Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Системного Программирования

Гарифуллин Шамиль Раифович

Исследование и сравнение алгоритмов SVM и TSVM для задач классификации

Курсовая работа

Научный руководитель:
аспирант кафедры Системного Программирования
Невоструев Константин

Санкт-Петербург
2014

Оглавление

Введение

Задача классификации или, как ее формулируют математически, задача агрегирования элементов произвольной природы, имеет обширную область практических применений в автоматике, управлении, экономике, социологии, медицине, геологии, астрономии, ядерной физике и т.д. Решением задачи агрегирования является такое разбиение множества анализируемых элементов на непересекающиеся подмножества (блоки, агрегаты, классы), в которых содержатся только сходные, близкие друг к другу в некотором, возможно неизвестном, но объективно существующем отношении.

Задача классификации — одна из наиболее распространенных задач в анализе данных и распознавании образов. 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. Из самых важных и общеизвестных является Метод Опорных Векторов (Support Vector Machines, SVM)[3]. Одним из, пожалуй самых больших, его недостатков является неиспользование информации входных данных у которых значения классов неизвестны. Увидел этот недостаток и сам создатель алгоритма — Владимир Вапник, им был предложен принципиально новый метод классификации который он назвал трансдуктивным (в противоположность стандартному индуктивному способу предсказания), так появилась Трансдуктивная SVM (TSVM)[7].

В данной работе были изучены алгоритмы SVM и TSVM, проведено сравнительное тестирование этих алгоритмов в применении к различным входным данным.

1. Постановка задачи

В рамках данной курсовой работы ставились следующие задачи:

- Реализовать алгоритмы построения Support Vector Machines (SVM) и Transductive Support Vector Machines (TSVM) для задач классификации
- Провести сравнительное тестирование алгоритмов

Основными критериями тестирования являлись:

- Качество обучения (точность и F1-score[5])
- Время, затраченное на обучение алгоритмов

Отметим, что тестирование должно проводиться на специальных наборах данных. Таких наборов в открытом доступе немного. Более того, данные в существующих наборах хранятся в различных форматах. В связи с этим возникает дополнительная задача: подготовить тестовые данные, приведя их к единому формату хранения и представления.

2. Обзор существующих алгоритмов

Как **Support Vector Machine**, так и **Transductive Support Vector Machine** реализуют идею разделителя с максимальным зазором. Математическая формулировка этих идей приводит к задаче квадратичного программирования, решение которой позволяет построить классификатор для входных данных.

В теории, решить такую задачу можно стандартными методами квадратичного программирования. На практике же оказывается, что данные методы неприменимы из-за ограничений по памяти. Квадратичная форма для данной задачи использует матрицу, в которой число элементов равно квадрату от числа объектов во входных данных. Такая матрица не умещается в 4 ГБайта в случае, если размерности матриц больше 4000*4000.

Используя специфику данных методов, были разработаны алгоритмы, обходящие это ограничение. Среди них **Sequential Minimal Optimization** (SMO[8]). Который, в конечном счете, и был использован в реализации алгоритмов.

В качестве альтернативы данным алгоритмам, из самых популярных можно выделить **Нейронные Сети**[11], **GBM**[6] и **Случайные Леса**[2].

Нейронные сети

являются, пожалуй, самыми старыми из перечисленных алгоритмов (первые варианты появились в 1971). Название получили потому, что утверждалось, что именно так работает мозг человека. В их основе лежит задача подбора матрицы коэффициентов линейного преобразования между входными и выходными данными. Существуют различные нелинейные модификации этого преобразования, за счет которых и добиваются высокого качества обучения. Основной трудностью является подбор размерностей матриц преобразования и настройка множества параметров, характеризующих нелинейные модификации алгоритмов. Считались лучшим алгоритмом классификации до появления SVM.

