Лекция 17. Самообучение чрез ансамбли

17.1. Въведение

Когато умните хора трябва да приемат някои важни решения, те обикновено не разчитат само на собствените си съждения или на предложения от един единствен съветник, а взимат под внимание мнения на няколко експерти. Например, преди да предложи някоя нова важна политика, един умен диктатор провежда широки консултации – лошия подход е той или тя да следва сляпо предложение само на един експерт. При демократичното управление дискусията, при която се разглежда някой проблем от различни гледни точки, може да доведе до консенсус; ако не, решението може да бъде прието с гласуване. Във всички случаи мненията на различни експерти се комбинират.

В машинното самообучение един модел, генериран от някой алгоритъм, може да се разглежда като „експерт" (макар че тази дума е твърде силна в този контекст, тъй като в зависимост от размера и качество на обучаващите данни и от това, дали алгоритмът е подходящ за решавания проблем, мнението на такъв „експерт" може да бъде прието на внимание или игнорирано). Един очевиден подход да направим процеса на приемане на решение по-надежден е да комбинираме изходи на различни модели. Няколко техники на машинно самообучение правят това чрез научаване на една колекция (*ансамбъл*) от модели и тяхното комбинирано използване.

Защо смятаме, че предсказванията на ансамбъла ще бъдат по-точни от предсказвания само на един единствен класификатор (модел)? Да разгледаме, като пример, ансамбълът от 5 модела (или хипотези) и да предположим, че ще комбинираме техните предсказвания чрез една проста схема за приемане на решения чрез гласуване с болшинство. За да бъде един пример класифициран *правилно* от такъв ансамбъл, трябва *най-малко три от петте* участващи в него модела да класифицират същия пример правилно. Нека предположим, че класификационната точност (т.е. вероятността, че един произволен пример да бъде класифициран правилно) на всеки от модели е $p = 0.7$. Освен това, да предположим, че класификациите (или грешки), които се правят от всеки от участващи в ансамбъла модели не зависят една от друга. В този случай теорията на вероятностите ни позволява да изчислим класификационната точност на такъв ансамбъл – тя е равна на сумата от вероятността, че примерът ще бъде правилно класифициран от три от петте модела, плюс вероятността, че четири от петте модела класифицират пример правилно, плюс вероятността че всички пет модела правят същото:

$$P(h_1, h_2, h_3, h_4, h_5) = \sum_{k=3}^{5} C_5^k p^k (1-p)^{5-k} = \sum_{k=3}^{5} \frac{5!}{(5-k)!k!} p^k (1-p)^{5-k} = 10 * 0.7^3 * 0.3^2 +$$

$$5 * 0.7^4 * 0.3 + 0.7^5 = 0.837$$

Ако предположим, че при същите условия в ансамбъла участват 101 модела, то неговата класификационната точност ще стане 0.999!

Очевидно е, че предположението за независимост на предсказванията на участващи в ансамбъла класификатори е нереалистична, тъй като по-вероятно е, че повечето от класификатори ще бъдат заблудени по сходния начин от едни и същи подвеждащи аспекти в обучаващите данни. Обаче, ако класификаторите са достатъчно различни, намалявайки по този начин корелацията между своите грешки, обучението чрез ансамбъл може да стане много ефективно.

Друг начин да се мисли за идеята на ансамбъла е като за един общ подход за увеличаване на пространството на хипотези за научаваните понятия. С други думи, всеки ансамбъл може да се разглежда като една хипотеза в новото пространство от хипотези, представляващо всички възможни ансамбли, които могат да бъдат създадени. Очевидно е, че ако оригиналното пространство от хипотези описва множество от сравнително прости и ефективни алгоритми, то методът на ансамбъла предоставя един начин за научаване на много по сложен клас от хипотези (и съответно, понятия) без съществено увеличаване на изчислителната или алгоритмичната сложност.

Най-известните подходи в областта на самообучение чрез ансамбли се наричат bagging, boosting и stacking. Всички те, в повечето от случаи, водят до увеличаване на предсказващата точност в сравнение с един единствен модел. Алгоритмите bagging, boosting и stacking са били създадени в последните 2-3 десетилетия и тяхното поведение е често учудващо добро. Изследователите опитвали да разберат причините за това, като при тези опити бяха разработени нови методи, които понякога дори са още по-добри. Всички комбинирани модели имат един общ недостатък – те много тежко могат да бъдат анализирани. Ансамблите могат да включват десетки и дори стотици индивидуални модели и макар че те имат добро поведение, съвсем не е лесно да се разбере, кои фактори допринасят до подобряване на произвежданите решения.

17.2. Bagging

Комбинирането на решенията на различни модели означава съчетаване на различните изходи в едно единствено предсказване. Най-простият начин да направим това е чрез гласуване (евентуално, чрез претегленото гласуване). И двата алгоритъма bagging и boosting се базират върху този подход, но използват индивидуалните модели по различен начин. При bagging моделите получават еднакво тегло, докато при boosting теглата се използват, за да се предаде по-голямо влияние на по-успешните модели – аналогично на това, като един отговорник за приемането на решения дава различни тегла на мнения на различни експерти в зависимост от това, доколко техните предсказвания са били успешни в миналото.

