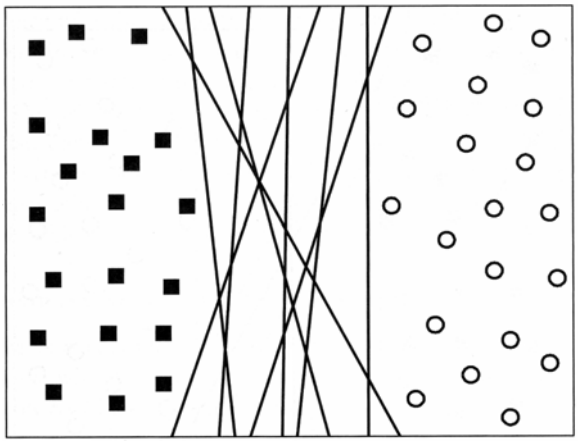
Лекция 16. Машина на поддържащи вектори

Машина на поддържащи вектори (Support Vector Machine – SVM) е в момента един от най-популярни подходи към машинно самообучение с учител при който не се използват никакви предварителни знания за проблемната област. Подходът води своето начало от статистическата теория за самообучение и показа много добри емпирични резултати в множество различни практически приложения – от разпознаване на ръкописен текст до категоризация на текстове. SVM работи много добре с данни с голяма размерност, като избягва „проклятието на размерността". Интересният аспект на подхода е, че за представяне на повърхнина на решение той използва само част от обучаващи примери – така наречени поддържащи вектори.

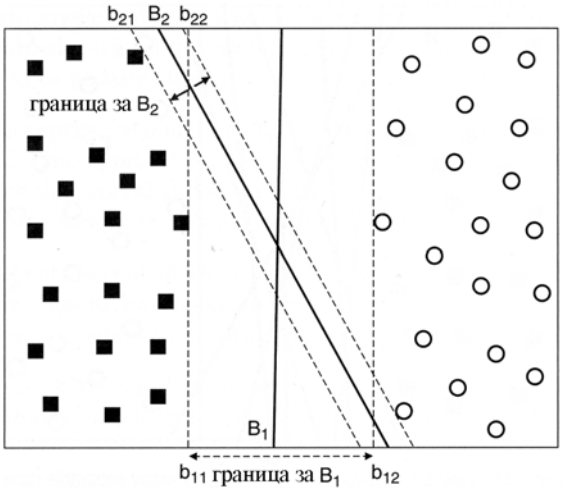
Преди да опишем идеята, стояща в основа на SVM, ще въведем понятие за *хиперравнина с максимално допустима граница* (maximal margin hyperplane) и обясним причини за избора на такава хиперравнина. След това ще покажем, как може да бъде обучена една линейна SVM, която търси такава хиперравнина в линейно сепарабелни данни. След това ще покажем, как този подход може да бъде обобщен за случая с нелинейно сепарабелни данни.

16.1. Хиперравнини с максимална допустима граница

На Фиг. 16-1 е графично показано едно множество данни, съдържащо примери от два класа, представени чрез квадратчета и кръгчета. Данните са линейно сепарабелни, т.е. може да се намери такава хиперравнина, че всички квадратчета се намират от едната й страна, а всички кръгчета – от другата. Обаче, както се вижда от рисунката, съществуват безкрайно много такива хиперравнини и макар че за всички тях грешката върху обучаващите данни (извадковата грешка) е равна на нула, няма никаква гаранция, че всички те ще имат еднакво добро класификационно поведение върху нови, невиждани до сега примери. Класификаторът трябва да избере една от тези хиперравнини, за да представи своята повърхнина на решения, базирайки се върху това, доколко добре се очаква те да класифицират тестови примери.



Фиг. 16-1. Възможни повърхнини на решения за едно множество линейно сепарабилни данни



Фиг. 16-2. Допустими граници на една повърхнина на решения

За да стане по-ясно, как изборът на различни хиперравнини влияе на класификационната грешка (т.е. грешка върху неизвестни данни), да разгледаме две различни повърхнини на решения B_1 и B_2, показани на Фиг. 16-2. И двете повърхнини на решения разделят обучаващите данни на съответните класове без никакви грешки. Всяка от тези повърхнини B_i е свързана с двойка хиперравнини, означени съответно с b_{i1} и b_{i2}. b_{i1} е получена чрез предвижване на една паралелна хиперравнина встрани от повърхнината на решение докато тя не докосне най-близкия квадрат (или квадрати), а b_{i2} е получена чрез предвижване на една паралелна хиперравнина встрани от повърхнината на решение докато тя не докосне най-близкото кръгче (или кръгчета). Разстоянието между тези две хиперравнини е известно като *допустимата граница на класификатора*. От фигурата се вижда, че допустимата граница за B_1 е значително по-голяма от тази за B_2. В този пример B_1 представлява хиперравнината с максимална допустима граница за зададени обучаващи примери.

16.1.1. Обосновка за избор на максимална допустима граница

Повърхнините на решения с голяма допустима граница имат тенденция да имат по-малки класификационни грешки (върху неизвестни данни) от тези с по-малки допустими граници. Интуитивно е ясно, че ако допустимата граница е малка, то всяко малко отклонение от повърхнината на решение може да окаже доста значимо влияние върху нейната възможност за класификация. Класификаторите, които произвеждат повърхнини на решения с малки допустими граници, са по-податливи към построяване на модели преспецифициращи данни и имат тенденция да правят лоши обобщения върху нови неизвестни примери.

Едно по-формално обяснение, свързващо допустимата граница на един линеен класификатор с неговата истинска класификационна грешка, се дава от принципа на статистическото самообучение известен като *минимизация на структурния риск* (*structural risk minimization – SRM*). Този принцип задава горната граница на истинската грешка на един классификатор (Err_G) в термините на неговата извадкова грешка (т.е. върху обучаващи данни – Err_T), броя на обучаващи примери (N) и сложността на използвания класификационен модел, известна като *капацитет (capacity – h)*. Принципът постановява, че с вероятност $1 - \delta$ истинската класификационна грешка на класификатора може да бъде в най-лошия случай:

$$Err_G \le Err_T + \varphi(\frac{h}{N}, \frac{\log(\delta)}{N}) \quad (16.1),$$

където φ е някоя монотонно нарастваща функция на капацитета h. Неравенството (16.1) много напомня на принципа на минимална дължина на описание (MDL), използващ за оценяване на хипотези. От тази гледна точка принципът SRM представлява един друг начин да се изрази истинската класификационна грешка като компромис между извадковата грешка и сложността на използвания класификационен модел. Капацитетът на един линеен модел на класификатора е обратно пропорционален на неговата допустима граница. Модели с малки допустими граници имат по-голям капацитет (т.е. сложност), тъй като са по-гъвкави и могат да се напасват към различни обучаващи множества в отличие от модели с големи допустими граници. Обаче съгласно SRM принципа при нарастване на капацитета границата на истинската класификационна грешка също нараства. Следователно е желателно да се построяват класификатори, които максимизират допустимите граници на техните повърхнини на

решения, за да осигурят, че техните истински класификационни грешки в най-лошия случай ще са минимални. Един такъв класификатор е линейната SVM, която е обяснено в следващия раздел.

16.2. Линейна SVM – сепарабелен случай

Една линейна SVM представлява класификатор, който търси хиперравнина с най-голяма допустима граница - по тази причина тя често се нарича *класификатор с максимална допустима граница*. За да разберем, как SVM научава такава повърхнина на решения, ще започнем с по-детайлно представяне на повърхнината на решения и допустимата граница на един линеен класификатор.

