САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Дудин Виктор Дмитриевич

Сравнение и тестирование реализаций алгоритма SVM для задач классификации

Курсовая работа

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

Санкт-Петербург
2013

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.[6]

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.[5]

На сегодняшний день разработано большое число подходов к решению задач классификации. Одним из них является Метод Опорных Векторов (Support Vector Machines, SVM). В данной работе были изучены различные алгоритмы построения SVM, проведено сравнительное тестирование этих алгоритмов в применении к различным входным данным. Также, с целью наглядного сравнения SVM с другими подходами, проводилось тестирование нескольких существующих реализаций других методов решения задачи классификации.

1. Постановка задачи

В рамках данной курсовой работы ставились следующие задачи:

- реализовать различные алгоритмы построения Support Vector Machines (SVM) для задач классификации;
- провести сравнительное тестирование алгоритмов построения SVM.

По каким критериям проводилось тестирование? Выделим ключевые показатели:

- качество обучения (точность обученного алгоритма);
- время, затраченное на обучение;
- скорость работы обученного алгоритма.

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.

Также, необходимо создать тестирующую систему для сравнения различных показателей исследуемых алгоритмов. Эта система должна быть разработана таким образом, чтобы в нее можно было без особых усилий интегрировать уже готовые библиотеки, реализующие SVM и прочие методы машинного обучения.

Помимо сравнения алгоритмов построения SVM между собой, необходимо провести тестирование нескольких библиотек, реализующих другие подходы для задачи классификации. В частности, необходимо сравнить SVM с такими методами, как Искусственные Нейронные Сети (Artificial Neural Networks, ANN) и Деревья Решений (Decision Trees). Данное тестирование должно показать лишь приблизительное сравнение между SVM и другими подходами к задаче классификации.

2. Обзор существующих алгоритмов

Подход Support Vector Machines реализует идею разделителя с максимальным зазором. Математическая формулировка этой идеи 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 128 МБайт в случае, если объектов больше 4000 [2].

Используя специфику данной матрицы, были разработаны алгоритмы, обходящие это ограничение. Среди них:

- Active Set Method;
- Sequential Minimal Optimization (SMO).

Данная курсовая работа посвящена сравнению этих алгоритмов.

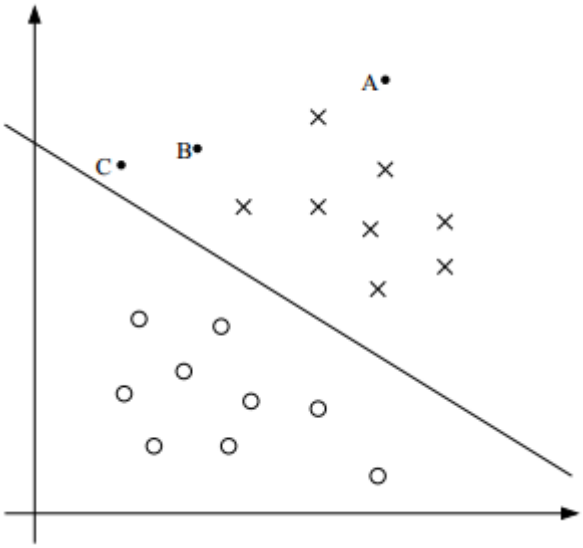
В качестве библиотек, реализующих другие подходы машинного обучения, были выбраны **Encog 3.1.0** для Artificial Neural Networks и **jaDTi 0.6.1** для Decision Trees. Напомним, что сравнение SVM и других подходов к решению задачи классификации не является главной целью данной курсовой. Данное сравнение призвано показать лишь приблизительное соотношение между SVM и другими методами. По этой причине указанные библиотеки выбирались почти произвольно, без какого-либо анализа в сфере существующих реализаций для Нейронных Сетей или Деревьев Решений. Единственным критерием выбора было топовое расположение в поисковых запросах Google.

3. SVM. Описание идеи

3.1. Зазор

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.

(Здесь θ, x - вещественные векторы).