Случайные леса

появились сравнительно недавно (2001) и стали массово популярны, в основном из-за высокой скорости обучения и простой интерпретируемости (чего нельзя сказать о Нейросетях). В основе лежат **Деревья Решений**[9], сравнительно старый алгоритм (1986), которые являются по факту простыми ‘деревьями’ вопросов, с ответами ‘да - нет’, которые и приводят к конечному ответу. В их основе лежит обычный ‘жадный’ алгоритм поэтому они строятся очень быстро, но не дают результатов, сравнимых с SVM или Нейросетями.

Однако, метод предложенный Лео Брейманом — строить тысячи деревьев и брать моду их ответа — даёт очень хорошие результаты, сравнимые, а иногда и превосходящие как SVM, так и Нейросети, при этом не нужно особо мучаться с подбором параметров, т.к. результат не может настолько сильно меняться от их выбора, как в предыдущих двух алгоритмах. И, что самое интересное, они работают гораздо быстрее как SVM, так и Нейросетей. Являются де-факто первым выбором при решении задач в сообществе Machine Learning.

GBM,

или Gradient Boosting Machine (2000), в своей основе тоже использует Деревья Решений, но несколько иначе: обучает одно Дерево, потом использует второе Дерево, чтобы оценить градиент целевой функции (насколько алгоритм ошибся), потом добавляет это дерево к классификатору и так далее для всех Деревьев. Этот процесс занимает некоторое время, поэтому обучение идет сравнительно долго. Однако, считается лучшим классификаторам на данный момент, т.к. по качеству обучения превосходит все перечисленные алгоритмы и методы.

3. Описание идей алгоритмов SVM и TSVM

Так как метод TSVM является своего рода расширением метода SVM, начнем с описания SVM.

Рассмотрим следующее изображение, на котором крестиками обозначены положительные объекты, а кружочками — отрицательные объекты. Также изображен разделитель (здесь это линия, задаваемая уравнением $\theta^T x = 0$, также ее называют **разделяющей гиперплоскостью**) и 3 точки, обозначенные A, B и C.

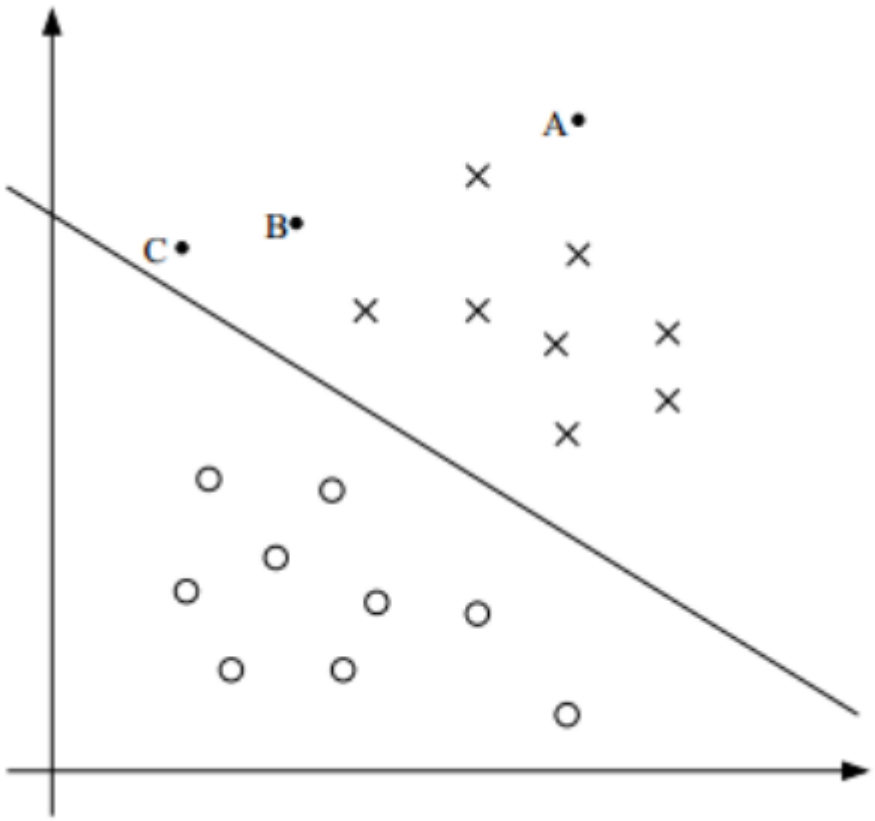

Рис. 1: Разделяющая гиперплоскость

Именно построением разделяющей гиперплоскости с максимальным зазором и занимаются оба метода.