За да се разбере алгоритъма на bagging, да предположим, че от някоя проблемна област сме избрали по случаен начин няколко различни обучаващи извадки с еднакъв размер. След това с помощта на някой от известните ни алгоритми от всяко обучаващото множество сме построили по едно класификационно дърво. Може да се очаква, че тези дървета ще бъдат практически еднакви и ще правят идентични предсказвания за всеки нов тестов пример. Обаче, учудващо е, че това предположение в повечето случаи е напълно погрешно, особено в случаите, когато обучаващите множества са малки. Това е един доста обезпокоителен факт, който изглежда хвърля сянка на съмнение на цялото ни начинание. Причината за това е, че пораждането на класификационни дървета (най-малко съгласно стандартния метод „отгоре-надолу”, който сме учили по-рано) е един нестабилен процес – малки промени в обучаващите данни могат да доведат до избора

на различни атрибути във всеки конкретен възел на дървото, което води до значителни промени в структурата на под-дърво с корен в този възел. От това автоматически следва, че ще съществуват такива тестови примери, за които едни дървета ще направят правилното предсказване, а други – грешното.

Връщайки се към аналогията с експерти, да представим, че всяко отделно класификационно дърво е експерт. Можем да комбинираме дървета като ги накараме да гласуват върху всеки тестов пример Ако един клас получи повече гласове от други, той ще бъде избран като крайното решение. В общия случай колкото повече експерти, толкова по-добро е решението, т.е. предсказванията, направени чрез гласуване, стават по-надеждни (по-точни) при нарастване на броя на отделни „гласове", участващи в гласуването. Качеството на крайните решения рядко се влошава при постъпване на нови множества от обучаващи примери – в тези случаи за тях се правят нови дървета и техните предсказвания започват да участват в гласуването. В частност, комбинираният класификатор рядко ще бъде по-малко точен от едно единствено класификационно дърво, построено само от едно от обучаващите множества. Обаче подобряването не е гарантирано – може да се покаже теоретически, че съществуват патологическите ситуации, в които комбинираните решения са по-лоши.

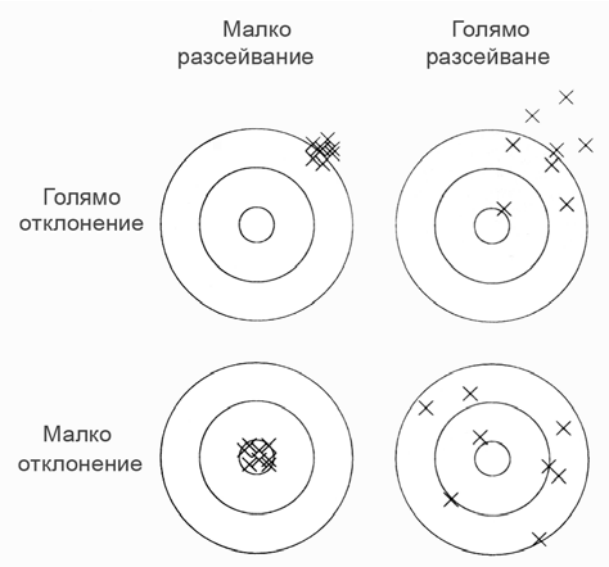
17.2.1. Декомпозиция „отклонение – разсейване"

Ефектът от комбиниране на различни класификатори може да бъде разгледан чрез призмата на една теоретична схема, известна като декомпозицията „отклонение – разсейване" (bias – variance). Да предположим, че разполагаме с неограничено количество независими обучаващи множества с един и същ размер, които можем да използваме за научаване на неограничен брой класификатори. Всеки тестов пример се обработва от всеки от класификаторите и крайната класификация на примера се определя от болшинството чрез гласуване. И в тази идеализирана схема ще има грешни класификации, тъй като нито един метод за самообучение не е съвършен: броят на грешките ще зависи от това, доколко конкретният метод за самообучение, използван за научаване на конкретния класификатор, отговаря на решаваната задача; освен това влиянието ще оказва и наличието на шума в данни, който на практика не може да бъде научен.

Да предположим, че очакваната класификационна грешка е била оценена чрез осредняване на грешки, допуснати от комбинацията от класификатори върху независимо избрани тестови примери. Класификационната грешка на един конкретен алгоритъм за самообучение се нарича *отклонение (bias)* на алгоритъма за решавания проблем и оценява, доколко добре съответния метод за самообучение „пасва" на проблема (в този термин включваме и компонента на шума, тъй като на практика той никога не е известен). Предложената техническа дефиниция е един опит да придаде количествения характер на по-размитото понятие *индуктивното пристрастие*, което вече беше въведено по-рано. Отклонението измерва „твърда" грешка на един алгоритъм за самообучение, която не може да бъде елиминирана чрез използване дори на безкраен брой обучаващите множества. Естествено е, че тази мярка не може да бъде изчислена абсолютно точно – тя може да бъде оценена само приблизително.

Вторият източник на грешки, допускан от един алгоритъм за самообучение в една конкретна ситуация, е видът на конкретното обучаващо множество, използвано при

обучение. Това множество е крайно и, следователно, не представлява напълно действителното разпределение на примерите от избраната проблемна област. Очакваната стойност на този компонент от класификационната грешка на алгоритъма, осреднена върху всички възможни обучаващи множества с дадения размер и всички възможни тестови множества, се нарича *разсейване* (*variance*) на метода за самообучение за конкретния проблем. Общата очаквана класификационна грешка на един класификатор е сума от неговото отклонение и разсейване. Тази схема на разбиване на класификационната грешка на отделни компоненти се нарича *декомпозиция „отклонение – разсейване”*. Фигура 17-1 илюстрира тези понятия чрез аналогията с хвърляне на стрелички в мишена (играта darts).



