САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет
Кафедра Системного Программирования

Кутькин Никита Андреевич

Исследование алгоритма SVM SMO для задач классификации и сравнение его с другими методами машинного обучения

Курсовая работа

Научный руководитель:
аспирант кафедры Системного Программирования
Невоструев К. Н.

Санкт-Петербург
2014

Оглавление

Введение

Предметная область – машинное обучение. Область практических применений обширна: автоматика, управление, экономика, социология, медицина, геология, астрономия, ядерная физика, биоинформатика и т.д.

Одна из наиболее распространенных задач машинного обучения – задача классификации. Для решения этой задачи требуется создание классифицирующей функции, которая присваивает каждому набору входных атрибутов значение метки одного из классов. Классификация входных значений производится после прохождения этапа «обучения», в процессе которого на вход обучающего алгоритма подаются входные данные с уже приписанными им значениями классов.

На сегодняшний день разработано большое число подходов к решению задач классификации. Одним из них является Метод Опорных Векторов (Support Vector Machines, SVM). В рамках курсовой работы проводится исследование алгоритма для построения SVM – SMO (Sequential Minimal Optimization). Работа предполагает реализацию алгоритма, тестирование на реальных данных, исследование методов автоматического подбора параметров и сравнение с другими алгоритмами машинного обучения.

1. Постановка задачи

В рамках курсовой работы ставилось несколько задач:

1. Реализация алгоритма SMO. Основные требования: скорость работы, простота использования, малый размер.
2. Исследование и реализация улучшений SMO: возможность использовать произвольные ядра, возможность мульти-классификации, прочие оптимизации.
3. Исследование методов автоматического подбора параметров для SVM. Реализация автоматического подбора параметров.
4. Тестирование на реальных данных. Данные нужно найти, привести к единому формату, провести предварительную обработку.
5. Сравнение с другими алгоритмами машинного обучения, а именно с их реализациями, выполненными другими студентами кафедры в аналогичных курсовых работах.

2. Обзор существующих алгоритмов

Как уже упоминалось, задача классификации – одна из самых распространенных задач машинного обучения. Поэтому неудивительно, что для её решения существует множество различных алгоритмов.

Самые известные – алгоритмы, реализующие искусственные нейронные сети, построенные по принципу организации и функционирования биологических нейронных сетей. По своей сути - это граф. Вершины в этом графе называются нейронами. Нейроны связаны взвешенными ребрами.

В качестве отдельного алгоритма обычно выделяют ELM (Extreme Learning Machine). Это нейронная сеть, в которой скрытые нейроны выбираются случайным образом.

Также стоит отметить так называемые деревья решений (Decision tree). На ребрах такого дерева записаны атрибуты, от которых зависит целевая функция, в листьях записаны значения целевой функции, а в остальных узлах — атрибуты, по которым различаются случаи. Чтобы классифицировать новый случай, надо спуститься по дереву до листа и выдать соответствующее значение.

Данная курсовая работа же посвящена SVM. Подход SVM реализует идею разделителя с максимальным зазором, предложенную В. Н. Вапником. Помимо SMO существуют и другие алгоритмы решения SVM. Прежде всего, алгоритм ASM (Active-Set Method), его сравнение с SMO можно найти, например, здесь [2]. Также, так как математическая формулировка идеи SVM приводит к задаче квадратичного программирования, то можно воспользоваться стандартными методами решения таких задач (методы квадратичного программирования), но на практике они мало применимы из-за ограничений по памяти.

3. SVM. Описание метода

3.1. Постановка задачи

Рассматривается задача обучения по прецедентам $\langle X, Y, y^*, X^l \rangle$ где X - пространство объектов, Y - множество ответов, $y^*: X \to Y$ - целевая зависимость, значения которой известны только на объектах обучающей выборки $X^l = (x_i, y_i)_{i=1}^l,\ y_i = y^*(x_i)$. Требуется построить алгоритм $a: X \to Y$, аппроксимирующий целевую зависимость на всём пространстве X.

Рассмотрим задачу классификации на два непересекающихся класса, в которой объекты описываются n-мерными вещественными векторами: $X = \mathbb{R}^n, Y = \{-1, +1\}$.