Заметим, что точка A расположена очень далеко от разделителя. Если мы захотим предсказать класс в точке A, мы будем достаточно уверены, что точка A - положительна. Напротив, точка C расположена очень близко к разделителю, и хотя сейчас C определяется как положительная, при небольшом смещении разделяющей гиперплоскости может оказаться так, что C будет определяться как отрицательная. Таким образом, мы гораздо больше уверены в нашем предсказании для A, чем для C. Точка B лежит между этими двумя случаями, и, обобщая, мы видим, что чем дальше точка от разделителя, тем больше мы уверены в нашем предсказании относительно нее.

Таким образом, мы приходим к мысли, что было бы здорово по заданному тренировочному набору строить разделитель таким образом, чтобы все наши предсказания были не только точными, но и как можно более надежными (т.е. чтобы точки лежали как можно дальше от разделяющей гиперплоскости).

3.2. Функциональный зазор

Введем понятия функционального и геометрического зазоров.

Для тренировочного объекта $(x^{(i)}, y^{(i)})$ мы определим **функциональный зазор** от (w, b) как (1)

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b) \tag{1}$$

Здесь $x^{(i)}$ - набор параметров для (i)-го объекта, $y \in \{-1, 1\}$ - идентификатор положительного или отрицательного класса, w - вектор нормали к разделяющей гиперплоскости, b задает смещение w.

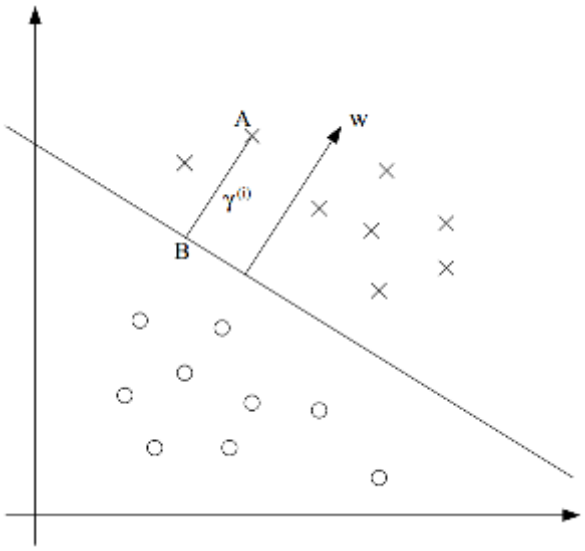
Заметим, что если $y^{(i)} = 1$, то для того, чтобы функциональный зазор был больше (т.е чтобы наше предсказание было корректным и надежным), необходимо, чтобы $w^T x + b$ было большим положительным числом. Напротив, если $y^{(i)} = -1$, то для большого функционального зазора необходимо, чтобы $w^T x + b$ было большим отрицательным числом. Более того, если $y^{(i)}(w^T x + b) > 0$, то наше предсказание для этого объекта корректно. Таким образом, большой функциональный зазор отражает достоверность и корректность предсказания.

Для тренировочного набора $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ определим функциональный зазор от (w, b) как наименьший из функциональных зазоров для его тренировочных объектов (2)

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)} \tag{2}$$

3.3. Геометрический зазор

Теперь поговорим о геометрическом зазоре. Рассмотрим следующую картинку:



Здесь изображен разделитель, соответствующий (w, b), вместе с вектором w. Заметим, что w ортогонален разделяющей гиперплоскости. Рассмотрим точку A, которая соответствует некоторому тренировочному объекту с набором параметров $x^{(i)}$ и меткой $y^{(i)} = 1$. Ее расстояние до разделителя, $\gamma^{(i)}$, задается отрезком AB.

Как найти значение $\gamma^{(i)}$? Поскольку A отображает $x^{(i)}$, точка B будет задаваться формулой $x^{(i)} - \gamma^{(i)} \cdot \frac{w}{\|w\|}$. Но точка B лежит на разделителе, а все точки x на разделяющей гиперплоскости удовлетворяют уравнению $w^T x + b = 0$. Отсюда (3)

$$w^T \left(x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|} \right) + b = 0 \tag{3}$$

Решая (3) для $\gamma^{(i)}$, получаем (4)

$$\gamma^{(i)} = \frac{w^T x^{(i)} + b}{\|w\|} = \left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \tag{4}$$

Данные формулы были выведены для положительного тренировочного объекта, расположенного в точке A. Обобщая, определим геометрический зазор от (w, b) для тренировочного объекта $(x^{(i)}, y^{(i)})$ как (5)

$$\gamma^{(i)} = y^{(i)} \left(\left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right) \tag{5}$$

Заметим, что в случае $\|w\| = 1$ функциональный зазор равен геометрическому зазору. Таким образом, это дает нам возможность связать два различных понятия зазора.

