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

Атаманова Анна Михайловна

Исследование алгоритма SVM
Алгоритм ASM

Курсовая работа

Научный руководитель:
аспирант кафедры Системного Программирования
Невоструев Константин

Санкт-Петербург
2014 г.

Оглавление

Введение …… 3

Постановка задачи ……..…….…….. 5

SVM (Support Vector Machine) 6

ASM (Active Set Method) ……...……... 9

Тестирование ……..……11

Заключение ……13

Список литературы …….……. 14

Введение

“Машинное обучение - это процесс, в результате которого машина (компьютер) способна показывать поведение, которое в нее не было явно заложено (запрограммировано)”

A.L. Samuel
Some Studies in Machine Learning Using the Game of Checkers
IBM Journal. July 1959. P. 210–229.

Пример жизненной задачи:

Некоторое сложное заболевание лечится рисковой операцией, которая может закончиться не удачно. Явная зависимость не наблдается, однако есть набор данных о уже пройденных таких операциях (состояние больного и исход).Требуется на основе имеющихся данных по состоянию больного (набор каких-то признаков) определить исход операции - выживет/не выживет (принадлежность к одному из классов).

Более формально можно описать так:

Имеется некоторое множество объектов X и множество классов Y. Известно, что существует некоторое отображение $y : X \to Y$, сопоставляющее каждому объекту его класс. Более того, для некоторого конечного подмножества $X' \subset X$ значение $y(x)$ уже определено.

Требуется построить классификатор (отображение) $a : X \to Y$, “максимально приближающий” y на всем X.

Определять термин “максимального приближения” можно по-разному. Для этого в широком ряде случаев, задается функция потерь $L : X' \to \mathbb{R}$, которая определяет величину ошибки на каждом объекте X'.

Примеры функции потерь:

- $L(x) = [a(x) \neq y(x)]$ - индикатор ошибки
- $L(x) = |a(x) - y(x)|$ - абсолютное значение ошибки
- $L(x) = |a(x) - y(x)|^2$ - квадратичная ошибка

Далее, по этой функции, строится функция эмпирического риска $Q(a)$, которая уже действует из множества алгоритмов.

Примером функции эмпирического риска может быть следующая:

$$Q(a) = \frac{1}{|x'|} \sum_{x \in X'} L(x)$$

Далее, ограничивая себя некоторым семейством алгоритмов A, мы можем перейти к методу минимизации эмпирического риска. А именно, поиска $argmin_{a \in A} Q(a)$

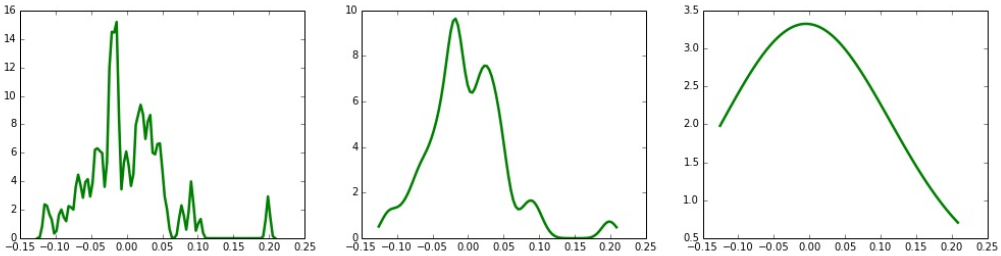
Для выбора семейства алгоритмов A, необходимо определить модель алгоритма, которая бы походила к конкретной задаче. Например, в случае $Y = \{1,-1\}$, $X = R^n$ и линейно-разделимых данных, моделью может служить гиперплоскость $< w,x > - b = 0$, где $w = (w_0, ..., w_{n-1}) \in R^{n-1}$. Тогда параметрами модели будет являться вектор $(w_0, ..., w_{n-1}, b)$. И именно по нему мы и будем минимизировать эмпирический риск.

Для проверки, на сколько выбранная модель алгоритмов соответствует данным, существуют следующие методы :

- LOO (Leave One Out). Выкидывает по одному элементу из выборки и сравниваемт насколько далеко истинное значение находится от значения на наилучшей модели в этом элементе.
- K-fold. Принцип тот же, но выкидывает уже по несколько элементов. (лучше работает на больших данных, хуже на маленьких)
- Cross-Validation. Первые два подхода, повторенные несколько раз

Так же, эти методы позволяют бороться с выбросами в данных и, так называемым, эффектом машинного переобучения, связанным с тем, что очень хорошее приближение на обучающей выборке (наше X'), может быть излишним на всей выборке X.