Будем строить линейный пороговый классификатор:

$$a(x) = sign\left(\sum_{j=1}^{n} w_j x^j - w_0\right) = sign(\langle w, x \rangle - w_0), \qquad (1)$$

где $x = (x^1, \dots, x^n)$ - признаковое описание объекта x; вектор $w = (w^1, \dots, w^n) \in \mathbb{R}^n$ и скалярный порог w_0 являются параметрами алгоритма. Уравнение $\langle w, x \rangle = w_0$ описывает гиперплоскость, разделяющую классы в пространстве $\mathbb{R}^n$.

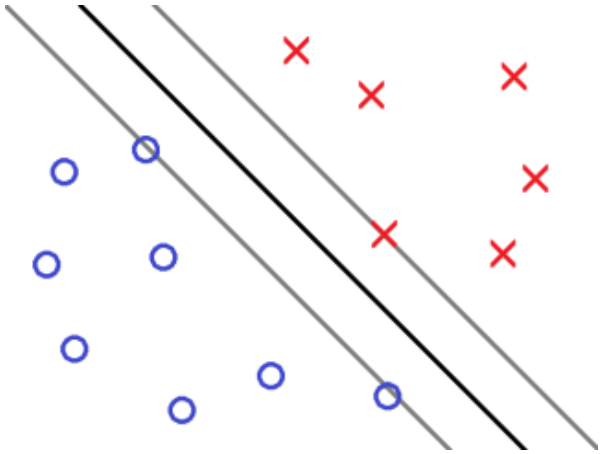
3.2. Понятие оптимальной разделяющей гиперплоскости

Предположим, что выборка линейно разделима, то есть существуют такие значения параметров w, w_0, при которых функционал числа ошибок

$$Q(w, w_0) = \sum_{i=1}^{l} [y_i(\langle w, x_i \rangle - w_0) < 0]$$

принимает нулевое значение. Но тогда разделяющая гиперплоскость не единственна, существуют и другие, реализующие то же самое разбиение выборки. Идея метода заключается в том, чтобы разумным образом распорядиться этой свободой выбора.

Потребуем, чтобы разделяющая гиперплоскость максимально далеко отстояла от ближайших к ней точек обоих классов. Первоначально данный принцип классификации возник из эвристических соображений: вполне естественно полагать, что максимизация зазора (margin) между классами должна способствовать более уверенной классификации.



3.3. Нормировка.

Заметим, что параметры линейного порогового классификатора определены с точностью до нормировки: алгоритм $a(x)$ не изменится, если w и w_0 одновременно умножить на одну и ту же положительную константу. Удобно выбрать эту константу таким образом, чтобы для всех пограничных (т. е. ближайших к разделяющей гиперплоскости) объектов $x_i \in X^l$ выполнялись условия

$$\langle w, x_i \rangle - w_0 = y_i$$

Сделать это возможно, поскольку при оптимальном положении разделяющей гиперплоскости все пограничные объекты находятся от неё на одинаковом расстоянии. Остальные объекты находятся дальше. Таким образом, для всех $x_i \in X^l$

$$\langle w, x_i \rangle - w_0 \begin{cases} \leq -1, \text{если } y_i = -1 \\ \geq +1, \text{если } y_i = +1 \end{cases} \qquad (2)$$

Условие $-1 < \langle w, x_i \rangle - w_0 < 1$ задаёт полосу, разделяющую классы. Ни одна из точек обучающей выборки не может лежать внутри этой полосы. Границами полосы служат две параллельные гиперплоскости с направляющим вектором w. Точки, ближайшие к разделяющей гиперплоскости, лежат в точности на границах полосы. При этом сама разделяющая гиперплоскость проходит ровно посередине полосы.

3.4. Ширина разделяющей полосы.

Чтобы разделяющая гиперплоскость как можно дальше отстояла от точек выборки, ширина полосы должна быть максимальной. Пусть x_- и x_+ - две произвольные точки классов −1 и +1 соответственно, лежащие на границе полосы. Тогда ширина полосы есть

$$\langle (x_+ - x_-), \frac{w}{||w||} \rangle = \frac{\langle w, x_+ \rangle - \langle w, x_- \rangle}{||w||} = \frac{(w_0 + 1) - (w_0 - 1)}{||w||} = \frac{2}{||w||}$$

Ширина полосы максимальна, когда норма вектора w минимальна.