3.1. Математическая формализация идей

Математическая формализация построения гиперплоскости с максимальным зазором для SVM выглядит так:

$$\min_{\mathbf{w},\xi^s} \quad F^s(\mathbf{w}) = \quad \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{l}\sum_{i=1}^{l}\xi_i^s \tag{1}$$

Где $\xi_i^s = \xi(\mathbf{w}, \mathbf{x}_i^s, y_i^s)$ элемент функции потерь[17] и $\{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^l$ множество маркированных объектов.

$\lambda > 0$ параметр регуляризации[16]. Метка s означает 'маркированные'; мы будем использовать метку u для обозначения немаркированных объектов.

В Трансдуктивном методе, как уже упоминалось, мы используем множество немаркированных объектов, $\{\mathbf{x}_i^u\}_{i=1}^n$ и включаем определение класса этих объектов в процесс обучения:

$$\min_{\mathbf{w}, \mathbf{y}^u} \quad F^s(\mathbf{w}) + \frac{C^u}{n} \sum_{i=1}^{n} \xi_i^u \tag{2}$$

$$\text{так, что} \quad \sum_{i=1}^{n} \delta(y, y_i^u) = n(y) \quad \forall y \tag{3}$$

где $\mathbf{y}^u = \{y_i^u\}$, $\xi_i^u = \xi(\mathbf{w}, \mathbf{x}_i^u, y_i^u)$ и δ дельта Кронекера[15].

C^u параметр регуляризации для немаркированной части.

(3) состоит из ограничений на метки, который приходят от экспертных знаний в области поставленной задачи.

Без этих ограничений TSVM обычно назначает класс большинства всем объектам немаркированной выборки.

3.2. Подходы к оптимизации

Обычно выделяют два подхода: непрерывный и комбинаторный.

Комбинаторный: Обычно множество меток $\mathbf{y}^u$ определяется одновременно с $\mathbf{w}$. Часто используют последовательность сменных шагов оптимизации (фиксируем $\mathbf{y}^u$ и получаем решение для $\mathbf{w}$, потом фиксируем $\mathbf{w}$ и получаем решение для $\mathbf{y}^u$) чтобы найти решение.
Важным является следующий момент: каждая подзадача уже решается простыми и/или стандартными методами.

Непрерывный: Тут уже сразу избавляемся от $\mathbf{y}^u$ и потом решаем **невыпуклую** задачу оптимизации в $\mathbf{w}$ путем минимизации следующей функции:

$$F^s(\mathbf{w}) + \frac{C^u}{n} \sum_{i=1}^{n} \rho(\mathbf{w}, \mathbf{x}_i^u) \tag{4}$$

где $\rho(\mathbf{w}, \mathbf{x}_i^u) = \min_{y^u} \xi(\mathbf{w}, \mathbf{x}_i^u, y_i^u)$. Обычно ξ и ρ сглаживаются, чтобы чтобы функция (4) могла быть дифференциируема и мы могли применить стандартные градиентные методы. Также ограничения в (3), включающие $\mathbf{y}^u$ заменяются гладкими ограничениями на $\mathbf{w}$.

Однако для повышения эффективности и скорости обучения был выбран эври-

стический метод обучения, в основе которого лежит алгоритм смены меток [12] для TSVM.