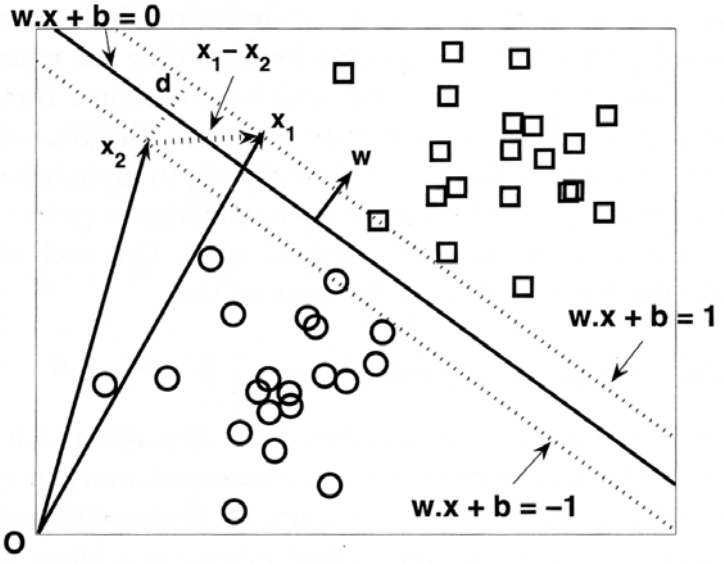
16.2.1. Линейна повърхнина на решения

Да разгледаме задача за двоична класификация, съдържаща N обучаващи примера. Всеки пример е означен като двойка $<\mathbf{x}_i, y_i>$ $(i = 1, 2, ..., N)$, където $\mathbf{x}_i = (x_{i1}, x_{i2}, ..., x_{id})^T$ съответства на атрибутните стойности на i-я пример. По съглашение ще означаваме с $y_i \in \{-1,1\}$ целевия атрибут. Повърхнината на решения на един линеен класификатор може да бъде записана във вида:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad (16.2),$$

където **w** и b са параметри на модела.

На Фиг. 16-3 е показано едно двумерно обучаващо множество, състоящо се от кръгчета и квадратчета. Една повърхнина на решения, която разделя обучаващите примери съгласно техните класове, е показана с плътна линия. Всеки пример, разположен върху повърхнината на решения, трябва да удовлетворява уравнение (16.2).



Фиг. 16-3. Повърхнина на решения и допустима граница на SVM

Например, ако $\mathbf{x}_a$ и $\mathbf{x}_b$ са две точки, разположени върху повърхнината на решения, то

$$\mathbf{w} \cdot \mathbf{x}_a + b = 0$$

$$\mathbf{w} \cdot \mathbf{x}_b + b = 0$$

Изваждайки едно уравнение от другото, получаваме:

$$\mathbf{w} \cdot (\mathbf{x}_b - \mathbf{x}_a) = 0 ,$$

където $\mathbf{x}_b - \mathbf{x}_a$ е вектор, паралелен на повърхнината на решения и е направен от $\mathbf{x}_a$ към $\mathbf{x}_b$. Тъй като скаларното произведение е равно на нула, направление на вектора **w** трябва да е перпендикулярно на повърхнината на решения, както е показано на Фиг. 16-3.

За всяко квадратче $\mathbf{x}_s$, разположено *над* повърхнината на решения, може да се покаже, че $\mathbf{w} \cdot \mathbf{x}_s + b = k$ *(16.3),*

където *k > 0*. Аналогично за всяко кръгче $\mathbf{x}_c$, разположено *под* повърхнината на решение, може да се покаже, че

$$\mathbf{w} \cdot \mathbf{x}_c + b = k' \quad (16.4),$$

където *k' < 0.* Ако означим всички квадратчета като принадлежащи към класа +1, а всички кръгчета – като принадлежащи към класа -1, то можем да предскажем стойността на целевия атрибут *y* на всеки тестов пример **z** в съответствие със следното правило:

$$y = \begin{cases} +1, & \text{ако } \mathbf{w} \cdot \mathbf{z} + b > 0 \\ -1, & \text{ако } \mathbf{w} \cdot \mathbf{z} + b < 0 \end{cases} \qquad (16.5)$$

16.2.2. Допустима граница на линеен класификатор

Да разгледаме квадратчето и кръгчето, които са най-близко да повърхнината на решения. Тъй като квадратчето е разположено над повърхнината, то трябва да удовлетворява уравнение (16.3) за някое положително *k*, докато кръгчето трябва да удовлетворява уравнение (16.4) за някое отрицателно *k'*. Можем да променим мащаб на параметрите **w** и *b* в уравнението на повърхнина на решения по такъв начин, че двете паралелни хиперравнини b_{i1} и b_{i2} да бъдат изразени по следния начин:

$$b_{i1} : \mathbf{w} \cdot \mathbf{x} + b = 1 \quad (16.6)$$

$$b_{i2} : \mathbf{w} \cdot \mathbf{x} + b = -1 \quad (16.7)$$

Допустимата граница на повърхнината на решения се задава чрез разстоянието между тези две хипарравнини. За да изчислим допустимата граница, да предположим, че $\mathbf{x}_1$ е точка, разположена на b_{i1}, а $\mathbf{x}_2$ е точка, разположена на b_{i2}, както е показано на Фиг. 16-3. Подставяйки тези точки в уравнения (16.6) и (16.7), допустимата граница *d* може да се изчисли чрез изваждане на второто уравнение от първото:

$$\mathbf{w} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 2$$

$$\| \mathbf{w} \| \times d = 2$$

$$d = \frac{2}{\| \mathbf{w} \|} \quad (16.8)$$

16.2.3. Научаване на модела на линейната SVM

Обучаващата фаза на SVM включва оценяване на параметрите **w** и *b* на повърхнината на решения чрез обучаващите данни. Параметрите трябва да бъдат избрани по такъв начин, че да удовлетворяват следните две условия:

$$\begin{aligned} &\mathbf{w}\cdot\mathbf{x}_i + b \geq 1 \quad ако\ y_i = 1 \\ &\mathbf{w}\cdot\mathbf{x}_i + b \leq -1 \quad ако\ y_i = -1 \end{aligned} \quad (16.9)$$

Тези условия налагат ограничения, че всички обучаващи примери от клас *y = 1* (т.е. квадратчета) трябва да са разположени върху или над хиперравнина $\mathbf{w}\cdot\mathbf{x} + b = 1$, докато примерите от клас *y = -1* (т.е. кръгчета) трябва да са разположени върху или под хиперравнина $\mathbf{w}\cdot\mathbf{x} + b = -1$. И двете неравенства могат да бъдат обобщени в следната по-компактна форма:

$$y_i(\mathbf{w}\cdot\mathbf{x} + b) \geq 1, \quad i = 1,2,...,N \quad (16.10)$$

Макар че тези условия са приложими към всеки линеен класификатор (включително и към персептрони), SVM налага едно допълнително изискване – допустимата граница на нейната повърхнина на решения трябва да е максимална. Максимизирането на допустимата граница, обаче, е еквивалентно на минимизацията на следната целева функция:

$$f(\mathbf{w}) = \frac{\| \mathbf{w} \|^2}{2} \quad (16.11)$$

Определение 1. Линейна SVM – сепарабелен случай. Задача за самообучение на SVM може да бъде формализирана като следната задача за оптимизация с ограничения:

$$\min_{\mathbf{w}} \frac{\| \mathbf{w} \|^2}{2}$$

при ограничения $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, \quad i = 1,2,...,N$

Тъй като целевата функция е квадратична, а ограниченията са линейни по отношение на параметрите **w** и *b*, то тази задача е известна като *задача за изпъкнала оптимизация* (*convex optimization problem*), която може да бъде решена чрез стандартен метод на *множители на Лагранж*. Накратко идеята на този метод се състои в следното.