Итак, в случае, когда выборка линейно разделима, достаточно простые геометрические соображения приводят к следующей задаче: требуется найти такие значения параметров w и w_0, при которых норма вектора w минимальна при условии (2). Это задача квадратичного программирования. Она будет подробно рассмотрена в следующем разделе. Затем будет сделано обобщение на тот случай, когда линейной разделимости нет.

3.5. Линейно разделимая выборка

Построение оптимальной разделяющей гиперплоскости сводится к минимизации квадратичной формы при l ограничениях-неравенствах вида (2) относительно n+1 переменных w, w_0:

$$\begin{cases} \langle w, w \rangle \to min \\ y_i(\langle w, x_i \rangle - w_0) \geq 1, \;\; i = 1, \dots, l \end{cases} \qquad (3)$$

По теореме Куна-Таккера эта задача эквивалентна двойственной задаче поиска седловой точки функции Лагранжа:

$$\begin{cases} L(w, w_0; \lambda) = \frac{1}{2}\langle w, w \rangle - \sum_{i=1}^{l} \lambda_i (y_i(\langle w, x_i \rangle - w_0) - 1) \to min_{w, w_0} max_{\lambda} \\ \lambda_i \geq 0, \;\; i = 1, \dots, l \\ \lambda_i = 0, \text{либо } \langle w, x_i \rangle - w_0 = y_i, \;\; i = 1, \dots, l \end{cases}$$

где $\lambda = (\lambda_1, \dots, \lambda_l)$ - вектор двойственных переменных. Последнее из трёх условий называется условием дополняющей нежёсткости.

Необходимым условием седловой точки является равенство нулю производных Лагранжиана. Отсюда немедленно вытекают два полезных соотношения:

$$\frac{dL}{dw} = w - \sum_{i=1}^{l} \lambda_i y_i x_i = 0 \qquad => \qquad w = \sum_{i=1}^{l} \lambda_i y_i x_i \qquad (4)$$

$$\frac{dL}{dw_0} = -\sum_{i=1}^{l} \lambda_i y_i = 0 \qquad => \qquad \sum_{i=1}^{l} \lambda_i y_i = 0 \qquad (5)$$

Из (4) следует, что искомый вектор весов w является линейной комбинацией векторов обучающей выборки, причём только тех, для которых $\lambda_i \neq 0$. Согласно условию дополняющей нежёсткости на этих векторах x_i ограничения-неравенства обращаются в равенства: $\langle w, x_i \rangle - w_0 = y_i$, следовательно, эти векторы находятся на границе разделяющей полосы. Все остальные векторы отстоят дальше от границы, для них $\lambda_i = 0$, и они не участвуют в сумме (1.4). Алгоритм (1) не изменился бы, если бы этих векторов вообще не было в обучающей выборке.

Если $\lambda_i > 0$ и $\langle w, x_i \rangle - w_0 = y_i$, то объект обучающей выборки x_i называется *опорным вектором* (support vector).

Подставляя (4) и (5) обратно в Лагранжиан, получим эквивалентную задачу квадратичного программирования, содержащую только двойственные переменные:

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{l} \lambda_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \lambda_i \lambda_j y_i y_j \big(\langle x_i, x_j \rangle\big) \quad \to min_\lambda \\ \lambda_i \geq 0, \;\; i = 1, \dots, l \\ \sum_{i=1}^{l} \lambda_i y_i = 0 \end{cases} \qquad (6)$$

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

Допустим, мы решили эту задачу. Тогда вектор w вычисляется по формуле (4). Для определения порога w_0 достаточно взять произвольный опорный вектор x_i и выразить w_0 из равенства $w_0 = \langle w, x_i \rangle - y_i$. На практике для повышения численной устойчивости рекомендуется брать в качестве w_0 среднее по всем опорным векторам или медиану.

В итоге алгоритм классификации может быть записан в следующем виде:

$$a(x) = sign(\sum_{i=1}^{l} \lambda_i y_i \langle x_i, x \rangle - w_0) \qquad (7)$$

Обратим внимание, что реально суммирование идёт не по всей выборке, а только по опорным векторам, для которых $\lambda_i \neq 0$. Именно это свойство разреженности является отличительной чертой SVM.