3.3. Описание алгоритма оптимизации TSVM

Algorithm 1 *TSVM*

1: Решаем проблему SVM, (1) чтобы получить $\mathbf{w}$.
2: Задаем начальные метки для немаркированных объектов, $\mathbf{y}^u$ используя шаги 3-9.
3: Задаем $Y = \{y\}$, множество всех классов, $A_y = \emptyset \ \forall y$, и $I = \{1, \ldots, n\}$.
4: **repeat**
5: $S_i = \max_{y \in Y} \mathbf{w}^T \mathbf{f}(y; \mathbf{x}_i^u)$ и $y_i = \arg\max_{y \in Y} \mathbf{w}^T \mathbf{f}(y; \mathbf{x}_i^u) \ \forall i \in I$.
6: Сортируем I уменьшая порядок S_i.
7: По порядку назначить i к A_{y_i} пока не превзошли размеры, заданные $n(y_i)$.
8: Удаляем все выделенные i из I и удаляем все перенасыщенные y (т.е. $|A_y| = n(y)$) из Y.
9: **until** $Y = \emptyset$
10: **for** $C^u = \{10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, \ldots, 1\}$ (in that order) **do**
11: **repeat**
12: Решаем (2) находим $\mathbf{w}$ при фиксированных $\mathbf{y}^u$.
13: Решаем (2)-(3) находим $\mathbf{y}^u$ при фиксированных $\mathbf{w}$.
14: **until** шаг 13 не меняет $\mathbf{y}^u$
15: **end for**

Алгоритм 1. Состоит из инициализации(шаги 1-9) $\mathbf{w}$ и $\mathbf{y}^u$, потом идет итеративная часть(шаги 10-15) в которой $\mathbf{w}$ и $\mathbf{y}^u$ уточняются методом Трансдукции. Можно доказать[7, 12], что Алгоритм 1 сходится.

Инициализация $\mathbf{w}$ происходит путем решения задачи SVM, без использования немаркированных объектов. Потом можно использовать $\mathbf{w}$, чтобы предсказать $\mathbf{y}^u$. Однако, обычно $\mathbf{y}^u$ не вписывается в ограничания из (3). Чтобы выбрать $\mathbf{y}^u$, которые все-таки подходят (3), мы делаем 'жадную' модификацию предсказанных $\mathbf{y}^u$. (Шаги 3-9 раскрывают детали)

Итеративная часть алгоритма состоит из внешего и внутреннего циклов. Во внешнем цикле (шаги 10-15) параметр регуляризации C^u варьируется от малых до заданного изначально параметра , используя шаги алгоритма Имитации Отжига[14]. Это делается для предотвращения резких смен меток $\mathbf{y}^u$, что позволяет достичь лучшего минимума (2)-(3) и, следовательно, лучшего качества обучения. Например, из 10 обучений на обучающей выборке[10] из 100 маркированных и $10,000$ немаркированных объектов, средний F-score достигнутый с помощью SVM, Алгоритма 1 без и Алгоритма 1 с Имитацией Отжига, составляют 0.4577, 0.5377 и 0.6253 соответственно.

Внутренний цикл (шаги 11-14) делает переменную оптимизацию $\mathbf{w}$ и $\mathbf{y}^u$ для фиксированного C^u. В шагах 12 и 13 мы используем самые новые $\mathbf{w}$ и $\mathbf{y}^u$ в качестве

начальных точек для соответствующих подзадач оптимизации. Это дает нам значительный выигрыш в скорости. В шаге 12 мы используем SMO[1].

[1]Стоит также отметить, что ключевая причина, по которой SMO является очень эффективным алгоритмом, заключается в том, что оптимизация идет по двум переменным и может быть выполнена очень быстро, аналитически, без привлечения громоздких методов квадратичного программирования[8].

4. Тестирование

4.1. Данные

Для проведения сравнительного тестирования алгоритмов построения SVM необходимы специальные данные, описанию которых посвящен данный раздел.

Тестовые наборы данных должны удовлетворять определенным требованиям. Прежде всего, эти данные должны описывать различные параметры некоторого объекта, причем каждый параметр должен описываться вещественным числом. Также для объекта должно быть указано, к какому классу он принадлежит.