Наконец, для тренировочного набора $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ мы также определим геометрический зазор от (w, b) как наименьший из геометрических зазоров для его тренировочных объектов (6)

$$\gamma = \min_{i=1,\ldots,m} \gamma^{(i)} \tag{6}$$

3.4. Классификатор с оптимальным зазором

Опираясь на наши предыдущие рассуждения, попытаемся найти разделитель, который будет максимизировать (геометрический) зазор, поскольку это будет отображать очень достоверное множество предсказаний на тренировочном наборе. В частности, это даст нам разделяющую гиперплоскость, которая разделяет положительные и отрицательные объекты с "промежутком" (геометрическим зазором).

Сейчас будем рассматривать ситуацию, когда наш тренировочный набор линейно разделим (т.е. возможно разделить положительные и отрицательные объекты, используя некоторую разделяющую гиперплоскость). Как нам найти разделитель? Мы можем сформулировать следующую оптимизационную задачу:

$$\begin{aligned} \max_{\gamma,w,b} \quad & \gamma \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq \gamma, i = 1, \ldots, m \\ & \|w\| = 1 \end{aligned}$$

Т.е. мы хотим максимизировать γ при условии, что каждый тренировочный объект имеет функциональный зазор не меньше γ. Ограничение $\|w\| = 1$ гарантирует, что функциональный и геометрический зазоры совпадают, поэтому геометрический зазор также не меньше γ. Таким образом, решение этой задачи даст нам (w, b) с наибольшим геометрическим зазором для данного тренировочного набора.

Попытаемся избавиться от ограничения $\|w\| = 1$. Из-за этого ограничения теряется выпуклость множества определения, и задача становится нерешаемой стандартными методами оптимизации. Поэтому преобразуем задачу в более приятную форму:

$$\begin{aligned} \max_{\gamma,w,b} \quad & \frac{\hat{\gamma}}{\|w\|} \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq \hat{\gamma}, i = 1, \ldots, m \end{aligned}$$

Поскольку геометрический и функциональный зазоры связаны равенством $\gamma = \hat{\gamma}/\|w\|$, это преобразование корректно. Проблема в том, что наша новая целевая функция $\hat{\gamma}/\|w\|$ снова не выпуклая.

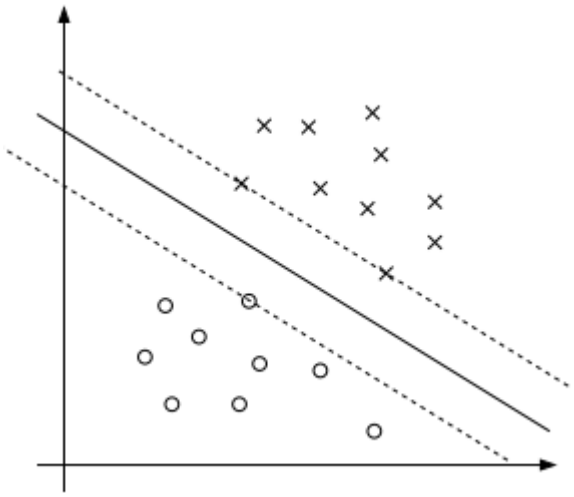
Сделаем следующий трюк. Используя тот факт, что домножение w и b на произвольную константу не меняет результат предсказания (важен только знак), подберем такую константу, при которой функциональный зазор для тренировочного набора равен 1:

$$\hat{\gamma} = 1$$

Заметим также, что максимизировать $\hat{\gamma}/\|w\| = 1/\|w\|$ - это то же самое, что минимизировать $\|w\|^2$. Опираясь на эти рассуждения, получим следующую оптимизационную задачу:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}\|w\|^2 \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, \ldots, m \end{aligned}$$

Это оптимизационная задача с выпуклой квадратичной целевой функцией и линейными ограничениями. Решение этой задачи даст нам **классификатор с оптимальным зазором**. Эта задача может быть решена с помощью стандартных методов оптимизации.