Първо, трябва да препишем целевата функция във вида, който отчита наличните ограничения на търсените решения. Новата целева функция е известна под името *Лагранжиан* на оптимизационната задача:

$$L_P = \tfrac{1}{2} \| \mathbf{w} \|^2 - \sum_{i=1}^{N} \lambda_i (y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1) \quad (16.12),$$

където параметрите λ_i се наричат множители на Ланганж. Първият член на Лагранжиана е същия, като и оригиналната целева функция, докато вторият отразява ограничителните неравенства. За да разберем, защо целевата функция трябва да бъде модифицирана, да разгледаме оригиналната целева функция, задавана от ф-ла (16.11). Очевидно е, че тя има минимум при ||**w**|| = 0, т.е. при нулевия вектор, чиито компоненти са нули. Обаче това решение нарушава ограничения, зададени в Определение 1, тъй като при него няма приемливо решение за параметъра *b*. Решенията за **w** и *b* са *неприемливи*, ако те нарушават ограничителните неравенства, т.е. ако $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 < 0$. Лагранжианът, задаван от формула (16.12), включва в себе си тези ограничения чрез изваждане от члена, представящ оригиналната целева функция,

на линейната комбинация от тези ограничения. Подразбирайки, че $\lambda_i \geq 0$, става ясно че всяко неприемливо решение само увеличава стойността на Лагранжиана.

За минимизиране на Лагранжиана трябва да установим частните производни на L_P по **w** и b в нула:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \qquad (16.13)$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \lambda_i y_i = 0 \qquad (16.14)$$

Тъй като множителите на Лагранж са неизвестни, ние все още не можем да намерим решение за **w** и b. Ако Определение 1 съдържаше ограничения във вид на равенства вместо на неравенства, то можехме да използваме N уравнения от тези равенства заедно с уравнение (16.13) и (16.14), за да намерим приемливи решения за **w**, b и λ_i. Трябва да се подчертае, че когато ограниченията са във вид на равенства, множителите на Лагранж са свободни параметри, които могат да приемат произволни стойности.

Един от начините за работа с ограничения във вид на неравенства се състои в превръщането им в равенства. Това е възможно докато множителите на Лагранж са ограничени да бъдат неотрицателни числа. Подобната трансформация води до следните ограничения върху множители на Лагранж, известни като *условията на Каруш-Кун-Такер* (Karush-Kuhn-Tucker – KKT):

$$\lambda_i \geq 0 \qquad (16.15)$$

$$\lambda_i [y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0 \qquad (16.16)$$

На пръв поглед изглежда, че има толкова множители на Лагранж, колкото и обучаващи примери. Обаче може да се покаже, че повечето от множителите на Лагранж стават равни на нула след прилагането на ограничение, задавано от ф-ла (16.16). Това ограничение утвърждава, че множителят на Лагранж λ_i трябва да е равен на нула, освен в случаи, когато обучаващият пример $\mathbf{x}_i$ удовлетворява уравнението $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$. Такъв обучаващ пример, за когото $\lambda_i > 0$, лежи върху хиперравнина b_{i1} или b_{i2} и е известен като *поддържащ вектор*. За всички обучаващи примери, които не лежат върху тези две хиперравнини, $\lambda_i = 0$. Уравненията (16.15) и (16.16) също така утвърждават, че параметрите **w** и b, които определят повърхнината на решения, зависят само от поддържащите вектори.

И така, ние сведохме задача за оптимизацията, задавана от Определение 1, към задача за минимизиране на Лагранжиана (16.12) при ограничения (16.15) и (16.16). За съжаление търсенето на решение и на тази оптимизационна задача е доста сложна задача, тъй като тя съдържа голям брой параметри: **w**, b и λ_i. Обаче тя може да бъде опростена чрез трансформиране на Лагранжиана във функция, която има като аргументи само множителите на Лагранж – такава трансформирана задача е известна под името *двойствена (dual) задача*. За да стане това, отначало подставяме уравнения (16.13) и (16.14) в уравнение (16.12). Това ще доведе до следната двойствена формулировка на наша оптимизационна задача:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \qquad (16.17)$$

Ключовите разлики между този двойствен и оригиналния Лагранжиан се състоят в следното:

1. Двойственият Лагранжиан съдържа само множителите на Лагранж и обучаващите данни (при това само във вид на скаларното им произведение), докато оригиналният Лагранжиан включва не само множителите на Лагранж, но и параметрите на повърхнината на решения. Обаче решенията и на двете оптимизационни задачи са еквивалентни.
2. Квадратичният член в двойствения Лагранжиан има отрицателен знак, което означава, че първоначалната задача за минимизация на оригиналния Лагранжиан L_P, трябва да бъде превърната в задача за максимизация на двойствения Лагранжиан L_D.

За големи бази данни двойствената оптимизационна задача може да бъде решена чрез използване на числените техники от рода на квадратичното програмиране – детайлите на тези техники са извън обсега на тази лекция. След като множителите на Лагранж са намерени, можем да използваме уравнения (16.13) и (16.16) за да получим допустимите решения за **w** и b. По този начин повърхнината на решение може да бъде изразена по следния начин:

$$h(\mathbf{x}): \quad \left(\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{x}\right) + b = 0 \qquad (16.18)$$

Както вече е казано, b се получава чрез решаване на уравнение (16.16) за поддържащите вектори. Обаче, тъй като λ_i се изчисляват чрез числените методи и могат да имат грешки, изчислената стойност на b може да не бъде една и съща, а различна за различни поддържащи вектори, използвани в уравнение (16.16). По тази причина на практика като параметър на повърхнина на решение се използва средната стойност на b, изчислена за различни поддържащи вектори.

След като параметрите на повърхнината на решения са намерени, всеки тестов пример **z** се класифицира съгласно следната функция:

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \mathbf{z} + b) = sign\left(\sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i \cdot \mathbf{z} + b\right)$$

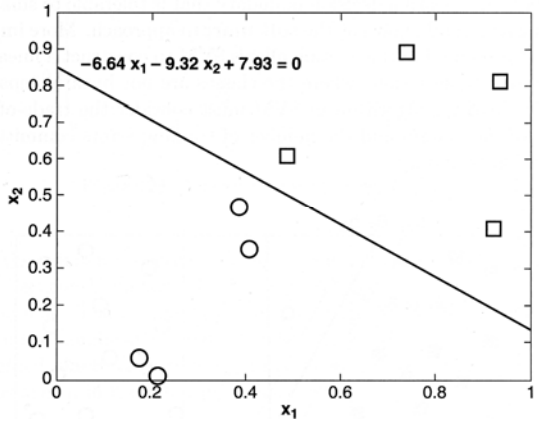
Ако $f(\mathbf{z}) = 1$, то тестовият пример се класифицира към положителния клас, в противен случай – като отрицателен.

Пример 1. Да разгледаме двумерни данни, показани на Фиг. 16-4 – те съдържат само 8 обучаващи примера. Чрез използване на квадратичното програмиране можем да решим оптимизационната задача, дефинирана от уравнение (16.17), за да получим стойностите на множителите на Лагранж за всеки обучаващ пример. Получените стойности са показани в последната колонка на Таблица 16-1. Обърнете внимание, че само първите два примера имат ненулеви стойности на множителите на Лагранж. Тези примери съответстват на поддържащите вектори в това множество данни.