Фиг. 17-1. Отклонение и разсейване при хвърляне на стрелички

От гледната точка на тази схема комбинирането на няколко класификатори в общия случай намалява очакваната грешка чрез намаляването нейния компонент „разсейване”. Колкото повече класификатори ще бъдат включени в ансамбъла, толкова по-голямо ще бъде намаляване на разсейването. Естествено е, трудностите възникват при практическото прилагане на тази схема, тъй като в реалните случаи разполагаме само с едно обучаващото множество и получаването на повече данни или е невъзможно, или е много скъпо.

Алгоритмът bagging прави опит да неутрализира нестабилността на методите за самообучение чрез симулация на описания по-горе процес, използвайки само едно единствено налично обучаващо множество. Вместо да създава всеки път нови независими обучаващи извадки от оригиналното обучаващото множество, самото оригинално обучаващо множество се променя чрез изтриване на едни негови примери и дублиране на други. За създаване на една нова обучаваща извадка с фиксиран размер от оригиналното обучаващо множество се избират по случаен начин примери, които *не се изтриват* от оригиналното множество (такава извадка се нарича извадка със заместване – sampling with replacement). Тази процедура на практика дублира едни примери и изтрива други.

Трябва да се отбележи, че методът за създаване на извадка със заместване стои в основа на така наречения метод bootstrap за оценяване на класификационната грешка на

алгоритми за самообучение. Може да се покаже, че ако от едно множество от *n* примера се създава друго множество пак от *n* примера чрез прилагане на метода за създаване на извадка със заместване, то полученото множество ще съдържа 63.2% от примерите, присъстващи в оригиналното множество. По тази причина тези 63.2% от примерите могат да се ползват като обучаващото множество, а останалите 36.8% - като тестово множество, а класификационната грешка на алгоритъма може да се прецени като претеглената сума от грешките на алгоритъма върху тестовото и обучаващото множества:

$$Err = 0.632 \times Err_{Testing} + 0.368 \times Err_{Training}$$

17.2.2. Описание на алгоритъма

Самото название на алгоритъма bagging произтича от Bootsrap AGGregatING (Breiman 1996). Алгоритмът прилага някой избран метод за самообучение (например класификационни дървета) към всеки от така създадени изкуствени обучаващи множества, като след това решенията на получените класификатори се комбинират чрез простата схема на гласуване с болшинство. Псевдо-кодът на алгоритъма bagging е приведен в Таблица 17-1.

Генериране на модела (обучение)
Вход: обучаващото множество, съдържащо *n* примери
K – броят на базови модели в ансамбъла
L – видът на избрания алгоритъм за самообучение

Алгоритъм:
За всяка от *K* итерации направи:
Направи извадка с заместване с размер *n* от обучаващото множество
Приложи избрания алгоритъм за самообучение *L* към тази извадка
Запомни научения модел (класификатор)

Класификация

За всеки от *K* модела направи:
Предскажи клас на проверявания пример с помощта на модела
Върни клас, който е бил предсказан от болшинство от моделите

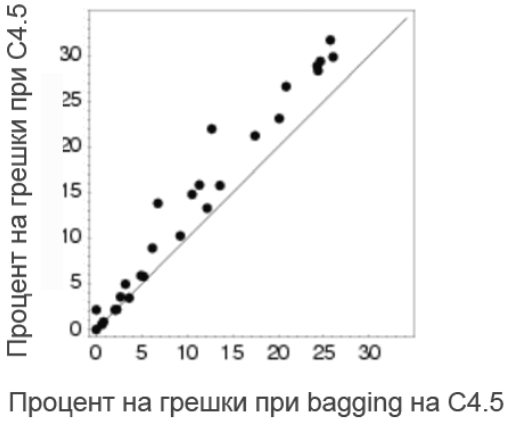
Таблица 17-1. Псевдо-код на алгоритъма bagging

Разликата между алгоритъма bagging и описаната в началото на лекцията идеализирана схема на алгоритъма за създаване на ансамбъла се състои в начина на формиране на обучаващите множества, използвани за научаване на участващи в ансамбъла модели. Вместо получаване от проблемната област на независими обучаващи множества, bagging използва само изкуствено създадени „варианти" (извадки) на едно единствено обучаващо множество. Очевидно е, че обучаващите множества, генерирани чрез извадка със заместване, се отличават едно от друго, но по никой начин не са независими, тъй като са направени от примерите, взети от едно и също множество. Обаче, се оказва, че bagging създава комбинирания модел, който често има значително

по-добро поведение от един единствен модел, научен върху оригиналните обучаващи данни, и никога няма значително по-лошо поведение от подобния модел.

Алгоритмът bagging най-много помага, когато използваният базов алгоритъм за самообучение е нестабилен, така че малките промени във входните данни могат да доведат до значително по-различни класификатори. Ясно е, че получените резултати могат да бъдат подобрени чрез засилване на различията между участващите в ансамбъла класификатори, което може да се постигне чрез направяне на използвания алгоритъм за самообучение колкото може по-нестабилен. Например, при използване на bagging с класификационни дървета, които сами по себе си нестабилни, по-доброто поведение често се постига чрез премахване на процедурата по подрязване на дървета, в резултат на което дърветата стават още по-нестабилни. Другото подобрение може да се получи чрез промяна на начина на комбиниране на предсказвания на базовите класификатори. Оригиналният алгоритъм bagging използва непретегленото гласуване. Обаче в случаи, когато базовият класификатор може да предсказва не само клас, но и дава оценка (вероятност) за правилността на своето предсказване, има смисъл тези оценки да бъдат използвани като тегла при гласуването. Това води не само до неголямо увеличаване на точността, но, тъй като самият ансамбъл вече дава оценка за правилността на своето предсказване, то подобрява и точността на тази оценка. Повечето от имплементации на алгоритъма bagging използват този метод за комбиниране на предсказвания.

