Вариант параллельной реализации процесса обучения машин опорных векторов на основе алгоритма *Chunking*

Е. В. Котельников, Т. А. Стародубова, А. В. Котельникова

В статье рассматриваются способы повышения производительности одного из наиболее эффективных алгоритмов машинного обучения – метода опорных векторов (SVM). Для распараллеливания процесса обучения SVM используется идея алгоритма образования фрагментов *Chunking*. Предложены три стратегии распараллеливания и приведены их теоретические временны*е* оценки.

1. Введение

В области машинного обучения, в частности в системах обработки текстов, существуют проблемы, требующие высокопроизводительных алгоритмов, решающих задачи с высокой точностью. Одним из наиболее эффективных считается метод опорных векторов (Support Vector Machines, SVM) [1].

Точность распознавания, достижимая при использовании метода опорных векторов, является одной из самых высоких среди соответствующих алгоритмов (например, нейронных сетей). Однако время процесса обучения, как правило, квадратично зависит от числа обучающих примеров (векторов) и для ряда задач недопустимо велико. В качестве примера можно привести задачу текстовой классификации, где количество обучающих векторов, как и размерность пространства признаков, могут достигать 10^5.

На данный момент существует множество методов обучения SVM, например, метод последовательной минимальной оптимизации (Sequential Minimal Optimization, SMO) [2], градиентный алгоритм Kernel-Adatron, инкрементные и декрементные алгоритмы, вероятностные методы и др. Все эти методы довольно эффективно строят SVM-классификатор для наборов данных небольшой размерности и ограниченного объема. При этом распараллеливание процесса обучения представляет непростую задачу, так как в большинстве эффективных алгоритмов построения классификатора присутствует множество зависимостей по данным.

В связи с этим возникает идея использования в качестве основы распараллеливания возможно медленного, но не обладающего зависимостями по данным, алгоритма обучения.

Одним из первых методов, предложенных для обучения SVM, являлся метод образования фрагментов *Chunking* [3]. Во многих работах (см., например, [2]) было показано, что данный метод работает намного медленнее других (разница достигает порядка), более новых алгоритмов. Однако идея, заложенная в методе *Chunking*, позволяет реализовать несколько интересных стратегий распараллеливания, неприменимых или ограниченно применимых для других алгоритмов.

2. Алгоритм образования фрагментов (Chunking)

Алгоритм *Chunking* можно отнести к методам декомпозиции, идея которых основана на разбиении одной большой задачи на ряд подзадач. Главным преимуществом методов декомпозиции является то, что они предлагают алгоритмы с требованиями памяти, которые линейны относительно количества обучающих примеров и количества опорных векторов.

Чтобы избежать решения задачи большой размерности, обучающие данные разделяются на фрагменты (*chunks*), которые обрабатываются итеративно.

Алгоритм *Chunking* заключается в следующем.

1) Формируется начальный набор данных, который является случайным небольшим подмножеством обучающих данных. Желательно, чтобы в это подмножество входили положительные и отрицательные примеры в той же пропорции, как и в исходном множестве.

2) На созданном наборе данных происходит обучение. Результатом обучения является классификатор или, иными словами, множество опорных векторов мощностью N_{SV}.
3) Все данные, не вошедшие в текущий обучающий набор, тестируются на полученном классификаторе, в результате чего строится список векторов, которые были неправильно классифицированы. Если список пуст или его размер не превышает некоторого допустимого уровня ошибок, процесс обучения завершается.
4) Полученный список векторов сортируется по убыванию величины ошибки, т. е. по расстоянию до разделяющей плоскости.
5) Формируется новый набор данных путем объединения первых N векторов из отсортированного списка неправильно классифицированных векторов с уже найденными N_{SV} опорными векторами, где сумма $N + N_{SV}$ определяется эвристически (размер набора векторов, который бы рос слишком быстро или слишком медленно, давал бы общую медленную сходимость).
6) Переход к шагу 2.

Следует отметить, что векторы, являющиеся опорными на одном из этапов обучения, могут не оказаться в конечном решении.

Заметим также, что удачный начальный выбор обучающего подмножества может существенно уменьшить время обучения классификатора.