Нека $\mathbf{w} = (w_1, w_2)$ и b са параметри на повърхнината на решения. Чрез използване на уравнението (16.13) можем да намерим w_1 и w_2 по следния начин:

$$w_1 = \sum_i \lambda_i y_i x_{i1} = 65.5621 \times 1 \times 0.3858 + 65.5621 \times (-1) \times 0.4871 = -6.64$$

$$w_2 = \sum_i \lambda_i y_i x_{i2} = 65.5621 \times 1 \times 0.4687 + 65.5621 \times (-1) \times 0.611 = -9.32$$



Фиг. 16-4. Пример на линейно сепарабелно множество

x_1	x_2	y	Множител на Лагранж
0.3858	0.4687	1	65.5261
0.4871	0.611	−1	65.5261
0.9218	0.4103	−1	0
0.7382	0.8936	−1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	−1	0
0.2146	0.0099	1	0

Таблица 16-1. Точки данни и съответните множители на Лагранж

Свободният член *b* може да се изчисли чрез използване на уравнения (16.16) за всеки от поддържащите вектори:

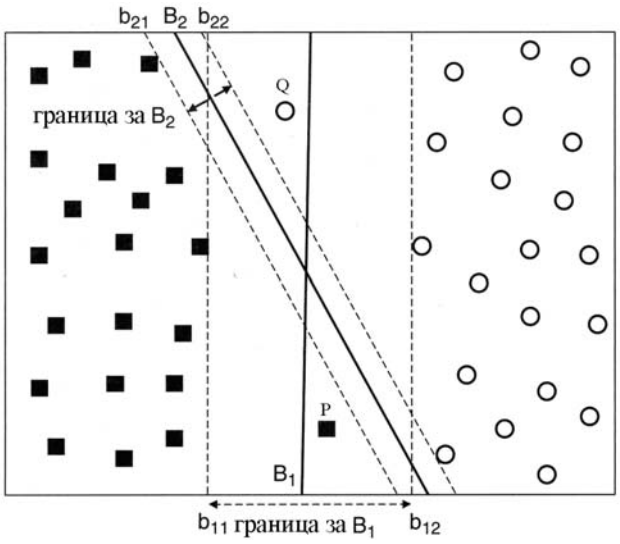
$$b^{(1)} = 1 - \mathbf{w} \cdot \mathbf{x}_1 = 1 - (-6.64)(0.3858) - (-9.32)(0.4687) = 7.9300$$

$$b^{(2)} = 1 - \mathbf{w} \cdot \mathbf{x}_2 = 1 - (6.64)(0.4871) - (-9.32)(0.611) = 7.9289$$

Осредняйки тези стойности получаваме $b = 7.93$. Повърхнината на решения, съответстваща на тези параметри, е показана на Фигура 16-4.

16.3. Линейна SVM – несепарабелен случай

На Фиг. 16-5 е показано множеството от данни, приличащо на тези от Фиг. 16-2 – единствената разлика е в наличието на два нови примера – *P* и *Q*. Макар че повърхнината на решения B_1 класифицира неправилно тези примери, а B_2 класифицира ги правилно, това още не означава, че B_2 е по-добра повърхнина на решения от B_1, тъй като тези нови примери могат да бъдат шум в данни. B_1 трябва да бъде предпочетена пред B_2, тъй като тя има по-голяма допустима граница и, следователно, не е толкова чувствителна към преспецифициране.



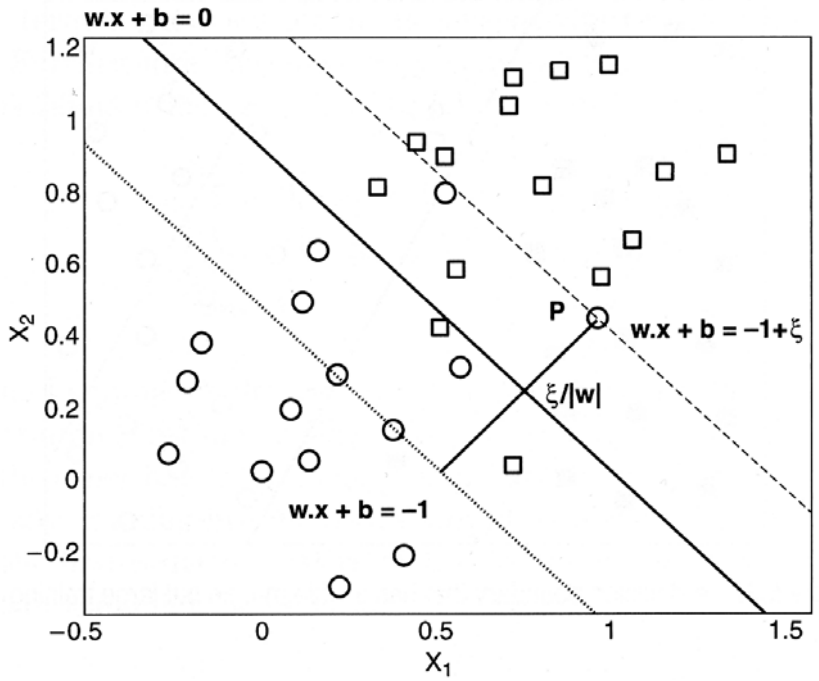
Фигура 16-5. Повърхнина на решения на SVM за несепарабелния случай

Обаче, формулировката на SVM, представена в предишния раздел, конструира само повърхнини на решения, които са свободни от грешки. В този раздел ще разгледаме как тази формулировка може да бъде модифицирана, за да позволява научаване на повърхнини на решения, които са толерантни към малки грешки при класифициране на обучаващите данни. Това се постига с метод, известен под името *подходът на мека (soft) допустима граница*. Което е по-важно, методът, представен в този раздел, позволява на SVM да конструира една линейна повърхнина на решения дори в ситуации, когато класове не са линейно сепарабелни. За да постигне това, обучаващият алгоритъм на SVM трябва да разглежда баланса между ширината на допустимата граница и броя на класификационни грешки при обучение, които се допускат от тази линейна повърхнина на решения.

Докато оригиналната целева функция, задавана от формула (16.11), продължава да е приложима и в този случай, повърхнината на решения B_1 вече не удовлетворява ограничения, задавани от формула (16.10). Следователно, тези ограничителни неравенства трябва да бъдат ослабени, за да се пригодят към линейно несепарабелните данни. Това нагаждане може да се направи чрез въвеждане на положителни *„отслабващи" (slack) променливи* (ξ) в ограниченията на оптимизационната задача, както е показано в следните уравнения:

$$\begin{aligned} &\mathbf{w}\cdot\mathbf{x}_i + b \geq 1-\xi_i \quad \text{ако } y_i = 1 \\ &\mathbf{w}\cdot\mathbf{x}_i + b \leq -1+\xi_i \quad \text{ако } y_i = -1 \end{aligned} \tag{16.19}$$

където $\forall i : \xi_i > 0$.

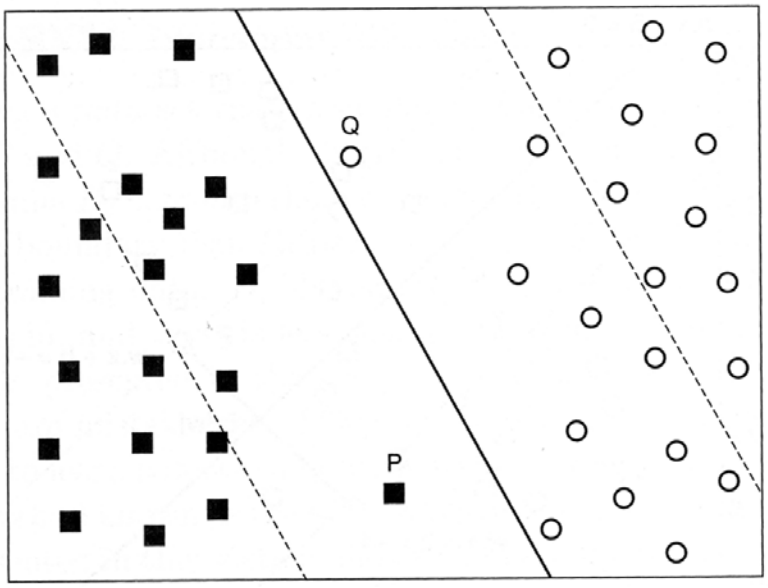


Фиг. 16-6. Отслабващи променливи за несепарабелни данни

За интерпретиране на смисъла на отслабващите променливи ξ_i, да разгледаме рисунка, представена на Фиг. 16-6. Кръгчето *P* е един от примерите, които нарушават ограничения, задавани от равенството (16.9). Нека $\mathbf{w}\cdot\mathbf{x}+b=-1+\xi$ означава линията, която е паралелна на повърхнината на решения и която минава чрез точка *P*. Може да се покаже, че разстоянието между тази линия и хиперравнината $\mathbf{w}\cdot\mathbf{x}+b=-1$ е $\xi/\|\mathbf{w}\|$.