Пример 17-1. В работата на (Freund and Shapire 1996) е направено сравнение на алгоритъма bagging върху 27 бази данни от известното ни MC хранилището на данни UCI. В сравнението участвали алгоритмът за пораждане на класификационни дървета C4.5 и ансамбъл, съставен от 100 C4.5 класификатори. Приведената по-долу графика показва, че ансамбълът има по-добро поведение практически на всички бази от данни.



17.3. Рандомизация

Алгоритмът Bagging генерира един разнообразен ансамбъл от класификатори чрез въвеждане на рандомизация (случаен елемент) във входа на алгоритъма за самообучение, което често води до прекрасни резултати. Обаче има и други начини за постигане на разнообразие чрез въвеждане на рандомизацията. Някои алгоритми за самообучение имат вграден случаен компонент. Например, при обучение на многослойни невронни мрежи чрез алгоритъма Backpropagation инициализацията на

теглата на мрежата става по случаен начин с малки числа. Наученият класификатор зависи от тези случайни числа, тъй като в зависимост от тях алгоритмът може да намери различни локални минимуми на функцията на грешката. Един от начините да направиш изхода на такъв класификатор по-стабилен се състои в няколкократно пускане на алгоритъма с различни случайни числа като ядра и след това в комбиниране на предсказвания на получените класификатори чрез гласуване.

Почти всеки алгоритъм за самообучение е податлив на някой вид рандомизиране. Да разгледаме, например, един алгоритъм, който на всяка стъпка прави изчерпващо претърсване, за да намери най-добрата стойност на определен свой параметър, както това прави алгоритмът за научаване на класификационни дървета при избора на най-добрия атрибут за разделяне в текущия възел на дървото. Този процес може лесно да бъде рандомизиран като вместо избора на най-добрата възможност да се прилага случаен избор от N най-добрите кандидата или чрез случаен избор на N кандидата и след това - избор на най-добрия от тях. Естествено, че трябва да има компромис – по-голямата доза на случайността води до по-голямо разнообразие в научаваните алгоритми, обаче това води и до по-малка степен на използване на наличните данни, което, евентуално, води до намаляване на точността на отделни класификатори. Най-добрата доза на случайността може да се определи само чрез експерименти.

Макар че bagging и рандомизацията водят до сходни резултати, понякога заслужава тези подходи да бъдат комбинирани, тъй като те въвеждат случаен елемент по различни, възможно взаимно допълващите се начини. Ансамбълът от класификационни дървета, получени чрез рандомизация, често се нарича *случайна гора (random forest)*. Един популярен алгоритъм за научаване на случайна гора построява рандомизираното класификационно дърво на всяка итерация на алгоритъма bagging, като полученият ансамбъл често има много добра предсказващата точност.

17.3.1. Рандомизация срещу bagging

Рандомизацията изисква повече работа от колко bagging, тъй като самият алгоритъм за самообучение трябва да бъде модифициран, обаче този подход може да бъде приложен към по-широк кръг от алгоритми за самообучение. Bagging пропада в случая на стабилни алгоритми за самообучение, изход на които не е чувствителен към малките промени във входни данни. Например, безсмислено е да използваш bagging за класификатори, построени на базата на алгоритъма за най-близки съседи, тъй като техния изход се променя много малко при промяната да входни данни чрез прилагането на извадки със заместване. Обаче рандомизацията може да бъде приложена дори към стабилни класификатори – трикът е да ги рандомизираш по начина, който прави класификатори достатъчно разнообразни без да бъде пожертвана в значителната степен тяхната точност. Предсказванията на алгоритъма на най-близкия съсед зависят от разстояние между примери, което, от своята страна, силно зависи от това, кои атрибути са били избрани за изчисляване на разстоянието. Следователно, класификатори на базата на алгоритъма за най-близкия съсед могат да бъдат рандомизирани чрез случаен избор на подмножества от атрибути, участващи в изчисляване на разстоянията. Този подход се нарича *методът на случайни подпространства (random subspace)* и за пръв път е бил предложен за научаване на една случайна гора. Както и bagging този метод не изисква никаква модификация на самия алгоритъм за самообучение. Освен това случайните подпространства могат да се използват заедно с bagging, за да добавят към

процеса на самообучение случаен елемент както в термините на примери, така и на атрибути.