На картинке выше изображен разделитель с максимальным зазором (разделитель - сплошная линия). Точки с наименьшим зазором - те, которые лежат ближе всех к разделяющей гиперплоскости. На данной картинке это 3 точки (1 отрицательная и 2 положительные), которые лежат на пунктирных линиях, параллельных разделителю. Эти 3 точки называются **опорными векторами** (support vectors).

3.5. Нелинейный классификатор. Ядра

Мы научились строить классификатор для линейно разделимых множеств. Теперь покажем, как построить классификатор для любого множества, в том числе и линейно неразделимого.

Прежде всего, воспользуемся стандартными методами решения оптимизационных задач. Для нашей задачи построим Лагранжиан, после чего напишем двойственную форму для него. В результате этих преобразований получим следующую задачу:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ s.t. \quad & \alpha_i \geq 0, i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

У данной задачи дополнительно существуют условия Куна-Таккера, которые мы пока что опустим.

В ходе данных преобразований делается 2 важных замечания. Прежде всего, выясняется, что α_i, указанные выше, будут равны нулю на всех векторах, кроме опорных. Во-вторых, выводится новая форма записи разделителя:

$$w^T x + b = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

Здесь x - вектор, для которого хотим сделать предсказание, $x^{(i)}$ - тренировочный объект.

Исходя из сделанных замечаний, мы делаем вывод, что для предсказания класса некоторого объекта x нам достаточно посчитать скалярные произведения между этим вектором и опорными векторами, в то время как остальные тренировочные объекты никак не влияют на предсказание.

Для создания нелинейного разделителя используется следующая идея: давайте повысим размерность исходного пространства и построим разделяющую гиперплоскость в новом, "расширенном" пространстве. Разделитель, построенный в "расширенном" пространстве, будет линейным. Однако при отображении его обратно в "обычное" пространство разделитель станет нелинейным, чего мы и добивались.

Для реализации этой идея необходима некая нелинейная функция, с помощью которой мы будем отображать каждый вектор из исходного пространства в пространство более высокой размерности. Рассмотрим следующий пример. Допустим, размерность исходного пространства равна n. Отобразим векторы из этого пространства следующим образом:

$$\begin{aligned} z_1 &= x_1, \ldots, z_n = x_n \\ z_{n+1} &= x_1^2, \ldots, z_{2n} = x_n^2 \\ z_{2n+1} &= x_1 x_2, \ldots, z_{\frac{n(n+3)}{2}} = x_{n-1} x_n \end{aligned}$$

Скалярное произведение в новом пространстве будет обычным евклидовым, как и раньше. Следовательлно, существует функция двух переменных $K(x^{(1)}, x^{(2)}) = (z^{(1)} \cdot z^{(2)})$. В таком случае, нет необходимости переводить все векторы в новое пространство, достаточно лишь найти подходящую функцию $K(\cdot, \cdot)$. Тогда задача будет выглядеть

следующим образом:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K(x^{(i)}, x^{(j)}) \\ s.t. \quad & \alpha_i \geq 0, i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

Описанную выше функция K принято называть **ядром**. Стоит отметить, что не всякая функция может быть ядром. По теореме Мерсера, ядром может быть симметричная положительно определенная функция $K(x, y)$, удовлетворяющая следующему условию:

$$\int K(x, y) z(x) z(y) dx dy \geq 0$$

для всех функций $z(x)$ таких, что:

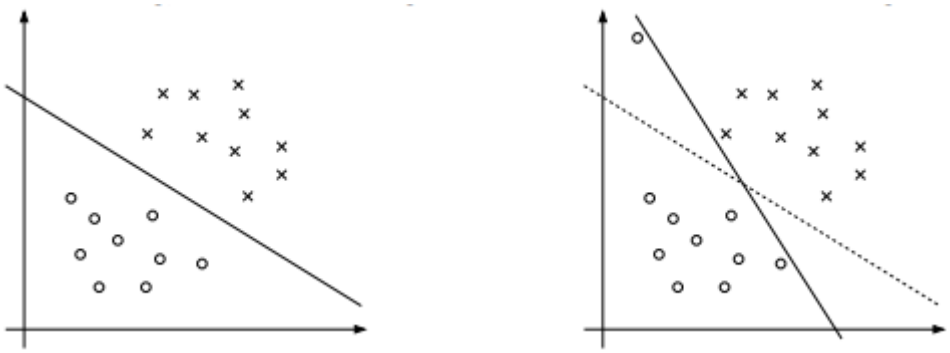
$$\int z^2(x) dx < \infty$$

Примером такой функции может быть широко используемая Гауссова функция[4]:

$$K(x, y) = exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

3.6. Регуляризация. Случай линейной неразделимости

Хотя вероятность того, что в "расширенном" пространстве данные будут линейно разделимы, велика, мы не можем гарантировать, что это всегда будет так. Более того, в некоторых случаях нахождение разделяющей гиперплоскости - это не совсем то, что мы хотим сделать, поскольку такой разделитель будем восприимчив к "шуму". Рассмотрим это на примере:



Чтобы сделать наш разделитель менее чувствительным к шуму, мы модифицируем нашу исходную оптимизационную задачу:

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, i = 1, \ldots, m \\ & \xi_i \geq 0, i = 1, \ldots, m \end{aligned}$$

Таким образом, объектам запрещено иметь функциональный зазор меньше 1, а за объекты, чей функциональный зазор равен $1-\xi_i$, мы должны увеличить целевую функцию на $C\xi_i$. Параметр C контролирует отношение между 2 задачами: увеличить целевую функция и гарантировать, что у большинства объектов функциональный зазор не меньше 1.

Как и раньше, напишем Лагранжиан для данной задачи, затем сформулируем для него двойственную задачу. В результате получим новую оптимизационную задачу:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j K(x^{(i)}, x^{(j)}) \\ s.t. \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned}$$

Кроме того, запишем условия Куна-Таккера:

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1 \end{aligned}$$

Теперь необходимо предложить алгоритм для решения полученной двойственной задачи. [1]

4. SVM. Active Set Method

Пусть у нас имеется выборка из n векторов. Введем матрицу Q размера $n \times n$ следующим образом. Пусть $Q_{ij} = y^{(i)}y^{(j)}K(x^{(i)}, x^{(j)})$. И введем вектор переменных s так, что $s_i = y^{(i)}\left(\sum_j \alpha_j y^{(j)} K(x^{(i)}, x^{(j)}) - b\right) - 1 + \xi_i$. Дополнительно обозначим за e единичный вектор. Тогда получим следующую оптимизационную задачу:

$$min_\alpha W(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$$

И условия для этой задачи:

$$\begin{aligned}
\alpha_i s_i &= 0 \\
(C - \alpha_i)\xi_i &= 0 \\
\sum_{i=1}^{n} \alpha_i y^{(i)} &= 0 \\
Q\alpha - by + \xi - s &= e \\
0 < \alpha_i &< C \\
s \geq 0, \xi &\leq 0
\end{aligned}$$

Введем дополнительные обозначения. Пусть $I = \{1, \ldots, n\}$ - множество индексов. И пусть I_0, I_s и I_c - подмножества I, для которых выполняется $\alpha_i = 0$, $0 < \alpha_i < C$ и $\alpha_i = C$ соответственно. Также, таким же образом обозначим подвектора и подматрицы по соответствующим индексам. Например, Q_{sc} или α_c.

Перейдем к описанию алгоритма.

Этап 1. Оптимизация выбранных векторов.

Если $|I_s| < 1$, перейти к этапу 2, иначе:

1. Решаем следующую задачу квадратичного программирования:

$$\begin{aligned}
&\frac{1}{2}\alpha_s^T Q_{ss}\alpha_s + C\alpha_s^T Q_{sc}e - e^T\alpha_s \to \min \\
&s.t. : y_s^T\alpha_s = -Cy_c^T e
\end{aligned}$$

Если решение существует и удовлетворяет условию, то переходим к этапу 2, иначе:

2. Находим направление, по которому происходит уменьшение значения целевой функции. Если задача имеет конечное значение, то направлением будет $d = \alpha_s^* - \alpha_s$, где α_s^* - решение задачи, иначе находим бесконечное направление d.