Тестирование проводилось на 4 наборах данных[2], а именно:

- **Reuters**[10] (9947 параметров, 10 объектов обучающая, 600 тестовая выборки). Классификация какие из статей про корпорационные приобретения.
- **Titanic**[13] (6 параметров, 1526 объектов обучающая, 673 тестовая выборки). Классификация утонул пассажир или нет.
- **Diabetes**[4] (8 параметров, 576 объектов обучающая, 192 тестовая выборки). Распознавание больна ли пациентка диабетом.
- **Adult**[1] (153 параметра, 32561 объектов обучающая, 16281 тестовая выборки). Распознавание зарабатывает ли человек 50K в год.

4.2. Поиск наилучших параметров

В основе обучения алгоритмов лежит несколько параметров: ядро, параметры ядра (их от 0 до 3), cost. Также у TSVM есть параметр cost2, таким образом можно сказать, что TSVM тяжелее настроить.

Было решено выбрать линейное ядро, дабы не нужно было настраивать дополнительные параметры ядра. Cost являет-ся ‘штрафом’ за неверное предсказание объектов с классом и контролирует отношение между получением классификатора с максимальным зазором и отбрасыванием ‘шума’ в данных. Cost2 выполняет те же функции для объектов без начального класса. От этих параметров зависит скорость и качество обучения SVM и TSVM.

К сожалению, эти параметры нельзя получить аналитически, по какой-нибудь формуле. Их необходимо подбирать индивидуально для каждой задачи.

[2]Надо заметить что TSVM обучается как на тестовой так и на обучающей выборках сразу, мы просто не знаем классов тренировочной выборки

В данной работе подбор наилучших параметров осуществлялся по следующей схеме:

- Выбирается линейное ядро на основе того, что параметров достаточно много и можно предположить линейную разделимость. В качестве бонуса линейное ядро работает гораздо быстрее других ядер.

- Запускается тестирование на небольшой части от набора данных. Значения для обоих параметров выбираются из набора $\{0.01, 0.1, 1, 10, 50\}$. Обучение делается на каждой паре параметров. Всего 25 пар. Если расположить один параметр по оси X, а другой — по оси Y, то заметим, что параметры образуют сетку (с различными интервалами). В каждой точке этой сетки лежит комбинация параметров, каждой комбинации соответствует некоторый обученный классификатор.
  На этой сетке можно выделить прямоугольники, образованные близлежащими параметрами. Для каждого такого прямоугольника будем рассматривать 'точность прямоугольника' — сумма точностей классификаторов, соответствующих его вершинам. Выберем прямоугольник с наибольшей точностью. В силу того, что все его вершины обладают высоким качеством обучения, можно предположить, что где-то внутри этого прямоугольника находится комбинация параметров, на которой точность обучения будет еще больше. Поэтому дальнейший поиск продолжается внутри этого прямоугольника.

- Увеличивается часть данных, на которой запускается тестирование. Теперь обучение будет проводиться не только в вершинах полученного прямоугольника, но и в точках, лежащих между каждой парой исходных вершин. Итого получается 9 точек, образующих 4 уменьшенных прямоугольника. Как и на предыдущем этапе, выбирается прямоугольник с самой высокой точностью, и работа продолжается на нем.
  Теоретически, можно повторить второй этап еще некоторое количество раз, постепенно уменьшая площадь тестирования и увеличивая количество тестовых данных, на которых проходит обучение.

- Последний этап. Выбирается точка, лежащая в центре прямоугольника. Предположительно, комбинация параметров, соответствующая этой точке, должна давать самое высокое качество обучения. Поэтому данная комбинация параметров выбирается как наилучшая из всех возможных

4.3. Сравнение SVM и TSVM

Опираясь на описанную выше схему, для каждого набора данных были найдены оптимальные комбинации параметров.

Ниже приведены результаты обучения SVM с данными параметрами. Для каждого набора каждым алгоритмом обучение проводилось по 10 раз. Затем обученные классификаторы сортировались по качеству обучения. В таблицах приведены результаты медианных классификаторов (тех, которые оказались посередине в результате сортировки).