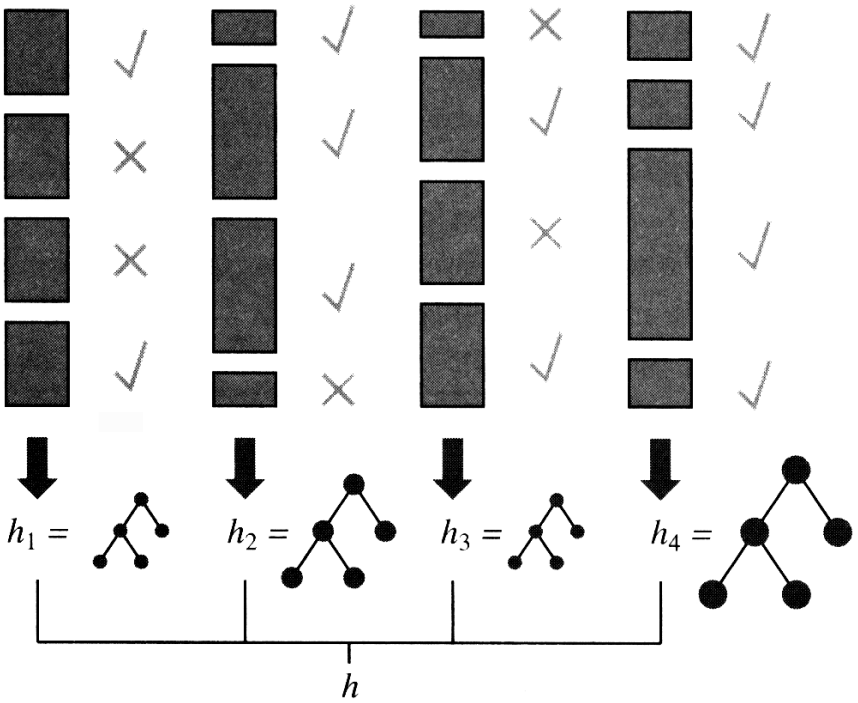
17.4. Boosting

Както вече отбелязахме няколко пъти, bagging използва нестабилността, характерна за алгоритми за самообучение. На интуитивното ниво е ясно, че комбинирането на няколко модела може да помогне само, когато от едната страна тези модели са съществено различни, а, от другата, всеки модел обработва една разумна част от данни коректно. В идеалния случай моделите се допълват, като всеки от тях е специалист в тази част от проблемната област, в която останалите нямат добро поведение.

Методът за комбиниране на класификатори boosting (засилване) използва описаната по-горе идея чрез явното търсене на модели, които допълват един друг. Както и bagging, за комбиниране на изходите на базовите модели boosting използва метода на гласуване; освен това както и bagging той комбинира модели от един и същ тип – например, класификационни дървета. Обаче в отличието от bagging, boosting представлява един итеративен метод – докато при bagging базовите модели се построяват независимо един от друг, при boosting всеки нов модел е повлиян от поведението на моделите, построени преди него. Boosting насърчава новите модели да бъдат експерти за примери, обработени неправилно от по-ранните модели, като назначава на такива примери по-голямо тегло. И накрая, при крайната класификация boosting претегля приноса на всеки модел чрез неговата компетентност, а не ги оценява с еднакви тегла, като при bagging.

Boosting е най-широко използван метод за самообучение чрез ансамбли. В основата на метода лежи идеята за *претеглено обучаващо множество*. В такова множество със всеки пример е асоциирано тегло ($w_i \geq 0$) – колкото по-голямо е теглото на примера, толкова е по-важен той за процеса на научаване на модела. Наличието на теглата на примери променя начина на изчисляване на грешката, допускана от класификатора върху обучаващото множество: вместо пропорцията на неправилно класифицирани примери сега тя се изчислява като сума от теглата на неправилно класифицирани примери делена върху общото тегло на всички примери от множеството. Чрез претеглянето на обучаващите примери един алгоритъм за самообучение може да бъде накаран да се концентрира върху определено множество от примери – тези с по-голямо тегло. Подобни примери стават особено важни, тъй като алгоритмът има по-голям стимул да ги класифицира коректно. Повечето от алгоритмите за самообучение могат сравнително лесно да бъдат модифицирани в посока на работа с претеглени примери. Например алгоритмът C4.5 е способен да работи без никакви модификации с претеглените примери, тъй като вече използва понятието за дробни примери при обработка на примери с липсващите атрибутни стойности. За тези алгоритми за самообучение, които не могат да бъдат лесно приспособени към претеглените примери, оригиналното обучаващото множество се заменя с така наречено *обучаващото множество с реплики (replicated training set)*, в който примерът с теглото w_i се повтаря w_i пъти, като за дробните тегла се използва техниката за рандомизацията. Недостатъкът на този подход е в това, че някои примери с малки тегла могат изобщо да не попаднат в построеното по този начин обучаващо множество с реплики, така че се получава определена загуба на информация преди прилагането към това множество на алгоритъма за самообучение.

Boosting започва работата си с присвояване на всички обучаващи примери еднакво тегло $w_i = 1$ (т.е. с оригиналното обучаващо множество). Към тези данни се прилага избрания алгоритъм за самообучение и се генерира първия модел. Моделът класифицира някои от примери правилно, а други – неправилно. На следващата стъпка теглата на правилното класифицирани примери се намаляват, а на тези, които класифицирани грешно, се увеличават. Полученото претеглено обучаващо множество се използва на следващата итерация за получаване на новия модел. Цикълът с промяна на теглата на примери и пораждане от тях на нов класификатор се повтаря K пъти, където K е параметър на алгоритъма. Крайният ансамбъл представлява комбинация от създадените K модела, като техните решения се комбинират чрез претегленото гласуване, при което теглото на всеки модел се определя от това, колко добре той е класифицирал пораждащо го претеглено обучаващо множество от примери. Фиг. 17-2 илюстрира работата на алгоритъма boosting.



Фиг. 17-2. Как работи boosting. Всеки защрихован правоъгълник съответства на един обучаващ пример; височина на правоъгълника отговаря на теглото на примера. „Човката" и „кръстче" указват, дали примерът е бил класифициран правилно от текущата хипотеза. Размерът на класификационното дърво указва теглото на „гласа" съответната хипотеза в ансамбъла.