3. Стратегии распараллеливания

В данном разделе будут предложены три варианта распараллеливания процесса обучения SVM на основе алгоритма *Chunking*.

3.1. Первая стратегия (*Предварительное обучение*)

Предлагается в алгоритм *Chunking* ввести этап параллельного предварительного обучения, который позволит получить в определенном смысле «хороший» начальный набор обучающих данных.

Рассмотрим данную стратегию подробнее. На этапе предварительного обучения всё исходное множество данных разделяется на P непересекающихся подмножеств по числу доступных процессорных узлов. При делении следует сохранять пропорцию положительных и отрицательных примеров, как в исходном множестве. Полученные подмножества отправляются на процессорные узлы, где происходит обучение SVM-классификаторов. Отметим, что для обучения можно использовать любой доступный алгоритм, например, метод последовательной минимальной оптимизации SMO.

В результате обучения на каждом узле формируется набор опорных векторов. Объединение полученных наборов опорных векторов является начальным набором для обучения в алгоритме *Chunking*. Таким образом, первая стратегия является способом параллельного быстрого поиска приближенного решения, которое затем уточняется стандартными методами.

3.2. Вторая стратегия (*Турнир*)

В этом варианте распараллеливания проводится турнирный отбор между классификаторами процессоров. На каждом этапе турнира результаты классификации для пары процессоров объединяются.

Приведем подробное описание второй стратегии (см. рис. 1).

1) Множество обучающих примеров разделяется на P непересекающихся подмножеств и распределяется по процессорам, как и в первой стратегии, с сохранением пропорции положительных и отрицательных примеров.

2) На каждом процессоре параллельно осуществляется обучение своего классификатора, например с использованием алгоритма SMO, таким образом, формируется множество опорных векторов.

3) Каждый процессор параллельно тестирует свой классификатор на примерах другого процессора из пары. Формируется множество ошибочно распознанных векторов.

4) Из пары классификаторов выбирается один с наименьшим числом ошибок на примерах другого классификатора.

5) Формируется новый обучающий набор для выбранного процессора, состоящий из опорных векторов обоих классификаторов и ошибочных векторов проигравшего на этапе 4 классификатора.

6) Шаги со 2-го по 5-й повторяются, причем на каждой итерации число задействованных процессоров уменьшается вдвое, пока не останется единственный процессор.

7) Классификатор оставшегося процессора обучается на сформированном на шаге 5 наборе данных.

8) Обученный классификатор тестируется на всем наборе примеров. Если ошибок нет, процесс обучения заканчивается. Иначе формируется новый обучающий набор из опорных векторов классификатора и ошибочных векторов. Классификатор обучается на полученном наборе.