Следователно ξ дава една оценка за класификационната грешка на повърхнината на решения върху обучаващия пример *P*.

По принцип можем да използваме същата целева функция както и по-рано и да използваме ограничения (16.19), за да намерим повърхнината на решения. Обаче, тъй като няма никакви ограничения върху броя на грешки, които може да направи повърхнината на решения, алгоритмът за самообучение може да намери една повърхнина на решения с много широка допустима граница, която неправилно класифицира много обучаващи примера (виж, например Фиг. 16-7).



Фиг. 16-7. Една повърхнина на решения, която има широка допустима граница, но голяма извадкова грешка

За да избегнем този проблем целевата функция трябва да се модифицира, за да наказва повърхнините на решения с големи стойности на отслабващи променливи. Модифицираната версия на целевата функция се задава от следната формула:

$$f(\mathbf{w}) = \frac{\| \mathbf{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \zeta_i \right)^k ,$$

където *C* и *k* са параметри, чиито стойности се определят от потребителя и които представляват наказанието за неправилното класифициране на обучаващите примери. За простота на изложението ще приемем, че *k = 1*. Параметър *C* може да бъде избран на базата на поведението на модела върху валидиращото множество данни.

Лагранжианът за тази оптимизационна задача с ограничения може да бъде записан във вида:

$$L_P = \frac{1}{2} \| \mathbf{w} \|^2 + C\sum_{i=1}^{N} \zeta_i - \sum_{i=1}^{N} \lambda_i \{ y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \zeta \} - \sum_{i=1}^{N} \mu_i \zeta_i \qquad (16.20)$$

където първите два члена са целевата функция, подлежаща на минимизиране, третият член съответства на ограничаващите неравенства, асоциирани с отслабващите променливи, а последният член е резултатът от изискванията за неотрицателност на променливите ξ_i. По-нататък, ограничителните неравенства могат да бъдат трансформирани в ограничителните равенства чрез използване на следните KKT условия:

$$\zeta_i \geq 0, \quad \lambda_i \geq 0, \quad \mu_i \geq 0 \qquad (16.21)$$

$$\lambda_i \{y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \zeta\} = 0 \qquad (16.22)$$

$$\mu_i \zeta_i = 0 \qquad (16.23)$$

Обърнете внимание, че множителите на Лагранж, задавани от уравнение (16.22), са ненулеви само, ако обучаващите примери лежат върху хиперравнини $\mathbf{w} \cdot \mathbf{x}_i + b = \pm 1$ или имат $\xi_i > 0$. От другата страна, множителите на Лагранж μ_i, задавани от уравнение (16.23), са ненулеви само за обучаващите примери, които са класифицирани неправилно (т.е. имащи $\xi_i > 0$).

Установявайки първите производни на Лагранжиана по отношение на **w**, b и ξ_i в нула, получаваме следните уравнения:

$$\frac{\partial L_P}{\partial w_j} = w_i - \sum_{i=1}^{N} \lambda_i y_i x_{ij} = 0 \quad \Rightarrow w_j = \sum_{i=1}^{N} \lambda_i y_i x_{ij} \qquad (16.24)$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{N} \lambda_i y_i = 0 \quad \Rightarrow \sum_{i=1}^{N} \lambda_i y_i = 0 \qquad (16.25)$$

$$\frac{\partial L_P}{\partial \zeta_i} = C - \lambda_i - \mu_i = 0 \quad \Rightarrow \lambda_i + \mu_i = C \qquad (16.26)$$

Подставяйки уравненията (16.24), (16.25) и (16.26) в Лагранжиан (16.20), получаваме следния двойствен Лагранжиан:

$$L_D = \frac{1}{2}\sum_{i.j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + C\sum_i \zeta_i - \sum_i \lambda_i \{y_i \left(\sum_j \lambda_j y_j \mathbf{x}_i \cdot \mathbf{x}_j + b \right) - 1 + \zeta_i\} - \sum_i (C - \lambda_i)\zeta_i$$

$$= \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \qquad (16.27)$$

Този двойствен Лагранжиан изглежда по същия начин, както и двойственият Лагранжиан за линейно сепарабилен случай (уравнение (16.17)), обаче ограниченията, налагани върху множителите на Лагранж λ_i, се отличават от тези за сепарабелния случай. При линейно сепарабелния случай множителите на Лагранж трябва да бъдат неотрицателни, т.е. $\lambda_i \geq 0$. При несепарабелният случай уравнението (16.26) утвърждава, че λ_i не трябва да превишават C (тъй като и μ_i и λ_i са неотрицателни). Следователно множителите на Лагранж за линейно несеперабелни данни са ограничени до диапазона $0 \leq \lambda_i \leq C$.

Описаната по-горе двойствена оптимизационна задача се решава числено чрез методи на квадратичното програмиране, за да се получат стойности на множителите на Лагранж λ_i. След това тези множители се подставят в уравнение (16.24) и в условията на ККТ, за да бъдат изчислени параметрите на повърхнината на решения.

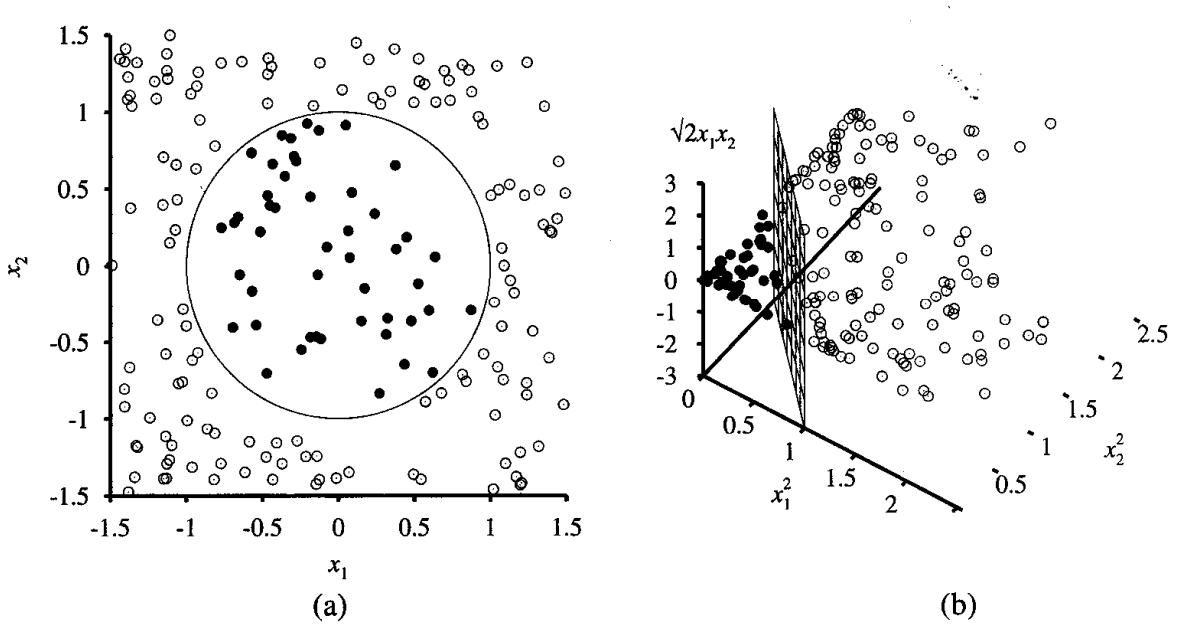
16.4. Нелинейна SVM

Формулировките на SVM, описани в предишните раздели, конструират една линейна повърхнина на решения, за да разделят обучаващите примери на два класа. В този

раздел ще разгледаме методология за прилагане на SVM към данни, които имат нелинейни повърхнини на решения. Трикът се състои в трансформиране на данни от тяхното оригиналното координатно пространство **x** в едно ново пространство Φ(**x**) по такъв начин, че в това пространство може да се намери някаква линейна повърхнина на решения, която може да се използва за разделяне на трансформираните примери. След извършване на такава трансформация можем да приложим методология, описана в предишните раздели, за да бъде намерена една линейна повърхнина на решения в трансформираното пространство.