17.4.1. Алгоритъм AdaBoost

AdaBoost е един от най-широко използвани алгоритми за научаване на ансамбли чрез метода boosting. Алгоритмът предлага следния начин за промяна на тегла на обучаващите примери при научаване на нов базов модел (класификатор) – ако Err (стойност между 0 и 1) е грешката на текущия базов класификатор върху текущото претеглено обучаващо множество, то новото тегло на всеки *правилно класифициран* от този класификатор пример w_i се обновява съгласно следната формула:

$$w_i \leftarrow w_i \frac{Err}{1 - Err}$$

Теглата на *неправилно класифицирани* примери – не се променят. На пръв поглед това противоречи на идеята на boosting за увеличаване на тегла на неправилно класифицирани примери. Обаче, след изчисляване на новите тегла на всички обучаващи примери в алгоритъма AdaBoost се прилага процедура по нормализацията на теглата, която запазва сумата на теглата постоянна. А именно, ако означим с $w_i(old)$ – старото тегло на обучаващия пример i, чрез w_i – теглото на същия пример, изчислено след прилагането на текущия класификатор към този пример, то $w_i(new)$ – новото тегло на примера, което ще се използва за научаване на новия класификатор, се изчислява по следната формула:

$$w_i(new) = w_i \frac{\sum_j w_j(old)}{\sum_j w_j}$$

В резултат на тази нормализация теглото на всеки неправилно класифициран пример автоматично се увеличава, а на правилно класифициран пример – се намалява.

Когато грешката на текущия класификатор върху претегленото обучаващо множество става равна или по-голяма от 0.5, алгоритъм AdaBoost изтрива текущия класификатор и повече не извършва никакви итерации (т.е. прекратява процедурата по построяване на ансамбъла). Същото става и когато грешката на текущия класификатор става равна на 0, тъй като в този случай теглата на всички обучаващи примери стават равни на 0. Във всички останали случаи алгоритмът продължава итеративната процедура по научаване на следващия класификатор, докато броят на научените класификатори не достигне зададената от потребителя стойност K.

Както беше отбелязано по-рано, един boosting алгоритъм формира своето крайно предсказване чрез претегления вот от предсказвания на всеки от формиращите ансамбъл класификатори. Ясно е, че теглото на всяко предсказание зависи от поведението на съответния класификатор върху претегленото обучаващо множество, от което той е бил научен. С други думи, един класификатор, който има добро поведение върху съответното обучаващо множество (т.е. грешката му *Err* е близо до *0*), трябва да получи голямо тегло, докато класификаторът с лошо поведение (*Err* е близо до *0.5*) трябва да получи малко тегло. В алгоритъма AdaBoost се прилага следната формула за изчисляване на теглата на гласове на класификатори, формиращи ансамбъла:

$$z_k = -\log \frac{Err_k}{1 - Err_k}$$

Изчисленото тегло е положително число между нула и безкрайност. Между другото, тази формула обяснява, защо класификатор с перфектно поведение върху обучаващите данни трябва да бъде премахнат - когато $Err = 0$ теглото става неопределено.

За получаване на предсказването на ансамбъла теглата на класификатори, гласуващи за определен клас, се сумират и се избира класът са най-големия вот. Псевдо-кодът на алгоритъма AdaBoost е приведен в Таблица 17-2.

Алгоритъм AdaBoost
Вход: *examples* – множество от N обучаващи примера $(x_1,y_1),\ldots,(x_N,y_N)$
L – алгоритъм за самообучение
K – желаният брой на хипотези (класификатори) в ансамбъла
Изход: h – вектор от K хипотези
z - вектор от K тегла на хипотези

Локални променливи: ***w*** – вектор от *N* тегла на обучаващи примери.
В началото $w_i = 1/N$ за всички *i*.

---

for *k = 1* ***to*** *K* ***do***
h[k] ← *L(examples,* ***w)***
Err ← *0*
for *j=1* ***to*** *N* ***do***
if $\mathbf{h}[k](x_j) \neq y_i$ ***then*** *Err* ← *Err +* ***w[j]***
for *j=1* ***to*** *N* ***do***
if $\mathbf{h}[k](x_j) = y_i$ ***then*** ***w[j]*** ← ***w[j]*** * *Err/(1 – Err)*
w ← *Normalize(****w****)*
if *(Err* ≠ *0* ***and*** *Err* < *0.5)* ***then***
z[k] ← *log(1- Err)/Err*
return(h, z)