3. Далее, движемся по этому направлению, пока не появится индекс i такой, что $\alpha_i = 0$ или $\alpha_i = C$. Обозначаем его i^* и исключаем из множества I_s.

4. Если $\alpha_i^* = 0$, то добавляем его в I_0, иначе в I_c.

Этап 2. Выбор векторов для оптимизации.

На данном этапе мы работаем лишь с подмножеством множеств I_0 и I_c. Обозначим их как I_0' и I_c' соответственно.

1. Находим значения s_0' и ξ_c':

$$\begin{aligned} s_0' &= -Q_{0s}'\alpha_s - CQ_{0c}'e - \beta y_0' + e \\ \xi_c' &= Q_{cs}'\alpha_s + CQ_{cc}'e + \beta y_c' - e \end{aligned}$$

   Нужно понимать, что в матрицах Q_{0c}' и Q_{cc}' только первый индекс определяется множеством I_0' и I_c' соответственно. Второй индекс определяется множеством I_c.

2. Находим индексы i_0 и i_c:

$$\begin{aligned} i_0 &= argmin_i\{s_i : i \in I_0'\} \\ i_c &= argmin_i\{\xi_i : i \in I_c'\} \end{aligned}$$

   Если I_0' или I_c' пусто, то положим s_{i_0} или ξ_{i_c} равным нулю соответственно.

3. $s_{i_0} \geq 0$ и $\xi_{i_c} \geq 0$. Если $s \geq 0$ и $\xi \geq 0$, то алгоритм завершен, иначе выделим новые подмножества I_0' и I_c' и перейдем к началу этапа 2.

4. Если $s_{i_0} \leq \xi_{i_c}$, то удаляем i_0 из I_0 и добавляем его в I_s, иначе удаляем i_c из I_c и добавляем его в I_s. Переходим к этапу 1.[4]

5. SVM. Sequential Minimal Optimization

Прежде чем приступить к описанию SMO, расскажем про один из методов решения оптимизационных задач, называемый **продвижением по координатам** (Coordinate ascent). Покажем этот метод на примере. Рассмотрим следующую задачу:

$$\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m)$$

В данном примере W - это просто функция от параметров α_i, и она не имеет никакого отношения к SVM. Будем решать эту задачу следующим способом:

Loop until convergence {

For $i = 1, \ldots, m$\{

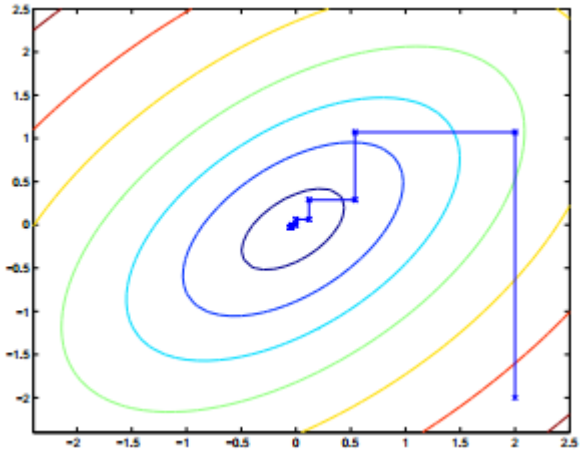
$\alpha_i := argmax_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$

}

}

Во внутреннем цикле этого алгоритма мы фиксируем все переменные, кроме некоторой α_i, и оптимизируем W только по отношению к переменной α_i. В представленном алгоритме оптимизация происходит в порядке $\alpha_1, \alpha_2, \ldots, \alpha_m, \alpha_1, \alpha_2, \ldots$. Данный порядок можно изменить, выбирая на каждом шаге ту переменную, от которой мы ожидаем получить наибольший прирост целевой функции.

В случае, когда функция $argmax$ во внутреннем цикле может быть выполнена быстро, метод продвижения по координатам становится весьма эффективным алгоритмом.