Резюмируя, отметим, что пока остаются открытыми два вопроса: как быть, если классы линейно не разделимы, и как решить двойственную задачу (6)?

Начнём с первого вопроса. В следующем разделе рассматривается обобщение двойственной задачи на случай отсутствия линейной разделимости. После этого будет рассмотрен переход от скалярных произведений к произвольным ядрам, так называемый «kernel trick», позволяющий строить нелинейные разделители.

3.6. Линейно неразделимая выборка

Чтобы обобщить SVM на случай линейной неразделимости, позволим алгоритму допускать ошибки на обучающих объектах, но при этом постараемся, чтобы ошибок было поменьше. Введём набор дополнительных переменных $\xi_i \geq 0$, характеризующих величину ошибки на объектах $x_i, \;\; i = 1, \dots, l$. Возьмём за отправную точку задачу (3); смягчим в ней ограничения-неравенства, и одновременно введём в минимизируемый функционал штраф за суммарную ошибку:

$$\begin{cases} \frac{1}{2}\langle w,w\rangle + C\sum_{i=1}^{l}\xi_i \to min_{w,w_0,\xi} \\ y_i(\langle w,x_i\rangle - w_0) \geq 1-\xi_i, \;\; i=1,\dots,l \\ \xi_i \geq 0, \;\; i=1,\dots,l \end{cases} \qquad (8)$$

Отступом (margin) объекта x_i от границы классов называется величина $m_i = y_i(\langle w,x_i\rangle - w_0)$. Алгоритм допускает ошибку на объекте x_i тогда и только тогда, когда отступ m_i отрицателен. Если $m_i \in (-1,+1)$, то объект x_i попадает внутрь разделяющей полосы. Если $m_i > 1$, то объект x_i классифицируется правильно, и находится на некотором удалении от разделяющей полосы.

Положительная константа C является управляющим параметром метода и позволяет находить компромисс между максимизацией разделяющей полосы и минимизацией суммарной ошибки.

Запишем функцию Лагранжа задачи (8):

$$L(w,w_0,\xi;\lambda,\eta) = \frac{1}{2}\langle w,w\rangle + \sum_{i=1}^{l}\lambda_i(y_i(\langle w,x_i\rangle - w_0) - 1) - \sum_{i=1}^{l}\xi_i(\lambda_i + \eta_i - C)$$

где $\eta = (\eta_1, \dots, \eta_\ell)$ - вектор переменных, двойственных к переменным $\xi = (\xi_1, \dots, \xi_\ell)$. Как и в прошлый раз, условия Куна-Таккера сводят задачу к поиску седловой точки функции Лагранжа:

$$\begin{cases} L(w,w_0,\xi;\lambda,\eta) \to min_{w,w_0,\xi} max_{\lambda,\eta} \\ \xi_i \geq 0, \lambda_i \geq 0, \eta_i \geq 0, \;\; i=1,\dots,l \\ \lambda_i = 0, \text{либо } y_i(\langle w,x_i\rangle - w_0) = 1-\xi_i, \;\; i=1,\dots,l \\ \eta_i = 0 \text{ либо } \xi_i = 0, \;\; i=1,\dots,l \end{cases}$$

В последних двух строках записаны условия дополняющей нежёсткости. Необходимым условием седловой точки является равенство нулю производных Лагранжиана. Отсюда получаются три полезных соотношения:

$$\frac{dL}{dw} = w - \sum_{i=1}^{l}\lambda_i y_i x_i = 0 \qquad => \qquad w = \sum_{i=1}^{l}\lambda_i y_i x_i \qquad (9)$$

$$\frac{dL}{dw_0} = -\sum_{i=1}^{l}\lambda_i y_i = 0 \qquad => \qquad \sum_{i=1}^{l}\lambda_i y_i = 0 \qquad (10)$$

$$\frac{dL}{d\xi_i} = -\lambda_i - \eta_i + C = 0 \qquad => \qquad \lambda_i + \eta_i = C \qquad (11)$$

Первые два соотношения в точности такие же, как и в линейно разделимом случае. Из третьего соотношения и неравенства $\eta_i \geq 0$ следует ограничение $\lambda_i \leq C$. Отсюда, и из условий дополняющей нежёсткости вытекает, что возможны только три допустимых сочетания значений переменных ξ_i, λ_i, η_i и отступов m_i.