16.4.1. Трансформация на атрибути

За да илюстрираме, как трансформацията на атрибути може да доведе до една линейна повърхнина на решения, да разгледаме двумерни данни, показани на Фиг. 16-8 (а). Данните, описвани с атрибути **x** = *(x1, x2),* са генерирани по такъв начин, че положителните примери (y = 1) са вътре в окръжността, а отрицателните (y = -1) – извън нея.



Фиг. 16-8. (а) Множество от двумерни данни с положителни примери, означени като тъмни кръгчета и отрицателни – като светли. Истинската повърхнина на решения $x_1^2 + x_2^2 = 1$ също е показана. (b) Същото множество след неговата трансформация в тримерното пространство с координати $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$. Повърхнината на решения във вид на окръжност от (а) се превърнала в една линейна повърхнина на решения (двумерна равнина) в тримерното пространство.

Очевидно е, че не съществува никаква права линия, разделяща тези два класа в оригиналното двумерно пространство. Сега да предположим, че сме направили трансформация на тези данни, т.е. направихме отображение на всеки входен вектор **x** в новия вектор Φ(**x**), който има следните три атрибута:

$$f_1 = x_1^2, \quad f_2 = x_2^2, \quad f_3 = \sqrt{2}x_1x_2 \qquad (16.28)$$

С други думи:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1 x_2) \qquad (16.29)$$

На Фиг. 16-8 (b) са показани трансформираните данни в това ново тримерно пространство, определено от въведените нови атрибути. Както се вижда в това пространство данните са *линейно сепарабелни*! В действителност този феномен е доста общ – ако данни се отобразяват в някое пространство с достатъчно висока размерност, то те са почти винаги стават линейно сепарабелни – ако вие разглеждате някакво множество от точки от достатъчно голям брой направления, вие ще намерите начин да ги разделите с равнина. В примера са използвани три размерности. В общия случай (с малки изключения), ако имаме *N* точки данни, то те могат винаги да бъдат разделени (т.е. да станат линейно сепарабелни) в пространства с *N-1* или повече размерности.

16.4.2. Научаване на модела на нелинейна SVM

Макар че подходът с трансформация на атрибути изглежда обещаващ, той повдига няколко въпроса, свързани с имплементация. Първо, не е ясно, какъв тип отображение трябва да се използва, за да сме сигурни, че в трансформираното пространство може да бъде намерена някаква линейна повърхнина на решения. Една от възможностите е да трансформираме данни в някакво пространство с безкраен брой размерности, но с такова многоразмерно пространство не се работи лесно. Второ, дори ако подходящата отобразяваща функция е известна, решаването на оптимизационната задача с ограничения в пространството с много размерности е доста скъпа от изчислителната гледна точка задача.

За да илюстрираме тези проблеми и начини, по които те могат да бъдат решени, нека да предположим, че имаме подходяща функция Φ(**x**), която трансформира зададеното множество от данни. След трансформацията трябва да конструрираме една линейна повърхнина на решения, която ще разделя примерите на съответните класове. Линейната повърхнина на решения в това трансформирано пространство ще има следния вид: $\mathbf{w} \cdot \Phi(\mathbf{x}) + b = 0.$

Определение 2 (нелинейна SVM). Задача за научаване на една нелинейна SVM може да бъде формализирана като следната оптимизационна задача:

$$\min_{\mathbf{w}} \frac{\| \mathbf{w} \|^2}{2}$$

при ограничения $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1, \quad i = 1,2,...,N$

Обърнете внимание на сходството между задача за научаване на нелинейната SVM с тази на линейната SVM (Определение 1). Основната разлика се състои в това, че вместо използване на оригиналните атрибути **x**, в задачата се използват трансформираните атрибути Φ(**x**)**.** Следвайки подхода, описан в предишните раздели за линейна SVM, можем да изведем следния двойствен Лагранжиан за оптимизационната задача с ограничения:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \qquad (16.30)$$

След като множителите на Лагранж λ_i са намерени чрез квадратичното програмиране, параметрите **w** и *b* могат да бъдат изведени чрез следните уравнения:

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i) \qquad (16.31)$$

$$\lambda_i \{ y_i (\sum_{j=1}^{N} \lambda_j y_j \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) + b) - 1 \} = 0 \qquad (16.32)$$

Тези уравнения са аналогични на тези (16.13) и (16.14) за линейна SVM. И накрая, един тестов пример **z** може да бъде класифициран в съответствие със следното уравнение:

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign\left(\sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b \right) \quad (16.33)$$

Обърнете внимание, че освен формулата (16.31), всички останали изчисления включват изчисляване на скаларното произведени (т.е. сходство) между двойки от вектори в трансформираното пространство $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Подобни изчисления могат да бъдат доста тежки и могат да страдат от „проклятието на размерността". Пробивът в решаването на този проблем идва с метода, известен като *трикът на ядра (kernel trick)*.

16.4.3. Трик на ядра

Скаларното произведение често се разглежда като мярка за сходство между два входни вектора. Например, известната мярка за сходство – косинус – може да бъде определена като скаларното произведение между два вектора, които са нормализирани до единична дължина. Аналогично, скаларното произведение $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ може също да се разглежда като мярка за сходство между два примера $\mathbf{x}_i$ и $\mathbf{x}_j$ в трансформираното пространство от примери.

Трикът на ядрата е метод за изчисляване на сходство в трансформираното пространство чрез използване на оригиналното множество от атрибути. Да разгледаме, като пример, функцията на отображение Φ, зададена от уравнение (16.29). Скаларното произведение между входните вектори **u** и **v** в трансформираното пространство може да се запише по следния начин:

$$\begin{aligned} \Phi(\mathrm{u}) \cdot \Phi(\mathbf{v}) &= (u_1^2, u_2^2, \sqrt{2}u_1u_2) \cdot (v_1^2, v_2^2, \sqrt{2}v_1v_2) = u_1^2v_1^2 + u_2^2v_2^2 + 2u_1u_2v_1v_2 = \\ &= (\mathbf{u} \cdot \mathbf{v})^2 \end{aligned} \qquad (16.34)$$

Този анализ показва, че скаларното произведение в трансформираното пространство може да бъде изразено в термини на някоя функция на сходство в оригиналното пространство:

$$\mathrm{K}(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^2 \qquad (16.35)$$

Функцията на сходство K, която се изчислява в оригиналното пространство от атрибути, се нарича *функцията на ядра (kernel function)*. Трикът на ядра позволява да реши въпросът с имплементацията на нелинейна SVM. Първо, ние не трябва да знаем точната форма на функцията на отображение Φ, тъй като функциите на ядрата, използвани в нелинейната SVM, трябва да удовлетворяват един математически принцип, известен като *теоремата на Мерсер* (1909). Този принцип гарантира, че функциите на ядра могат винаги да бъдат изразени като скаларното произведение между два входни вектора в някое пространство с много размерности. Трансформираното пространство на SVM ядра се нарича *Хилбертово пространство на възпроизвеждащи ядра (Reproducing Kernel Hilbert Space – RKHS)*. Второ,

изчисляването на скаларното произведение чрез използване на функциите на ядра е значително по-евтино от изчислителната гледна точка от използване на трансформираното множество от атрибути Φ(**x**). Трето, тъй като изчисленията се изпълняват в оригиналното пространство, проблемите, свързани с „проклятието на размерността" могат да бъдат избегнати.