Вернемся к SVM. Вспомним полученную нами оптимизационную задачу:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K(x^{(i)}, x^{(j)}) \\ s.t. \quad & 0 \leq \alpha_i \leq C, i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

Попытаемся применить метод продвижения по координатам для решения данной задачи. Предположим, что мы уже нашли набор каких-то α_i-ых, удовлетворяющих условиям задачи. Заметим, что мы не сможем оптимизировать целевую функцию, зафиксировав все переменные, кроме одной. Объясним, почему. Допустим, мы зафиксировали все переменные, кроме первой. В этом случае, из ограничений задачи мы получим:

$$\alpha_1 y^{(1)} = -\sum_{i=2}^{m} \alpha_i y^{(i)}$$

Поскольку $y^{(1)} \in \{-1, 1\}$, получаем:

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{m} \alpha_i y^{(i)}$$

Таким образом, α_1 однозначно определяется остальными α_i-ыми.

Значит, если мы хотим увеличивать целевую функцию данным методом, мы должны изменять одновременно по крайней мере 2 переменные, чтобы не нарушать ограничений задачи. На этом основан алгоритм SMO, который коротко можно записать так:

Повторять до сходимости:

1. Выбрать 2 переменные α_i и α_j, по которым будем делать обновление (пытаемся выбрать такие переменные, изменение которых даст наибольший прирост $W(\alpha)$)

2. Оптимизировать $W(\alpha)$ по отношению к переменным α_i и α_j, зафиксировав остальные переменные.

Ключевая причина, по которой SMO является очень эффективным алгоритмом, заключается в том, что оптимизация по двум переменным может быть выполнена очень быстро, аналитически, без привлечения громоздких методов квадратичного программирования. [1]

6. Тестирование

6.1. 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. В данной работе тесты проводились только с 2-классовыми наборами данных.

Все тестовые наборы, использованные в этой работе, были взяты с сайта `http://archive.ics.uci.edu/ml/datasets/`. После скачивания, данные в наборах были нормализованы и усреднены. Указанная предварительная обработка данных способствовала увеличению скорости и качества обучения SVM.

Тестирование проводилось на 3 наборах данных, а именно:

- **Telescope** (10 параметров). Распознавание гамма/не-гамма сигналов.
- **Spambase** (57 параметров). Распознавание спамовых/не-спамовых писем.
- **Musk** (166 параметров). Распознавание мускусного/не-мускусного запаха.

6.2. Поиск наилучших параметров SVM

В основе обучения алгоритма SVM лежит 2 параметра: **sigma** и **cost**. **sigma** настраивает функцию ядра и отвечает за понятие "похожести" 2 объектов. **cost** является "штрафом" за неверное предсказание и контролирует отношение между получением классификатора с максимальным зазором и отбрасыванием "шума" в данных. От этих параметров зависит скорость и качество обучения SVM. К сожалению, эти параметры нельзя получить аналитически, по какой-нибудь формуле. Их необходимо подбирать индивидуально для каждой задачи.

В данной работе подбор наилучших параметров осуществлялся по следующей схеме:

- 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 SVM 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 SVM. Поэтому данная комбинация параметров выбирается как наилучшая из всех возможных.

6.3. Сравнение SVM

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

Набор	**sigma**	**cost**
Telescope	5.5	40
Spambase	20	40
Musk	20	40

Ниже приведены результаты обучения SVM с данными параметрами. Для каждого набора каждым алгоритмом обучение проводилось по 15 раз. Затем обученные классификаторы сортировались по качеству обучения. В таблицах приведены результаты медианных классификаторов (тех, которые оказались посередине в результате сортировки).

ASM - Active Set Method

SMO Sequential Minimal Optimization

Таблица 1: Результаты для набора **Telescope**

	ASM	SMO
Точность обучения, %	86.16	86.17
Время обучения, сек	123	141
Время на 1 предсказание, мсек	0.1809	0.1770

Таблица 2: Результаты для набора **Spambase**

	ASM	SMO
Точность обучения, %	92.03	92.03
Время обучения, сек	8	49
Время на 1 предсказание, мсек	0.0902	0.0889

Таблица 3: Результаты для набора **Musk**

	ASM	SMO
Точность обучения, %	97.49	97.49
Время обучения, сек	7	83
Время на 1 предсказание, мсек	0.1964	0.1977

Из полученных результатов можно сделать следующие выводы:

- Качество обучения классификатора не зависит от выбора алгоритма обучения
- Обучение с помощью Active Set Method выполняется значительно быстрее, чем с помощью Sequential Minimal Optimization. Особенно это проявляется на данных с большим числов параметром, где время обучение различается почти в 12 раз.