Этот эффект наглядно демонстрирует следующие графики



Еще один подход к борьбе с шумом или не точностью модели заключается в добавлении некоторого регуляризатора. Который, в том числе, может отвечать за то, насколько сильно мы можем отклоняться от заданной модели.

В случае поиска разделяющей гиперплоскости, мы можем добавить допущение того, что каждый элемент x_i обучающей выборки X' может вылезать из своего класса на некоторое расстояние ξ_i. При этом в целевую функцию добавив сумму этих штрафных слагаемых с каким-нибудь весом.

Такая задача, уже будет решаться для существенно большего класса

Постановка задачи

- реализация и исследование алгоритма на языке Python
- сравнение алгоритма с другими по скорости и качеству обучения

SVM (Support Vector Machine)

Пусть $X = R^n$, $X' \subset X$, $Y = \{-1, 1\}$. Перед нами задача двуклассовой классификации. Каждую компоненту элементов из X, удобно воспринимать, как некоторый признак объекта. Размерность объектов, тем самым, будет задавать количество определенных признаков. Более общий случай многоклассовой классификации можно потом будет свести к двуклассовой, например, в лоб поочередным решением задачи, лежит ли объект в $i-$м классе или нет. Длля SVM существует и более емкий способ, но об этом позднее.

Линейный SVM

Здесь, как и в примере выше, мы будем жить в предположении того, что входная выборка линейно-разделима. А именно, будем предполагать, что существует гиперплоскость $< w,x > - b = 0$, где $w = (w_0, ..., w_{n-1}) \in R^{n-1}$, которая четко разделяет пространство на два наших класса.

За функциею потерь L(x) возьмем расстояние от объекта x до ближайшего элемента $z \in X,$ такого, что $y(x) = a(z)$. (то есть расстояние до нашей гиперплоскости)

Таким образом,

$L(x) = 0$, при при $a(x) = y(x)$, то есть правильном определении класса

$L(x) = |< w,x >|$, при $a(x) \neq y(x)$, при ошибки классификатора.

Не умоляя общности, будем считать, что положительный класс - это та часть пространства, где $< w,x > + b \geq 0$, а отрицательный, где $< w,x > - b \leq 0$. (Границу пока будем определять к любому из классов)

Тогда функцию потерь можно переписать следующим образом:

$L(x) = 0$, при при $sign(< w,x > - b) = y(x)$

$L(x) = |< w,x >|$, при $sign(< w,x > - b) \neq y(x)$

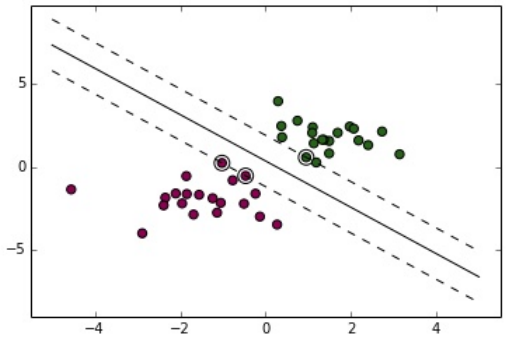
Заметим, что разделяющих гиперплоскостей может быть много. Мы же выберем из них ту, которая максимально отделяет один класс от другого, то есть максимизирует наименьшее расстояние между объектами разных классов.

По скольку параметр (w, b) задается с точностью до масштаба, можем его выбрать таким образом, чтоб $min_{x \in X'} y(x)(< w,x > - b) = 1$ на всей выборке X'.

Тогда разделяющей полосой будет

$\{x : -1 \leq y(x)(< w,x > - b) \leq 1\}$

Для определения ширины этой полосы, выберем два элемента x_+и x_- на противоположных границах



ширина = $< x_+ - x_-, \frac{w}{||w||} >= \frac{<x_+,w>}{||w||} - \frac{<x_-,w>}{||w||} = \frac{2}{||w||}$

Таким образом, для того чтобы максимизировать ширину зазора, надо минимизировать $||w||$

Добавляем, $||w||$ в качестве регуляризатора (и для удобства, к L(x) единичку, в случае непопадания) и получаем задачу SVM в безусловном виде.

$$\sum_{x \in X'} (1 - y(x)(< w, x > - b))_+ + \frac{1}{2C}||w||^2 \to min_{w,b}$$

С условиями она будет выглядеть так

$$||w||^2 \to min_{w,b}$$
$$s.t. \quad y(x_i)(< w, x_i > - b))_+ \geq 1 \ \forall x_i \in X'$$