Таблица 17-2. Псевдо-код на алгоритъма AdaBoost

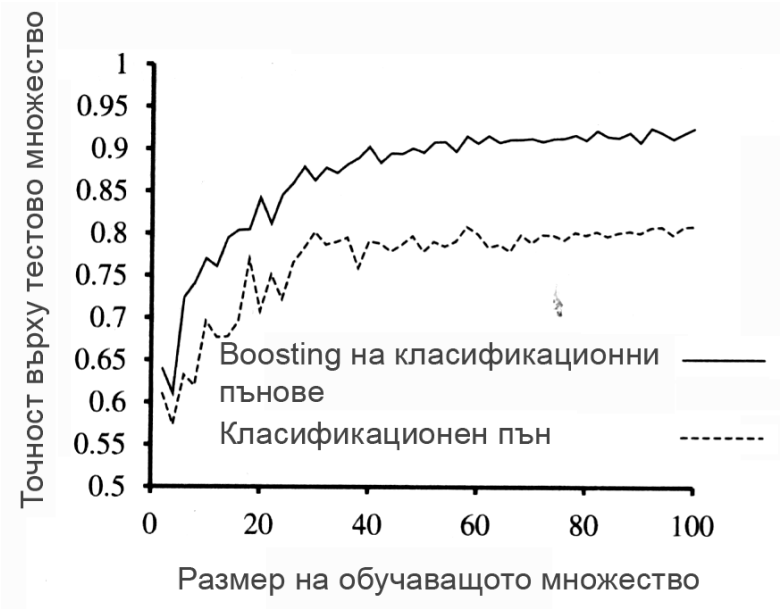
17.4.2. Потенциала на boosting

Идеята на boosting произлиза от така наречена *теория на изчислителното самообучение* (computational learning theory) – един от клонове на машинното самообучение. Теоретиците се интересували от boosting, защото този подход позволява да бъдат направени определени оценки за поведение на пораждани от него класификатори. Например може да се покаже, че при определени условия грешката на ансамбъла от класификатори върху обучаващото множество стига до нула експоненциално бързо по отношение на броя на итерации (т.е броя на класификатори в ансамбъла). С други думи, ако входния алгоритъм за самообучение, използван в процедурата boosting, е един *слаб* (week) алгоритъм за самообучение, което означава, че той винаги поражда хипотеза (класификатор), чиято класификационната точност върху обучаващото множество е малко по-добра от случайното налучкване (т.е. 50% + ε за двоична класификация), то създаденият чрез boosting ансамбъл винаги ще постигне 100% точност върху обучаващото множество при достатъчно голям брой класификатори в ансамбъла. Следователно, boosting алгоритъм засилва точността на оригиналния алгоритъм за самообучение върху обучаващите данни при това без значение, колко изразително е оригиналното пространство от хипотези и колко е сложна целевата функция, която трябва да бъде научена.

За съжаление, както вече знаем, постигането на абсолютната точност върху обучаващите данни не е гаранция за добро класификационно поведение върху нови, „свежи" данни. Обаче теоретично е доказано, че boosting пропада върху нови данни само тогава, когато индивидуалните класификатори, формиращи ансамбъла, са прекалено „сложни" в сравнение с обема на обучаващите данни, използвани за тяхното пораждане. Както винаги, проблемът е в намирането на правилния баланс между сложността на отделните модели и степента, в която те обясняват данни.

Като един пример, илюстриращ това забележително свойство на метода boosting, да разгледаме поведението на AdaBoost върху една от тестови бази данни. Като алгоритъм за самообучение ще използваме един опростен вид на класификационни дървета, а именно дървета, които имат само един тест – в корена. Този модел на класификационни

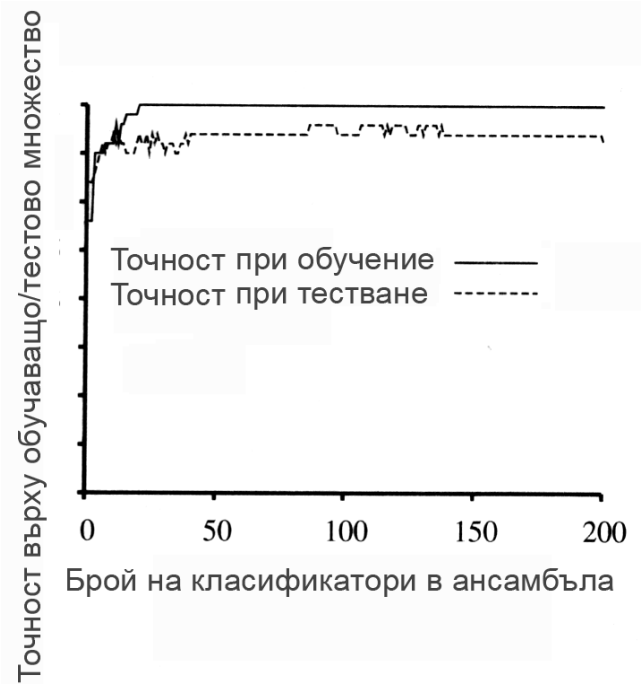
дървета се нарича *класификационни пънове (decision stumps)*. На Фиг. 17-3 са показани графики, отразяващи зависимостта на класификационната точност на научения класификатор, измерена върху тестови данни, от размера на обучаващото множество. Долната крива показва, че за тази база от данни класификаторът във вид на класификационния пън не е много ефективен, тъй като той постига максималната точност само от 81% върху тестови данни след обучение върху 100 обучаващи примера. Поведението на ансамбъла от 5 класификатора, получен след прилагане на метода AdaBoost, е много по-добро, тъй като при същия брой обучаващи примери (100) той постига точност от 93%.



Фиг. 17-3. Сравнение на точността върху тестовото множество на един класификационен пън и класификатор от 5 класификационни пънове, построен чрез boosting

В случаите, когато boosting успява да намали класификационната грешка върху нови данни, се наблюдава едно учудващо явление – изпълнението на нови итерации от процедурата на boosting (т.е. пораждане и добавяни на нови класификатори) може да намали грешката върху нови данни много след момента, когато грешката на ансамбъла върху обучаващи данни стана равна на нула. На Фиг. 17-4 са показани графики, отразяващи зависимостта на класификационната точност върху обучаващо и тестово множества за ансамбли, създадени чрез метода AdaBoost, като функция от броя на класификатори в ансамбъла. Пак в качеството на алгоритъма за самообучение е използван алгоритмът на класификационни пънове, обучаван върху 100 обучаващи примера от базата данни за ресторанти. Обърнете внимание, че ансамбълът, построен от 20 класификационни пънове, абсолютно точно класифицира всички 100 обучаващи примера. С нарастване на броя на пънове в ансамбъла класификационната грешка върху обучаващите примери остава равна на 0. Обаче, другата крива, отразяващата точността на ансамбъла върху тестови данни, показва че тази точност продължава да нараства и *след* момента на достигане на абсолютната точност при класифициране на обучаващите примери. Така при K = 20 точността е 95%, докато при K = 137 тя нараства до 98% след което постепенно се намалява.

Ефектът от нарастването на точността на ансамбъла върху тестови данни след постигането на перфектната класификация върху обучаващи данни се наблюдава както при различни бази от данни, така и при използване на различни алгоритми за самообучение. Този ефект много учуди изследователи, тъй като той изглежда като един аргумент *срещу* бръснача на Окам, който ни казва, че не трябва да правим хипотези по-сложни, отколкото е необходимо. Едно от възможни обяснения за това е, че boosting може да бъде разглеждан като една апроксимация към оптималния Бейсов класификатор.