Теоремата на Мерсер. Една функция на ядра K може да бъде изразена като

$$\mathrm{K}(\mathbf{u},\mathbf{v}) = \Phi(\mathbf{u})\cdot\Phi(\mathbf{v})$$

тогава и само тогава, ако за всяка функция g(**x**), такава че $\int g(\mathbf{x})^2\, d\mathbf{x}$ е краен, вярно е:

$$\int \mathrm{K}(\mathbf{x},\mathbf{y})g(\mathbf{x})g(\mathbf{y})d\mathbf{x}d\mathbf{y} \geq 0$$

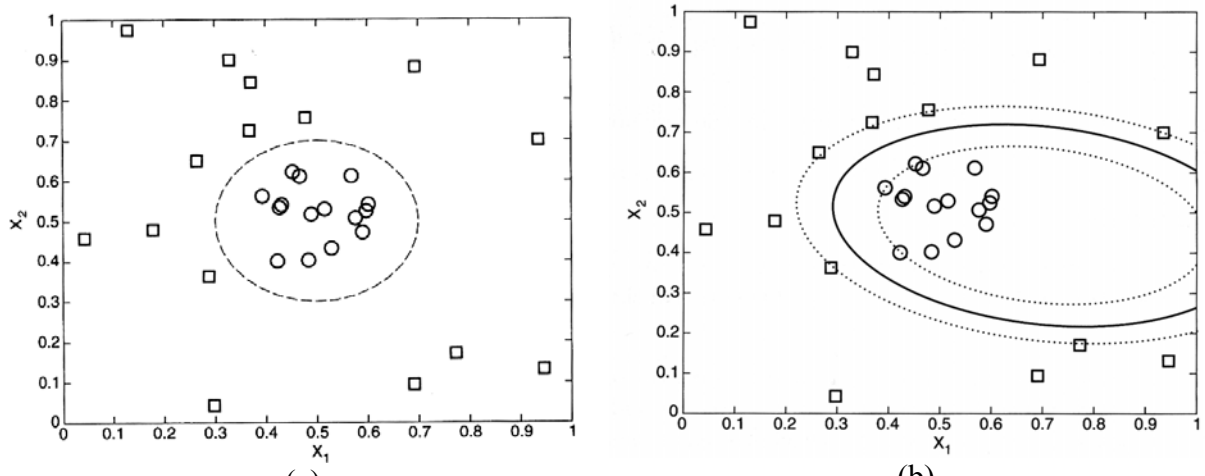
Функциите на ядра, удовлетворяващи теоремата на Мерсер, се наричат *положително определени функции на ядра (positive define kernel functions)*. Освен функцията (16.35), други примери на такива функции са:

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+1)^p \qquad (16.36)$$

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)} \qquad (16.37)$$

$$\mathrm{K}(\mathbf{x},\mathbf{y}) = \tanh(k\mathbf{x}\cdot\mathbf{y}-\delta) \qquad (16.38)$$

Пример 2. Да разгледаме множество двумерни данни, принадлежащи към два класа – квадратчета (положителни примери) и кръгчета (отрицателни примери), изобразени на Фиг. 16-9 (a). Оригиналната повърхнина на решение представлява окръжност, задавана с уравнение: $\sqrt{(x_1-0.5)^2+(x_2-0.5)^2}=0.2$. Фиг. 16-9 (b) показва повърхнината на решения, получена от SVM с използване на функцията на полиномиалните ядра (16.36) със стойността на параметъра $p = 2$.



(a) (b)

Фиг. 16-9. (a) Повърхнина на решение в оригиналното двумерно пространство. (b) Повърхнина на решение, намерена от нелинейната SVM с полиномиалното ядро.

Един тестовия пример **x** се класифицира в съответствие със следното уравнение:

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{N} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b\right) = sign\left(\sum_{i=1}^{N} \lambda_i y_i \mathrm{K}(\mathbf{x}_i, \mathbf{x}) + b\right) =$$

$$= sign\left(\sum_{i=1}^{N} \lambda_i y_i (\mathbf{x}_i \cdot \mathbf{x} + 1)^2 + b\right) \qquad (16.39)$$

където параметър *b* се получава чрез използване на уравнение (16.32). Повърхнината на решение, получена от нелинейната SVM, е доста близка до оригиналната повърхнина на решение, показана на Фиг. 16-9 (а).

16.4.4. Характеристики на SVM

SVM има много желателни качества, които я правят един от най-широко използвани класификационни алгоритми. Основните характеристики на SVM могат да бъдат резюмирани по следния начин:

- Задачата за обучение на SVM може да бъде формулирана като една изпъкнала оптимизационна задача, за която съществуват ефективни алгоритми за намиране на *глобалния минимум* на целевата функция. Други класификационни методи, такива като класификационни правила или изкуствени невронни мрежи, използват евристични стратегии за претърсване на пространството на хипотези. Такива методи имат тенденция за намиране само на локално оптимални решения.
- SVM осъществява контрол върху капацитета (сложността на построявания модел) чрез максимизиране на допустимата граница на повърхнината на решение. Въпреки това потребителят длъжен да зададе други параметри, такива като типа на използвана функция на ядра и ценовата функция *C* за въвеждане на всяка отслабваща променлива.
- SVM може да бъде прилагана и към номинални атрибути чрез въвеждане на фиктивни (dummy) променливи за всяка стойност на номиналния атрибут, срещана в данни. Например, ако атрибутът *Семейно_положение* има три възможни стойности *{Неженен, Женен, Разведен},* то могат да бъдат въведени по една двоична променлива за всяка от тези три атрибутни стойности.
- Формулировката на SVM, приведена в тази лекция, важи за данни, принадлежащи само към два класа. Съществуват няколко стандартни метода, които позволяват да разширят SVM за решаване на задачи с повече от два класа. Някои от тези методи са разгледани в следващия раздел.

Трикът на ядрата е бил предложени от Aizerman et al. (1964), но пълното развитие на теорията на SVM е дело на Вапник и неговите колеги (Boser et al. 1992). Използване на SVM стана практика след въвеждане на подхода за мека допустима граница за обработка на зашумени данни, представен за пръв път в статията на Cortes and Vapnik (1995), която спечели наградата за теория и практика на ACM за 2008 г., и след създаване на алгоритъма са последователна минимална оптимизация (Sequential Minimal Optimization – SMO), който позволява ефективно решаване на SVM задачи чрез използване на квадратичното програмиране (Platt 1999). SVM е много популярен и ефективен метод за такива задачи, като категоризация на текстове (Joachims 2001), изчислителната геномика (Cristiani and Kahn 2007) и обработка на естествен език, включващи разпознаване на ръкописни цифри (DeCoste and Scholkopf 2002). Като част

от този процес са били създадени множество нови ядра, позволяващи работа със символни низове, дървета и други типове нечислови данни.