Далее, как в примере страницей ранее, допускаем отсутствие четкого линейного разелителя.

$$\frac{1}{2}||w||^2 + C \sum_{x_i \in X'} \xi_i \to min_{w,b,\xi}$$
$$s.t. \quad y(x_i)(< w, x_i > - b))_+ \geq 1 - \xi_i \ \forall x_i \in X'$$

Применяем теорему Кунна-Таккера и переходим к двойственной задаче[1]:

$$-\sum_i \lambda_i + \frac{1}{2}\sum_j \sum_i \lambda_j \lambda_i y_j y_i < x_j, x_i > \ \to \ min_\lambda$$
$$s.t. \quad 0 \leq \lambda_i \leq C$$
$$\sum_i \lambda_i y_i = 0$$

Итак, мы свели задачу SVM к задаче квадратичного программирования, решив которую, мы сможем найти разделяющую гиперплоскость

$$w = \sum_i \lambda_i y_i x_i$$

[1] Чтобы не загромождать сильно формулы, обозначаю здесь yi=y(xi) и предполагаю, что i в каждой сумме ведется от 1 до |X'|

$$b = < w, x_i > - y_i$$

и алгоритм классификации

$$a(x) = \sum_i \lambda_i y_i < x_i, x >$$

Заметим, что теперь, вся информация о пространстве X, зашита в скалярном произведении, то есть, подобрав удачную функцию $K : X \times X \rightarrow \mathbb{R}$ (ядро), которая бы равнялась скалярному произведению в некотором другом пространстве H (то есть, было бы там линиейно разделимо), мы смогли бы классификатором $a(x) = \sum_i \lambda_i y_i K(x_i, x)$ разделить и наше. Более того, теперь нам даже не важно, какие признаки хранили за собой x, важно лишь уметь находить некоторое отношение между ними $K(x_i, x)$. По этому, SVM так же относят к безпризнаковым классификатором.

Примеры ядер:

$K(x_1, x_2) = < x_1, x_2 >$ - линейное ядро
$K(x_1, x_2) = (< x_1, x_2 > + 1)^d$ - полиномиальное ядро
$K(x_1, x_2) = \exp(-\frac{1}{2\sigma^2} < x_1, x_2 >)$ - RBF-ядро

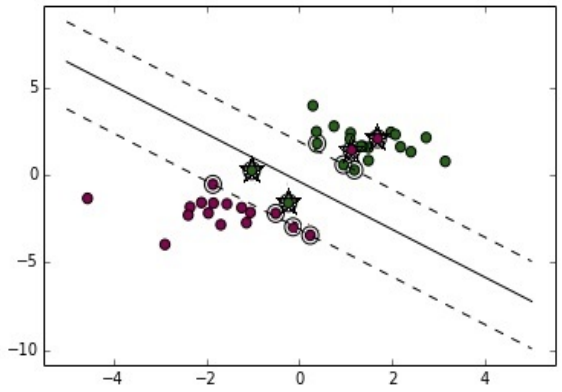
ASM (Active Set Method)

Пусть $m = |X'|$

$I = \{1, 2, ..., m\}$ - множество индексов объектов

$a(x)$ - некоторая разделяющая поверхность

Разобьем все объекты на три группы



- периферийные - те, которые определены правильно,
  $y(x_i)(< w, x_i > - b)) \geq 1$
  то есть $\lambda_i = 0$
- опорные - те, которые находятся на границе разделяющей поверхности ,
  $y(x_i)(< w, x_i > - b)) \geq 1$
  то есть $0 < \lambda_i < C$ (на картинке обведены кружочком)
- нарушители - те, которые оказались не в своем классе или в разделяющей полосе , $y(x_i)(< w, x_i > - b)) \leq 1$
  то есть $\lambda_i = C$ (звездочки на картинке)

Обозначим, соответственно, индексы этих групп I_o, I_s, I_c

Заметим, что периферийные объекты никак не влияют на классификатор (так как вес при них нулевой), а это значит, что добрую часть объектов, по идее, можно исключить, тем самым облегчив себе подсчеты.

В этом и заключается одна из основных идей алгоритма.

Второй полезной идеей является идея итеративного развития процесса. То есть, в случае прихода дополнительных данных после построения классификатора, мы все равно сможем их присоединить не пересчитывая все заново.