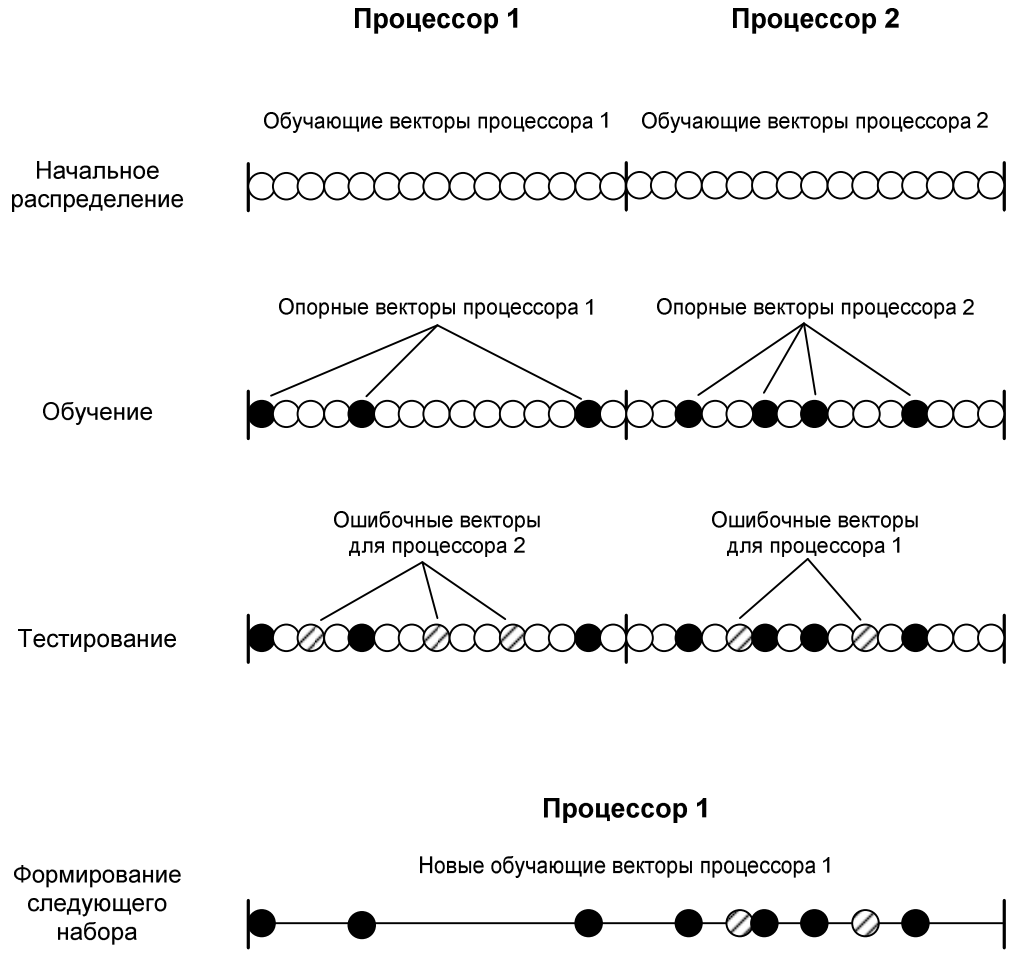

Рис. 1. Стратегия турнирного отбора для двух процессоров

3.3. Третья стратегия (*Гонки*)

Здесь реализован вариант поисковой декомпозиции (Exploratory Decomposition) [4]. Исходное множество обучающих данных целиком передается каждому процессорному узлу, после чего процессор начинает независимо от других выполнять обучение SVM по алгоритму *Chunking*. Идея заключается в том, что каждый процессор начинает работать со своим начальным набором данных, непересекающимся с начальными наборами данных других процессоров. Как отмечалось выше, удачный выбор начального обучающего подмножества в алгоритме *Chunking* может существенно сократить время обучения. В итоге процессор, первым нашедший решение, сообщает об этом остальным, а время, затраченное им на поиск, считается общим временем обучения.

Таким образом, поиск решения происходит в одном и том же пространстве поиска, но с разными начальными условиями.

4. Оценка стратегий

В этом разделе будет приведена временн*а*я оценка предложенных параллельных стратегий обучения.

Обозначим: N – число обучающих примеров, N_{SV} – число опорных векторов, P – количество вычислительных узлов в системе, $k = N_{SV} / N$ – отношение числа опорных векторов к общему числу примеров, S – коэффициент ускорения параллельной стратегии.

1) В **первой стратегии (*Предварительное обучение*)** небольшие временные затраты на параллельную часть алгоритма могут дать «хороший» начальный обучающий набор.

Укажем приближенную временную оценку данной стратегии в лучшем и в худшем случаях. В качестве базового метода для обучения отдельных классификаторов выбран алгоритм последовательной минимальной оптимизации (SMO), зависимость времени обучения которого от количества тренировочных примеров пропорциональна для разных данных от N до $N^{2,2}$ [2]. Учитывая это, примем, что время обучения отдельного классификатора пропорционально N^2. Также считаем, что время, затрачиваемое на тестирование классификатором набора данных, мало, по сравнению с временем обучения.