Таблица 1: Adult

	SVM	TSVM
Accuracy, %	85	86
F1-score	0.6	0.65
Time, s	286	22360

Таблица 2: Titanic

	SVM	TSVM
Accuracy, %	77	51
F1-score	0.57	0.61
Time, s	0.96	25.2

Таблица 3: Diabetes

	SVM	TSVM
Accuracy, %	75	67
F1-score	0.61	0.65
Time, s	0.43	3.5

Таблица 4: Reuters

	SVM	TSVM
Accuracy, %	84	96
F1-score	0.79	0.92
Time, s	0.45	16

Из полученных результатов можно сделать следующие выводы:

- Качество обучения классификатора не зависит от выбора алгоритма обучения при достаточно большом колиестве объектов с маркированными классами
- Обучение SVM выполняется значительно быстрее, чем обучение TSVM
- При малом количестве объектов SVM значительно уступает TSVM

Заключение

Алгоритмы машинного обучения, а именно различные виды SVM имеют широкую область прикладного применения: от задач медицинской диагностики до оптического распознавания символов. Как видно из данной работы, Трансдуктивный и Индуктивный алгоримы имеют свои преимущества и недостатки. Знание о границах и сферах применимости различных подходов может значительно сократить время на создание классификатора в ситуации, когда время критично. В случае, когда главной целью является качество, а не скорость обучения, знание об особенностях обучающих выборок помогает заранее отбросить самые низкокачественные варианты. Также при наличии очень малого количества маркированных объектов в обучающей выборке и некоторого, хоть мало-мальски значительного, множества немаркированных рекомендуется использовать TSVM в качестве метода классификации, т.к. именно там метод проявляет себя лучше всего по сравнению с SVM.

Список литературы

[1] Adult. — (Дата обращения 22.05.2014). URL: http://archive.ics.uci.edu/ml/datasets/Adult.

[2] Breiman Leo, Schapire E. Random forests. — 2001. — P. 5–32.

[3] Cortes Corinna, Vapnik Vladimir. Support-Vector Networks. — 1995. — P. 273–297.

[4] Diabetes. — (Дата обращения 22.05.2014). URL: http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

[5] F1-score. — (Дата обращения 22.05.2014). URL: http://en.wikipedia.org/wiki/F1_score.

[6] Friedman Jerome H. Greedy Function Approximation: A Gradient Boosting Machine // Annals of Statistics. — 2000. — Vol. 29. — P. 1189–1232.

[7] Joachims Thorsten. Transductive Inference for Text Classification using Support Vector Machines. — 1999. — P. 200–209.

[8] Platt John C. Sequential minimal optimization: A fast algorithm for training support vector machines. — 1999.

[9] Quinlan J. R. Induction of Decision Trees // MACH. LEARN. — 1986. — Vol. 1. — P. 81–106.

[10] Reuters. — (Дата обращения 22.05.2014). URL: http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[11] Siegelmann Hava T., Sontag Eduardo D. On The Computational Power Of Neural Nets // JOURNAL OF COMPUTER AND SYSTEM SCIENCES. — 1995. — Vol. 50, no. 1. — P. 132–150.

[12] Sindhwani Vikas, et al. Large Scale Semi-supervised Linear SVMs. — 2006.

[13] Titanic. — (Дата обращения 22.05.2014). URL: http://www.kaggle.com/c/titanic-gettingStarted/data.

[14] Алгоритм Имитации Отжига. — (Дата обращения 22.05.2014). URL: http://en.wikipedia.org/wiki/Simulated_annealing.

[15] Дельта-функция Кронекера. — (Дата обращения 22.05.2014). URL: http://en.wikipedia.org/wiki/Kronecker_delta.

[16] Регуляризация. — (Дата обращения 22.05.2014). URL: http://en.wikipedia.org/wiki/Regularization_(mathematics).

[17] Функция потерь. — (Дата обращения 22.05.2014). URL: http://en.wikipedia.org/wiki/Loss_function.