = 1050, количество признаков n = 10) плотны, данные второй выборки (l = 4974, n = 50, размер тестовой последовательности m = 4998) разрежены.

Пример предварительной настройки параметра регуляризации C при фиксированном значении параметра ядра (σ = 2) по 282 образцам второй выборки иллюстрирует табл. 1. Очевидно, что при удачном выборе значения параметра C достигается разумный баланс между временем работы алгоритма и качеством/устойчивостью построенного классификатора.

Результаты численных экспериментов по исследованию сравнительной эффективности SVM-классификаторов приведены в табл. 2.

Отметим основные достоинства рассмотренных алгоритмов. Алгоритм IU является точным. Данные для обучения могут поступать как в пакетном режиме, так и по одному, в online-режиме. Это самый быстрый SVM-классификатор. SMO – самый быстрый из итерационных алгоритмов. С увеличением объема тренировочной последовательности время работы алгоритма растет существенно медленнее, чем в реализациях других алгоритмов. Алгоритм SOR при эффективной организации вычислительного процесса допускает проведение обучения на выборках огромных объемов. Предварительная настройка параметра ω, регулирующего скорость сходимости, позволяет значимо сокращать время работы SOR-алгоритма. SVMLight – быстрый итерационный алгоритм, приводящий к построению наиболее устойчивого SVM-классификатора.

Результаты экспериментов показали, что при работе с каждой выборкой каждый алгоритм требует своих значений параметров, которые необходимо подбирать очень тщательно. При этом следует ориентироваться как на средние показатели качества, так и на среднее время работы алгоритма. При удачном выборе параметров существенно сокращается время работы алгоритма без ущерба для качества обучения.

Таблица 2. Характеристики SV-классификаторов

Алгоритм	C	σ	CPU-time	SV/BSV	(SV– BSV)/SV	CV_err	
Первая выборка: l = 1050							J_err
SMO	8	1	46,812	429 / 410	0,044	0,158	0,227
SOR (ω = 0,7)	4	0,25	161,531	328 / 247	0,247	0,149	0,068
IU	2000	2	8,297	278 / 247	0,111	0,134	0,264
SVM_Light	100	10	315,937	415 / 15	0,964	0,127	0,036
Вторая выборка: l = 4974, m = 4998							Test_err
SMO	50	1	11721,704	375 / 40	0,893	0,002	0,006
SOR (ω = 0,7)	128	1	14326,578	333 / 5	0,985	0,001	0,005
IU	250	1,5	418,188	234 / 8	0,957	0,001	0,006
SVM_Light	150	4	17963,963	431 / 3	0,993	0,001	0,004

SV-алгоритмы активно используются во многих областях научных и практических исследований. Реализации некоторых алгоритмов построения SV-классификаторов включены во многие системы программного обеспечения, например в MatLab и R. Наиболее полная информация об общедоступном программном обеспечении SVM-алгоритмов представлена на сайте [21]. Работа [22] содержит обзор и сравнительный анализ различных реализаций SVM-алгоритмов, включенных в системы программного обеспечения, использующие язык программирования R. Мощным и удобным современным средством для обработки биологических данных является открытый проект с открытым кодом Bioconductor [23], также основанный на языке программирования R и предоставляющий высокопроизводительные средства для анализа геномных данных.

Работа выполнена при финансовой поддержке гранта Министерства образования и науки РФ по программе 5–100–2020.

СПИСОК ЛИТЕРАТУРЫ

1. Н. О. Кадырова и Л. В. Павлова, Биофизика **59** (3), 446 (2014).
2. V. N. Vapnik, *The Nature of Statistical Learning Theory* (Springer-Verlag, 2000).
3. A. Elisseeff and M. Pontil, in *Advances in Learning Theory: Methods, Models and Applications*, Ed. by J. A. K. Suykens, et al. (2003), Chapter 6, p. 111.
4. T. Joachims, in *Internat. Conf. Machine Learning* (ICML, 2000), p. 431.
5. T. Fawcett, in *ROC Graphs: Notes and Practical Considerations for Researchers* (Kluwer Academic Publishers, 2004), p. 1.
6. J. Davis and M. Goadrich, *Proc. 23rd Internat. Conf. on Machine Learning* (Pittsburgh, PA, 2006).
7. E. Edgar Osuna, R. Freund, and F. Girosi, IEEE, 279 (1997).
8. T. Joachims, in *Advanced Kernel Methods – Support Vector Learning* (MIT Press, 1998), p. 41.
9. J. Platt, in *Advances in Kernel Methods. Support Vector Learning* (MIT Press, 1998), p. 41.
10. J. Platt, in *Advances in Neural Information Processing Systems 11* (MIT Press, 1999), p. 557.
11. S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy, Technical Report CD–99–14, P. 1 (1999).
12. P.-H. Chen, R.-E. Fan, and C.-J. Lin, Lecture Notes in Artifical Intelligence **3734**, 45 (2005).
13. S. Keerthi and E. Gilbert, Machine Learning **46** (1–3), 351 (2002).
14. O. Mangasarian and D. Musicant, IEEE Transactions on Neural Networks **10** (5), 1032 (1999).
15. O. Mangasarian and D. Musicant, Data Discrimination via Nonlinear Generalized Support Vector Machines. OAI-PMH server at cs1.ist.psu.edu, 1999.
16. G. Cauwenberghs and T. Tomaso Poggio, in *Advances in Neural Information Processing Systems* (MIT Press, 2001), Vol. 13, p. 409.
17. P. Laskov, C. Gehl, S. Krüger, and K.-R. Müller, *Incremental Support* Vector Learning: Analysis, Implementation and Applications. OAI-PMH server at eprints.pascal-network.org, 2005.
18. G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd (John Hopkins University Press, Baltimore, London, 1996).
19. J. Jyrki Kivinen, S. Smola, and R. Williamson, IEEE Transactions on Signal Processing **52** (8), 2165 (2004).
20. S. Vishwanathan, N. Schraudolph, and A. Smola, J. Machine Learning Res. **6**, 1 (2005).
21. http://www.kernel-machines.org/software.
22. A. Karatzoglou, D. Meyer, and K. Hornik, J. Statistical Software **15** (9), 1 (2006).
23. http://www.bioconductor.org/packages/devel/bioc/html/MLSeq.html.

Comparative Efficiency of Algorithms Based on Support Vector Machines for Binary Classification

N.O. Kadyrova and L.V. Pavlova

Institute of Applied Mathematics and Mechanics, St. Petersburg State Polytechnical University, ul. Polytechnicheskaya 29, St. Petersburg, 195251 Russia

Methods of construction of support vector machines require no further a priori information and provide big data processing, what is especially important for various problems in computational biology. The question of the quality of learning algorithms is considered. The main algorithms of support vector machines for binary classification are reviewed and they were comparatively explored for their efficiencies. The critical analysis of the results of this study revealed the most effective support-vector-classifiers. The description of the recommended algorithms, sufficient for their practical implementation, is presented.

Key words: binary classification, support vector machines, kernel functions, algorithms based on support vector machines, comparative efficiency of support-vector-classifiers