Соответственно, все объекты $x_i,\ i = 1, \ldots, l$ делятся на следующие три типа:

1. $\xi_i = 0;\ \lambda_i = 0;\ \eta_i = C;\ m_i > 1$. Объект x_i классифицируется правильно и находится далеко от разделяющей полосы. Такие объекты будем называть *периферийными*.
2. $\xi_i = 0;\ 0 < \lambda_i < C;\ 0 < \eta_i < C;\ m_i = 1$. Объект x_i классифицируется правильно и лежит в точности на границе разделяющей полосы. Такие объекты, как и раньше, будем называть *опорными*.
3. $\xi_i > 0;\ \lambda_i = C;\ \eta_i = 0;\ m_i < 1$. Объект x_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 x_i будем называть *нарушителем*.

В силу соотношения (11) в Лагранжиане обнуляются все члены, содержащие переменные ξ_i и η_i, и он принимает тот же вид, что и в случае линейной разделимости. Параметры разделяющей поверхности w и w_0, согласно формулам (9) и (10), также выражаются только через двойственные переменные λ_i. Таким образом, задача снова сводится к квадратичному программированию относительно двойственных переменных λ_i. Единственное отличие от линейно разделимого случая состоит в появлении ограничения сверху $\lambda_i \leq C$:

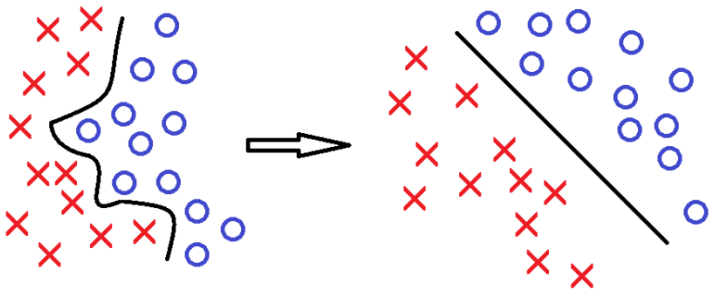
$$\begin{cases} -L(\lambda) = -\sum_{i=1}^{l} \lambda_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \lambda_i \lambda_j y_i y_j \big(\langle x_i, x_j \rangle\big) \quad \rightarrow min_\lambda \\ 0 \leq \lambda_i \leq C,\ \ i = 1, \ldots, l \\ \sum_{i=1}^{l} \lambda_i y_i = 0 \end{cases} \qquad (12)$$

На практике для построения SVM решают именно эту задачу, а не (6), так как гарантировать линейную разделимость выборки в общем случае не представляется возможным. Этот вариант алгоритма называют SVM с мягким зазором (soft-margin SVM), тогда как в линейно разделимом случае говорят об SVM с жёстким зазором (hard-margin SVM).

Для алгоритма классификации сохраняется формула (7), с той лишь разницей, что теперь ненулевыми λ_i обладают не только опорные объекты, но и объекты-нарушители.

3.7. Ядра и спрямляющие пространства

Существует подход к решению проблемы линейной неразделимости. Это переход от исходного пространства признаковых описаний объектов X к новому пространству H с помощью некоторого преобразования $\psi: X \to H$. Если пространство H имеет достаточно высокую размерность, то можно надеяться, что в нём выборка окажется линейно разделимой (легко показать, что если выборка X^l не противоречива, то всегда найдётся пространство размерности не более l, в котором она будет линейно разделима). Пространство H называют *спрямляющим*.



Если предположить, что признаковыми описаниями объектов являются векторы $\psi(x_i)$, а не векторы x_i, то построение SVM проводится точно так же, как и ранее. Единственное отличие состоит в том, что скалярное произведение $\langle x, x' \rangle$ в пространстве X всюду заменяется скалярным произведением $\langle \psi(x), \psi(x') \rangle$ в пространстве H. Отсюда вытекает естественное требование: пространство H должно быть наделено скалярным произведением, в частности, подойдёт любое евклидово, а в общем случае и гильбертово, пространство.