Время параллельного предварительного обучения при разделении исходного набора данных по P процессорам, будет пропорционально $\left(\frac{N}{P}\right)^2$. В результате будут сформированы P наборов данных, содержащих в среднем по $\left(\frac{kN}{P}\right)$ опорных векторов, если допустить, что опорные векторы равномерно распределены по начальным обучающим наборам. Полученные наборы объединяются в единый набор, включающий kN векторов. На этом наборе время обучения составляет $(kN)^2$, а в результате будет получено окончательное решение, если все опорные векторы вошли в объединенный набор. Таким образом, время обучения для первой стратегии:

$$T_1 \approx \frac{N^2}{P^2} + (kN)^2 = N^2\left(\frac{1}{P^2} + k^2\right). \qquad (1)$$

Ускорение для первой стратегии, если принять, что для лучшего последовательного алгоритма время обучения пропорционально N^2, будет выражаться следующим отношением:

$$S_1 \approx \frac{N^2}{N^2\left(\frac{1}{P^2} + k^2\right)} = \frac{P^2}{1 + P^2 k^2}. \qquad (2)$$

В приведенной формуле не учитываются время на тестирование, затраты на пересылки в случае распределенной памяти, время дообучения классификатора в ситуации, когда объединенный набор содержит не все опорные векторы.

Временн*а*я оценка в лучшем (идеальном) и худшем случаях будет определяться коэффициентом относительного числа опорных векторов k:

в идеальном случае $k \to 0$: $T_1 \approx \frac{N^2}{P^2}$, $S_1 \approx P^2$;

в худшем случае $k \to 1$: $T_1 \approx N^2\left(\frac{1}{P^2} + 1\right)$; $S_1 = \frac{P^2}{1 + P^2}$.

Следует отметить, что оценка для идеального случая на практике недостижима вследствие указанных для формулы (2) ограничений.

Таким образом, ускорение для первой стратегии (как и для двух следующих) существенно зависит от числа опорных векторов, которое в свою очередь определяется характеристиками как набора данных, так и параметров классификатора. При большом числе опорных векторов может иметь место замедление параллельной стратегии по сравнению с последовательным алгоритмом.

2) Во **второй стратегии (*Турнир*)** присутствуют существенные затраты на пересылки наборов данных (в случае распределенной памяти) при высокой вероятности ускорения процесса обучения по сравнению с последовательным алгоритмом.

Приведем приближенную временную оценку. На первом этапе турнира участвуют P процессоров, время обучения пропорционально $\left(\frac{N}{P}\right)^2$. Во второй этап выходят $P/2$ победивших процессоров, каждый с двумя наборами опорных векторов – своим и проигравшего процессора (ошибки не учитываем), время обучения на втором этапе пропорционально $\left(\frac{2k \cdot N}{P}\right)^2$. Турнир продолжается, пока не останется один процессор. Таким образом, для второй стратегии получаем:

$$T_2 \approx \left(\frac{N}{P}\right)^2 + \left(\frac{2k \cdot N}{P}\right)^2 + \left(\frac{4k^2 \cdot N}{P}\right)^2 + \ldots = \left(\frac{N}{P}\right)^2 \cdot \sum_{i=0}^{\log_2 P} (2k)^{2i}, \left(P = 2^m, m = 1,2,3,\ldots\right). \quad (3)$$

Как и в первой стратегии, ускорение в лучшем и в худшем случаях зависит от относительного количества опорных векторов k:

$$S_2 = \frac{P^2}{\sum_{i=0}^{\log_2 P} (2k)^{2i}}. \quad (4)$$

В идеальном случае $k \to 0$: $T_2 \approx \frac{N^2}{P^2}$, $S_2 \approx P^2$;

в худшем случае $k \to 1$: $T_2 \approx \left(\frac{N}{P}\right)^2 \cdot \sum_{i=0}^{\log_2 P} 4^i$, $S_2 \approx \frac{P^2}{\sum_{i=0}^{\log_2 P} 4^i}$.

В идеальном случае ускорение второй стратегии теоретически совпадает с ускорением первой стратегии, в худшем случае замедление во второй стратегии оказывается больше, чем в первой. На практике (для распределенных систем) ситуация во второй стратегии усложняется существенно б*о*льшим количеством пересылок по сравнению с первой.

3) **Третья стратегия (*Гонки*)** – в связи с выбором разных начальных наборов данных повышается вероятность ускорения процесса обучения, но в итоге только один процессор выполняет полезную работу, хотя простои отсутствуют.