6.4. Деревья Решений и Нейронные Сети

Для поверхностного сравнения SVM с другими методами решения задач классификации, в данной работе были протестированы такие подходы, как Деревья Решений и Нейронные Сети.

Деревья решений

У деревьев решений, как и у SVM, есть 2 настраиваемых параметра. Для них была создана сетка, по аналогии с SVM, и проведено тестирование с подбором оптимальной комбинации параметров для каждого набора данных.

Приведем результаты тестирования Дерева Решений с оптимальными параметрами:

Таблица 4: Результаты тестирования **Дерева Решений**

	Telescope	Spambase	Musk
Точность обучения, %	83.85	89.48	93.26
Время обучения, сек	< 0.5	< 0.5	1
Время на 1 предсказание, мсек	0.0005	0.0010	0.0027

Коротко результаты тестирования можно описать так:

- Обучение с любыми комбинациями параметров длилось не дольше 1 секунды.
- Качество обучения ниже, чем у SVM, на 2.5 - 4 процента.
- Дерево решений выдает предсказания на 2-3 порядка быстрее, чем SVM.

Нейронные сети

У нейронных сетей есть большое количество настраиваемых параметров, и перебрать их все, как в случае с SVM или Деревьями Решений, очень затруднительно. В данной работе у нейронных сетей был всего 1 изменяемый параметр: количество нейронов на внутреннем слое. Тестирование проводилось только на сетях с одним внутренним слоем и с одинаковыми нейронами-сигмоидами.

На каждом тестовом наборе нейронные сети обучались, имея разное количество нейронов: 10, 15, 25, 40 и 60. Очевидно, что такое тестирование не могло привести нас к наилучшему результату, который возможно получить с помощью нейронных сетей. Но для поверхностного сравнительного тестирования такого подхода должно быть достаточно.

Приведем лучшие результаты тестирования нейронной сети для каждого набора:

Таблица 5: Результаты тестирования **Нейронной сети**

	Telescope	Spambase	Musk
Точность обучения, %	85.09	91.72	94.80
Время обучения, сек	12	3	2
Время на 1 предсказание, мсек	0.0054	0.0105	0.0062

Коротко результаты тестирования можно описать так:

- Обучение с любыми комбинациями параметров проходило от 2.7 до 10 раз быстрее, чем у SVM ASM.
- Качество обучения ниже, чем у SVM, на 0.3 - 2.7 процента.
- Скорость выдачи предсказания примерно на 1-2 порядка выше, чем у SVM.

Заключение

Алгоритмы машинного обучения имеют широкую область прикладного применения: от задач медицинской диагностики до оптического распознавания символов. Как видно из данной работы, каждый из алгоритмов машинного обучения имеет 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 другом случае, когда главной целью является качество, а не скорость обучения, знание об особенностях различных методов помогает заранее отбросить самые низкокачественные варианты.[3]

Список литературы

[1] Dr. Andrew Ng. Support Vector Machines. — Stanford : CS229, Machine Learning, Lecture Notes, 2012.

[2] John C. Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. — Microsoft Research, 1998.

[3] MachineLearning.ru. Классификация // MachineLearning.ru. — 2011. — http://www.machinelearning.ru/wiki/index.php?title=Классификация.

[4] Scheinberg Katya. An Efficient Implementation of an Active Set Method for SVMs. — IBM T. J. Watson Research Center, Mathematical Science Department, 2006.

[5] Гончаров Максим. Модифицированный древовидный алгоритм Байеса для решения задач классификации // Business Data Analytics. — 2007. — http://www.businessdataanalytics.ru/AugmentedNaiveBayes.htm.

[6] Малюков Н.Н. Лекции по прикладному анализу данных. Задача классификации и ее постановка. — http://prand.ru/content/zadacha-klassifikatsii-i-ee-postanovka.