Функция $K: X \times X \to \mathbb{R}$ называется *ядром* (kernel function), если она представима в виде $K(x, x') = \langle \psi(x), \psi(x') \rangle$ при некотором отображении $\psi: X \to H$, где H - пространство со скалярным произведением.

Постановка задачи (1.14), и сам алгоритм классификации (7) зависят только от скалярных произведений объектов, но не от самих признаковых описаний. Это означает, что скалярное произведение $\langle x, x' \rangle$ можно формально заменить ядром $K(x, x')$. Поскольку ядро в общем случае нелинейно, такая замена приводит к существенному расширению множества реализуемых алгоритмов $a: X \to Y$.

Более того, можно вообще не строить спрямляющее пространство H в явном виде, и вместо подбора отображения ψ заниматься непосредственно подбором ядра.

Можно пойти ещё дальше, и вовсе отказаться от признаковых описаний объектов. Во многих практических задачах объекты изначально задаются информацией об их попарном взаимоотношении, например, отношении сходства. Если эта информация допускает представление в виде двуместной функции $K(x, x')$, удовлетворяющей аксиомам скалярного произведения, то задача может решаться методом SVM. Для такого подхода недавно был придуман термин беспризнаковое распознавание (featureless recognition).

4. SMO-подобные алгоритмы

Перепишем двойственную задачу (12) в матричных обозначениях. Введём матрицу $Q = y_i y_j K(x_i, x_i)$ размера $l \times l$ и три вектор-столбца длины l: вектор ответов $y = (y_i), i = 1, \dots, l$, вектор двойственных переменных $\lambda = (\lambda_i),\ i = 1, \dots, l$ и вектор единиц e. Тогда задачу (1.14) можно переписать в виде

$$\begin{cases} \frac{1}{2}\lambda^T Q\lambda - e^T\lambda \to min_\lambda \\ 0 \le \lambda_i \le C,\ \ i = 1, \dots, l \\ y^T\lambda = 0 \end{cases} \qquad (13)$$

Матрица Q может быть слишком большой для хранения в памяти, поэтому применяются методы декомпозиции (Decomposition methods). В отличие от большинства методов оптимизации, обновляющих весь вектор λ на каждой итерации, метод декомпозиции изменяет только подмножество λ на каждой из итераций. Это подмножество называется *рабочим множеством* (working set). В случае алгоритма SMO рабочее множество состоит всего из двух элементов. Поэтому каждая итерация – решение простой проблемы со всего двумя переменными.

Общая схема SMO:

1. инициировать λ значениями, удовлетворяющими ограничениям
2. выбрать рабочее множество $\{\lambda_i, \lambda_j\}$
3. оптимизировать пару λ_i, λ_j
4. вычислить новое значение w_0
5. **если** параметры системы изменились, то переход на п.2
   иначе, переход на следующий пункт.
6. Конец работы

Составные части алгоритма:

1. эвристический метод выбора двух множителей λ_i, λ_j для оптимизации
2. аналитический метод решения для двух множителей λ_i, λ_j
3. метод вычисление w_0

Существует несколько алгоритмов, которые работают по подобной схеме, различия кроются в эвристических методах выбора рабочего множества. SMO – первый подобный алгоритм, но не самый быстрый. В рамках курсовой работы сначала был реализован алгоритм SMO, но позже он был заменен на SMO-подобный алгоритм, использующий при выборе рабочего множества информацию второго порядка для достижения более быстрой сходимости (Working Set Selection Using Second Order Information). Алгоритм описан в [3]. Реализация выполнена на языке программирования C, выбор языка обосновывается требованиями к скорости работы алгоритма.

5. Улучшения и оптимизации

5.1. Мульти-классификация

SVM решает задачу классификации при $Y = \{-1, +1\}$, то есть предполагается, что все объекты исходного множества X принадлежат одному из двух классов. Если классов больше одного, то появляется так называемая задача мульти-классификации.

Два наиболее популярных метода её решения для SVM это один-против-всех (one vs. all) и каждый-против-каждого (one vs. one).

Метод один-против-всех состоит в обучении одной SVM на каждый из классов. Каждая такая SVM способна отличать объекты своего класса от остальных. Для классификации произвольного объекта нужно выбрать SVM с максимальным результатом.