Алгоритм:

Пусть у нас уже есть какое-то разбиение индексов I и, возможно, дан алгоритм классификации $a(x) = \sum_{i \in I_s} \lambda_i y_i K(x_i, x)$ (если нет, то по умолчанию считаем, что все объекты определены правильно, то есть $I = I_s$, $\lambda = 0$)

Пробуем решить задачу квадратичного программирования только на опорных векторах. Если решения не существует, то плавно сдвигаем веса λ (по вектору, уменьшающему целевую функцию) до тех пор пока какой-то из λ_i ,не достигнет 0 или C.

Он перекидывается из опорных в периферийные или нарушители, соответственно. Далее пытаемся решить задачу снова.

Если решение все же существует, то находим самый дальний от границы объект из I_o, который классифицировался как объект-нарушитель. Находим самый дальний от границы объект из I_c, который классифицировался, как периферийный объект. И того, кто дальше, перекидываем в опорные (они являются как бы промежуточными между периферийными и нарушителями) и продолжаем все сначала. Если таких объектов не оказалось, значит существующее разбиение правильное, и мы построили оптимальный классификатор.

Итеративный процесс и работа только с фиксированным множеством объектов, в которое объекты приходят и уходят по одному, позволяет нам не решать заново всю задачу квадратичного программирования, а лишь пересчитывать кусочек с предыдущей итерации.[2]

Однако, обновлять матрицы и перекидывать объекты нужно аккуратно. В результате
изменения порядка этих действий, изначальная реализация, которая копила ошибку и давала менее точный результат, чем стандартный SVM, была усовершенствована до точности SVM.

[2] Более подробно алгоритм описывается тут [1]

Тестирование

В следующих таблицах приведено сравнение моей реализации ASM на языке Python и SVM из библиотеки sklearn, которая вызывает libSVM, написанную на C++.

По этому, останавливаться на сравнении времени долго не будем.

Все параметры выставляю одинаковые.

Сравниваемые метрики: precision, recall, f1_score - точность, полнота, f1-мера

$$precision = \frac{tp}{tp+fp}$$

$$recall = \frac{tp}{tp+fn}$$

$$f1\_score = \frac{2precision*recall}{precision + recall}$$

где

tp − число правильно определенных объектов класса 1

fp − число неправильно определенных объектов класса 1

fn − число неправильно определенных объектов класса -1

Набор данных Titanic:

Размерность объекта = 5

Размер тренировочной выборки = 1526

Размер тестовой выборки = 673

	Precision [0,1]	Recall [0,1]	F1_score [0,1]	Time, sec
SVM	0.88	0.52	0.66	0.002
ASM	0.8	0.59	0.68	3.037

Набор данных Adult:
Размерность объекта = 123
Размер тренировочной выборки = 300
Размер тестовой выборки = 16280

	Precision [0,1]	Recall [0,1]	F1_score [0,1]	Time, sec
SVM	0.74	0.84	0.80	0,04
ASM	0.79	0.65	0.71	9,51

Набор данных Diabetes:
Размерность объекта = 8
Размер тренировочной выборки =575
Размер тестовой выборки = 191

	Precision [0,1]	Recall [0,1]	F1_score [0,1]	Time, sec
SVM	0.75	0.64	0.69	0.02
ASM	0.75	0.72	0.74	95.6

К сожалению, в языке программирования Python очень тяжелые циклы.
А алгоритм ASM (точнее моя реализация этого алгоритма) активно их использует, В связи с этим, реальное сравнение скоростей алгоритмов произвести не удалось.

Полнота, точность и F-мера абсолютно идентична, что еще раз показывает нам, что мы решаем одну и ту же задачу. По этому все имеющиеся достоинства и недостатки в плане точности SVM можно с уверенностью приписывать и ASM.

Заключение

В ходе курсовой были изучены и разобраны основные методы машинного обучения для машины опорных векторов, реализован алгоритм ASM на языке программирования Python и усовершенствован до точности SVM (то есть теоретической точности) .

Это было протестировано и показано экспериментальным образом. Приведены результаты на трех наборах данных.

Так же были проведены оптимизации по времен алгоритма, которые дали некоторые улучшения, однако пока не совместимые со сравнением других, менее зависящих от подсчета на языке Python большого числа операций, алгоритмов.

Список литературы

[1] Машины опорных векторов. Невоструев Константин. 8 августа 2013 г.
[2] Katya Scheinberg. An Efficient Implementation of an Active Set Method for SVMs.
2005
[3] К. В. Воронцов. Лекции по методу опорных векторов. 2007
[4] http://school-wiki.yandex.ru/courses/spring2014/machinelearning1 Дата обращения: весна, 2014
[5] http://www.machinelearning.ru/wiki Дата обращения: весна, 2014
[6] http://scikit-learn.org/ Дата обращений: весна, 2014