Наилучшая временная оценка для данной стратегии будет в случае, когда количество опорных векторов невелико и все они сосредоточены в начальном наборе данных, с которого один из процессоров начинает обучение. Время обучения будет пропорционально квадрату размера начального набора, который можно принять равным kN. Наихудшая оценка имеет место в случае, когда количество опорных векторов сравнимо с числом обучающих данных. Такая оценка совпадает с временем работы последовательного алгоритма *Chunking* и для разных наборов данных [2] варьируется от $N^{1,2}$ до $N^{3,4}$ (примем N^3).

Таким образом, приближенные временные оценки получаются следующими:

в лучшем случае: $T_3 \approx (kN)^2$, $S_3 \approx \frac{1}{k^2}$;

в худшем случае: $T_3 \approx N^3$, $S_3 \approx \frac{1}{N}$.

5. Программная реализация и эксперименты

Предложенные стратегии параллельного обучения SVM-классификатора на основе алгоритма *Chunking* были реализованы на двух платформах: многоядерная архитектура под

управлением операционной системы Microsoft Windows и кластерная архитектура под управлением Microsoft Windows Compute Cluster Server 2003.

Обе реализации разрабатывались в среде Microsoft Visual Studio 2005 на языке C# на основе принципов объектно-ориентированного проектирования и программирования.

Для многоядерной платформы было создано многопоточное приложение на основе системного класса *Thread* из пространства имен *System.Threading*.

Приложение для кластерной архитектуры разрабатывалось на основе библиотеки MPI.NET версии 1.0 [5] – свободно доступной реализации интерфейса передачи сообщений MPI для среды Microsoft.NET, позволяющей разрабатывать приложения для MPI на C# и других языках .NET.

В настоящее время авторы осуществляют экспериментальное тестирование предложенных стратегий распараллеливания процесса обучения машин опорных векторов, используя реальные и искусственные наборы данных. Экспериментальные расчеты для кластерной архитектуры проводятся на вычислительном кластере Вятского государственного гуманитарного университета, состоящем из 12 вычислительных узлов. Каждый вычислительный узел представляет собой персональный компьютер с процессором AMD Athlon 64 2 ГГц и 512 Мб оперативной памяти. Узлы связаны сетью Fast Ethernet. Кластер функционирует на базе операционной системы Microsoft Compute Cluster Server 2003 SP1, на каждом узле установлена среда выполнения MPI.NET Runtime версии 1.0.

6. Заключение

В статье были предложены три стратегии распараллеливания методов обучения машин опорных векторов SVM – *Предварительное обучение*, *Турнир* и *Гонки*. Дана теоретическая временная оценка, показывающая, что ускорение всех стратегий существенно зависит от количества опорных векторов в обучающих данных, причем ускорение в стратегии *Предварительное обучение* несколько превосходит ускорение в стратегии *Турнир* для систем с распределенной памятью.

Отметим, что возможна реализация комбинированных стратегий, например, стратегия *Гонки* сочетается с *Турниром*, т. е. каждым участником *Гонок* является не один процессор, а целая группа, которая реализует стратегию *Турнирного отбора*.

Целью дальнейших исследований является, во-первых, проведение вычислительных экспериментов, во-вторых, реализация идеи иерархического распараллеливания процесса обучения машин опорных векторов в задачах многоклассовой классификации.

Литература

1. Vapnik V. Statistical learning theory. Wiley, New York, 1998.
2. Platt J. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines // Advances in Kernel Methods Support Vector Machine, MIT Press, Cambridge, 1999. Pp. 185–208.
3. Boser, B. E., Guyon, I. M., Vapnik, V., A Training Algorithm for Optimal Margin Classifiers, Fifth Annual Workshop on Computational Learning Theory, ACM, 1992.
4. Grama A., Gupta A., Karypis G., Kumar V. Introduction to Parallel Computing. Second Edition. Addison-Wesley, 2003.
5. Project MPI.NET // The Open Systems Lab, Indiana University. http://www.osl.iu.edu/research/mpi.net.