Метод каждый-против-каждого состоит в обучении одной SVM на каждую пару классов (если всего классов n – обучаем $n(n-1)/2$ SVM). Каждая такая SVM способна отличать объекты одного класса от объектов другого. Для классификации произвольного объекта все SVM голосуют за один из классов и, затем, выбирается класс с наибольшим числом голосов.

Для реализации был выбран метод каждый-против-каждого, так как согласно различным исследованиям (например [4]) он работает быстрее при сравнимых значениях точности.

5.2. Автоматический подбор параметров

Для обучения SVM необходимо выбрать константу C и параметры ядра. В частности, при использовании линейного ядра ($K(x, x') = \langle x, x' \rangle$) никаких параметров подбирать не нужно, а при использовании RBF ядра ($K(x, x') = -\gamma||x - x'||^2$) необходимо выбрать параметр γ.

Метод подбора параметров такой:

1. Выбираем граничные значения всех параметров и шаг
2. Строим соответствующую n-мерную сетку (n – число параметров)
3. Обучаем SVM на небольшой обучающей выборке с параметрами соответствующими узлам сетки.
4. Выбираем наилучшие из опробованных параметров.

Была написана утилита, реализующая данный метод.

5.3. Оптимизации

Также были исследованы возможные оптимизации алгоритма: кэширование и shrinking (сокращение).

Как уже упоминалось, используемая в решении задачи (13) матрица Q не хранится в памяти: при большой обучающей выборке она может достигать гигантских размеров. При этом основное время работы алгоритма – вычисление элементов матрицы Q. Логичным выглядит кэшировать её значения. Проблема в том, что выбор двух значений для оптимизации, производящийся на каждой итерации алгоритма, непредсказуем. Поэтому, при большом размере Q, а именно в этом случае и нужно кэширование, велика вероятность, что при повторном обращении к какому-либо элементу матрицы, его значение уже давно удалено из кэша. Реализация кэширования «в лоб» не только не ускоряет, но и замедляет работу алгоритма. Возможное решение данной проблемы – ограничить выбор значений для оптимизации некоторым заранее выделенным множеством; когда все его значения оптимизированы, выбрать новое. К сожалению, данная оптимизация требует значительных изменений алгоритма, поэтому из-за нехватки времени она не была реализована.

Идея shrinking, вкратце, такова: многие граничные значения (значения, равные 0 илиC) вычисляются задолго до конца алгоритма, но по-прежнему участвуют в выборе пар для оптимизации. Отсюда вытекает желание убрать их из этого процесса. Требуемые модификации алгоритма во многом схожи с модификациями, необходимыми для кэширования, поэтому логичным выглядит использовать одновременно обе эти оптимизации.

Самая же простая и наиболее эффективная оптимизация – это предварительное вычисление и хранение, диагональных элементов матрицы Q и скалярных квадратов всех векторов. Идея такая же, как и у кэширования: не производить одинаковые вычисления более одного раза. Данная оптимизация требует малый объем памяти (линейный от размера обучающей выборки), при этом диагональные элементы Q используются на каждой итерации алгоритма, а скалярные квадраты векторов могут использоваться при вычислении внедиагональных элементов Q (зависит от используемого ядра, например, верно в случае ядра RBF). Эта оптимизация была включена в реализацию алгоритма.

6. Тестирование

6.1. Данные

Для проведения тестирования алгоритмов построения SVM необходимы специальные данные. Необходимо найти их, привести к единому формату, провести предварительную обработку (приведение параметров к диапазону $[0; 1]$ или $[-1; 1]$).

В качестве формата был выбран .csv (Comma-Separated Values) — значения, разделённые запятыми — текстовый формат, предназначенный для представления табличных данных. Данный формат подходит идеально, ведь входные данные – суть таблица $k \times n$, где k – количество векторов, а n – размерность X.

Тестирование проводилось на 5 наборах данных, а именно:

1. MNIST (10 классов, 784 параметра, обучающая выборка 60000) – распознавание цифр.
2. Poker (10 классов, 85 параметра, обучающая выборка 25010) – распознавание покерных рук.
3. Adult (2 классов, 123 параметра, обучающая выборка 32561) – определить зарабатывает ли человек 50000$ в год.
4. Titanic (2 классов, 6 параметра, обучающая выборка 1526) – определить утонул ли данный пассажир Титаника.
5. Diabetes (2 классов, 8 параметра, обучающая выборка 576) – определить болен ли человек диабетом.