16.5. Задачи с повече от два класа

Машината на поддържащи вектори, както и някои други класификационни техники (виж, например AdaBoost в лекцията за самообучение чрез ансамбли) са първоначално създадени за решаване на двоични (т.е. с два класа) класификационни задачи. Очевидно е, че съществуват много реални задачи, такива като, например, задачи по разпознаване на символи, идентификация на лица, класификация на текстове и т.н., в които входните данни са разделени на повече от два класа. В този раздел ще разгледаме няколко подхода за разширяване на двоични класификатори по такъв начин, че те могат да решават и задачи с множество класове. За илюстрация на тези подходи ще означим с $Y = \{y_1, y_2, ..., y_k\}$ множеството от класове на входните данни.

16.5.1. Подходи „всеки-срещу-останали" и „всеки-срещу-всеки"

Първият подход декомпозира задача с множество класове на K двоични класификационни задачи. За всеки клас $y_i \in Y$ се създава една двоична класификационна задача, в която всички примери, принадлежащи към класа y_i, се разглеждат като положителни, а всички останали примери – като отрицателни. След това се създава един двоичен класификатор, който разделя примерите от клас y_i от всички останали класове. Този подход е известен под името *всеки-срещу останали (one-against-rest – (1-r))*.

При втория подход, който е известен под името *всеки-срещу-всеки (one-against-one – (1-1))*, се конструират $k(k-1)/2$ двоични класификатора, като всеки класификатор се използва за разделяне на примери, принадлежащи само към една двойка класове (y_i, y_j). И в двата подхода *1-r* или *1-1* тестовият пример се класифицира чрез комбиниране на предсказвания, направени от всички двоични класификатори, като обикновено се използва схема на гласуване – клас, който получава най-много гласове, печели. В подхода *1-r*, ако един пример се класифицира като отрицателен, всички класове, освен положителния клас, получават по един глас. Обаче този подход може да доведе до ситуации, когато няколко класове получават еднакъв брой гласове. Другата възможност се състои в преобразование на изходи на двоични класификатори във вероятностни оценки и след това да се назначава тестовия пример към най-вероятния клас.

Пример 3. Да разгледаме една задача с четири класа $Y = \{y_1, y_2, y_3, y_4\}$. Да предположим, че тестовия пример се класифицира като {+, -, -, -} съгласно подхода *1-r*. С други думи, той се класифицира като положителен, когато y_1 се използва като положителен клас, и като отрицателен, когато y_2, y_3 и y_4 се използват като положителен клас. При използване на простата схема за гласуване, y_1 получава най-големия брой гласове – 4, докато останалите класове получават по 3 гласа. По тази причина тестовият пример се класифицира като принадлежащ към класа y_1.

Сега да предположим, че при използване на подхода *1-1* тестовият пример се класифицира по начина, показан в таблицата по-долу. Първите два реда в таблицата съответстват на двойки класове (y_i, y_j), избрани за построяване на класификатора, а

последният ред представлява класа, предсказания за тестовия пример. След комбиниране на предсказванията класове y_1 и y_4 получават по два гласа всеки.

Двойки класове	$+: y_1$ $-: y_2$	$+: y_1$ $-: y_3$	$+: y_1$ $-: y_4$	$+: y_2$ $-: y_3$	$+: y_2$ $-: y_4$	$+: y_3$ $-: y_4$
Класификация	$+$	$+$	$-$	$+$	$-$	$+$

По тази причина тестовият пример се класифицира или към y_1 или y_4 в зависимост от избраната процедура по разрешаване на конфликти (равенства в гласове).

16.5.2. Коригиращо грешки кодиране на изхода

Един потенциален проблем с предишните два подхода е, че те са чувствителни към грешки, допускани от двоични класификатори. Така за подхода *1-r*, използван в Примера 3, ако най-малко един от двоични класификатори сгреши в своето предсказване, то целият ансамбъл ще изведе решение за равенство между класове или ще направи някакво грешно предсказване. Например, да предположим, че тестовият пример е класифициран като {+,-,+,-} поради грешка на третия класификатор. В този случай ще бъде трудно да се каже, дали примерът трябва да бъде класифициран като y_1 или y_3 освен ако не се отчитат вероятностите, асоциирани със предсказания за всеки клас.

Методът за коригиращо грешки кодиране на изхода (error-correcting output coding – ECOC) предоставя един по-надежден начин за решаване на задачи с множество класове. Методът е инспириран от един информационно-теоретичен подход към задача за изпращане на съобщения по зашумени канали. Идеята, която стои зад този подход, е да се добави излишество (redundancy) към предаваното съобщение с помощта на една кодирана дума, така че получателят може да определи грешки в полученото съобщение и, вероятно, да възстанови оригиналното съобщение, ако броят на грешки е малък.

При класификационни задачи с множество класове всеки клас y_i се представя чрез един уникален битов низ с дължина n, който се нарича *кодова дума (codeword)*. След това n двоични класификатора се обучават да предсказват всеки бит от тази кодова дума. Предсказаният клас на тестовия пример се задава от кодираната дума, за която разстояние по Хеминг до кодираната дума, конструирана от ансамбъла от двоични класификатори, е най-малко. Напомням, че разстоянието по Хеминг между една двойка от битови низове се определя от броя на битове, които са различни в тази двойка.

Пример 4. Да разгледаме задача с четири класа $Y = \{y_1, y_2, y_3, y_4\}$. Да предположим, че сме кодирали класове, използвайки следните 7-битови кодови думи:

Клас	Кодова дума						
y_1	1	1	1	1	1	1	1
y_2	0	0	0	0	1	1	1
y_3	0	0	1	1	0	0	1
y_4	0	1	0	1	0	1	0

Всеки бит от кодовата дума се използва за обучение на един двоичен класификатор. Например, първият двоичен класификатор се обучава върху данни, в които положителните примери са от клас y_1, а всички останали са отрицателни. Вторият двоичен класификатор се обучава върху данни, в които като положителни са отбелязани примери, принадлежащи към класове y_1 и y_4, а всички останали – отрицателни, и т.н. докато всички 7 двоични класификатора не бъдат обучени. Да предположим, че един тестов пример се класифицира от получения ансамбъл като {0,1,1,1,1,1,1}. В този случай разстоянието по Хеминг между тази дума и кодовата дума за клас y_1 е равно на 1, докато същото разстояние до всички останали класове е равно на 3. Следователно този тестов пример се класифицира като принадлежащ към класа y_1.

Едно интересно свойство на коригиращия грешки код се състои в това, че ако минималното разстояние по Хеминг между всяка двойка на кодови думи е d, то всякакви $\lfloor (d-1)/2 \rfloor$ брой грешки в изходния код могат да бъдат коригирани чрез използване на най-близката кодова дума. В Примера 4, тъй като минималното разстояние по Хеминг между двойките думи, кодиращи класове, е 4, то ансамбълът може да толерира грешки, направени само от един от седемте двоични класификатора. Ако повече от един двоичен класификатор сгреши, ансамбълът няма да бъде в състояние да компенсира тези грешки.

Очевидно е, че важният въпрос при този подход е как да се конструира подходящото множество от кодирани думи за различни класове. В теорията на кодиране са били разработени множество методи за генериране на n-битови кодови думи с ограничено разстояние по Хеминг. Обаче, обзор на тези методи е извън обхвата на тази лекция. Струва да се отбележи, обаче, че има значителната разлика между създаването на коригиращи грешки кодове за комуникационни задачи и за тези, използвани при научаване на класификатори с множество класове. За комуникационните задачи кодираните думи трябва да максимизират разстоянието по Хеминг между редове, така че корекцията на грешки може да бъде извършвана. Задачите за научаване на много класове, обаче, изискват, както разстоянията между редове, така и разстоянията между колонки от кодираните думи трябва да бъдат достатъчно големи. По-голямото разстояние между колонки осигурява, че двоичните класификатори ще бъдат взаимно независими, което е едно важно изискване на методи за самообучение чрез ансамбли (виж съответната лекция).