Наборы данных взяты с [6].

6.2. Тестирование алгоритма

В таблице ниже приведены результаты обучения SVM на всех наборах данных с заранее выбранными параметрами обучения. Все тесты производились на машине с Intel Core i5 M430 2.27GHz, 4 ГБ ОЗУ.

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
MNIST	3578	98.57	24
Poker	4200	92.38	4
Adult	197	85.08	2
Titanic	0.07	77.31	<1
Diabetes	0.108	77.08	<1

6.3. Сравнение с другими алгоритмами

В таблицах ниже приведены результаты обучения SVM в сравнении с алгоритмами ELM (Extreme Learning Machine) и Нейронная сеть (Neural Network):

Набор данных MNIST

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
SMO	3578	98.57	24
Neural Network	900	96.39	<1
ELM	836	96.1	1.1

Набор данных Poker

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
SMO	4200	92.38	4
Neural Network	248	92.49	<1
ELM	109	72.5	1.2

Набор данных Adult

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
SMO	197	85.08	2.1
Neural Network	75	85.08	<1
ELM	30	85	<1

Набор данных Titanic

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
SMO	0.07	77.31	<1
Neural Network	0.06	77.62	<1
ELM	0.017	77.3	<1

Набор данных Diabetes

	Время обучения, сек	Точность, %	Время на 1 предсказание, мсек
SMO	0.108	77.08	1.8
Neural Network	1.4	75.52	<1
ELM	0.007	77	<1

Из полученных результатов можно сделать следующие выводы:

1. По точности обучения SVM не уступает, а иногда и превосходит ELM и нейронные сети.
2. По скорости обучения и предсказания SVM уступает как ELM, так и нейронным сетям.

То есть, выбор SVM является осмысленным, если мы отдаём предпочтение точности над скоростью.

При небольших размерах наборов данных логичным выглядит проверка всех возможных алгоритмов и последующий выбор алгоритма с наибольшей точностью предсказания. При отсутствии тестовой выборки точность можно оценить с помощью перекрёстной проверки (Cross-validation). Суть метода: имеющиеся в наличии данные разбиваются на k частей. Затем на k−1 частях данных производится обучение, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В качестве итоговой точности берется средняя точность по всем k циклам.

Заключение

В рамках курсовой работы был изучен метод построения классификаторов SVM и реализован SMO-подобный алгоритм решения SVM.

Изучены способы производить мульти-классификацию при использовании SVM, реализован один из них. Также реализован один из методов автоматического подбора параметров для обучения. Исследованы возможные оптимизации алгоритма, применены некоторые из них.

Подготовлены данные и проведено тестирование алгоритма, выполнено сравнение скорости и точности обучения с другими алгоритмами машинного обучения (ELM, нейронные сети). По результатам тестирования, к плюсам SMO можно отнести точность, к минусам – скорость. Отсюда вывод: SMO следует использовать тогда, когда точность предсказания критична.

Список литературы

[1] Platt. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. URL: http://research.microsoft.com/en-us/um/people/jplatt/smo-book.pdf

[2] Michael Vogt, Vojislav Kecman. Active-Set Methods for Support Vector Machines. URL: http://www.support-vector.ws/Book_chapter_Active_Set_for_SVM_Vogt_Kecman.pdf

[3] Rong-En Fan, Pai-Hsuen Chen, Chih-Jen Lin. Working Set Selection Using Second Order Information for Training Support Vector Machines. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf

[4] Jonathan Milgram, Mohamed Cheriet, Robert Sabourin. “One Against One” or “One Against All”. Which One is Better for Handwriting Recognition with SVMs? URL: http://hal.archives-ouvertes.fr/docs/00/10/39/55/PDF/cr102875872670.pdf

[5] К. В. Воронцов. Лекции по методу опорных векторов. 21 декабря 2007 г. URL: http://www.ccas.ru/voron/download/SVM.pdf

[6] UCI Machine Learning Repository. URL: http://archive.ics.uci.edu/ml/