Фиг. 17-4. Пропорция на правилно класифицирани обучаващи и тестови примери като функция от броя на класификатори в ансамбъла, построен чрез boosting

Обаче, как отбелязва Domingos (2012) по своята същност тези два метода за самообучение са много различни. Ансамблите променят пространството на хипотези (например от класификационни дървета към линейната комбинация от тях). Бейсовият класификатор назначава на хипотези от оригиналното пространство тегла по една фиксирана формула, като те са много различни от теглата, използвани при bagging и boosting. Другото възможно обяснение на наблюдавания ефект е, че добавянето на повече хипотези прави ансамбълът по-убеден в различаването на положителни и отрицателни примери, което води до повишаване на точността. С други думи ансамбълът повишава убедеността в своите предсказвания, като под убедеността се разбира разликата (margin) между вероятността на предсказания клас и вероятността на най-вероятния клас измежду останалите.

Обикновено boosting поражда класификатори, които са значително по-точни върху нови данни от тези, генерирани чрез bagging. Обаче в отличието от bagging, методът на boosting понякога пропада в някои практически ситуации – той може да генерира ансамбъл от класификатори, който е значително по-неточен от един единствен класификатор, породен от същите обучаващи данни. Това показва, че и комбинираният класификатор (т.е. ансамбъл) прекалено се нагажда към данни.

17.5 Stacking

Stacked generalization (обобщение чрез натрупване) или за краткост *stacking* (Wolpert 1992) е един друг начин за създаване на ансамбли. В отличие от методите bagging и boosting, stacking построява ансамбли от класификатори, изградени от *различни* алгоритми за самообучение. Да предположим, че искате да направите класификатор за определено множество от данни и разполагате с три алгоритъма за самообучение – класификационни дървета, наивен Бейсов класификатор и алгоритъм на k най-близки съседи. Обичайната процедура в такива случаи е да се приложи крос-валидацията и да се избере най-добрият класификатор от наличните. Обаче, дали това е най-добрия начин? Не можем ли ние да използваме и трите класификатора, като комбинираме техните предсказвания?

Един от начините да комбинираме изходите на тези класификатори е чрез гласуване – по същия начин, както и при bagging. Обаче непретегленото гласуване има смисъл само, ако използваните класификатори имат едно сравнително добро поведение. Ако два от трите класификатора правят доста некоректни предсказвания, то очевидно е, че поведението на ансамбъла ще бъде отчайващо. По тази причина stacking въвежда понятие *мета-алгоритъм за самообучение (metalearner)*, който заменя процедурата по гласуване. Основният проблем с гласуването е, че не е ясно, на кой от класификатори трябва да се вярва. Методът на stacking опитва *да научи*, кои от класификаторите са надеждни, използвайки за тази цел друг, мета-алгоритъм за самообучение, за да намери как по най-добрия начин да комбинира изходите на базовите класификатори.

Входът към мета-алгоритъма, който още се нарича мета-модел или *модел от ниво 1*, са предсказанията на базовите модели (или *модели от ниво 0*). Един пример на ниво 1 има толкова атрибути, колкото са модели от ниво 0, като стойността на всеки атрибут представлява класификация на конкретен пример, направена от съответния модел от ниво 0. Когато ансамбъл, направен по метода на stacking, се използва за класификация, новият пример отначало се подава на всяки от модели от ниво 0, които правят предсказания за неговата класификация. След това тези предсказани захранват входа на модела от ниво 1, кой ги комбинира, за да изведе крайната класификация на проверявания пример.

Как става обучение на модела от ниво 1? За тази цел трябва да намерим начин да трансформираме обучаващите данни от ниво 0 (чрез модели от ниво 0) в обучаващите данни от ниво 1 (които ще се използват са обучение на модел от ниво 1). Праволинейният подход изглежда много лесен – всеки модел от ниво 1 класифицира обучаващия пример, а след това към получения вектор от такива класификации добавим истинския клас на примера, като по този начин получаваме съответния обучаващ пример на ниво 1. За съжаление, този подход няма да работи добре. Той ще доведе до научаване на едни опростени правила от типа: *винаги вярвай на изхода на класификатора A и игнорирай тези на B и C*. Макар че подобни правила изглеждат подходящи за конкретни обучаващи данни, това не означава, че те ще работят добре и за тестови данни, тъй като ще предпочитат класификатори, които са прекалено нагодени към обучаващите данни, т.е. ще водят до преспецификацията.

Следователно, stacking не трябва просто да трансформира обучаващите данни от ниво 0 в данни от ниво 1. Един от възможните начини да се преборим с описания по-горе ефект от преспецификацията е да използваме вече ни познатия начин за разбиване на

наличните данни на два множества – обучаващото и потвърждаващото. В случая на stacking обучаващото множество се използва за обучение на модели от ниво 0, а потвърждаващото множество – за обучение на модела от ниво 1. С други думи, след като класификатори от ниво 0 са били построени от обучаващите данни, те се прилагат към потвърждаващото множество и техните предсказания формират обучаващото множество за класификатора от ниво 1. Тъй като потвърждаващите примери не са използвани за обучение на класификатори от ниво 0, получените предсказания са „безпристрастни" и дават по-добро приближение към поведение на съответните класификатори върху „свежи" данни. С други думи те по-акуратно отразяват истинското поведение на класификатори от ниво 0. След генериране на класификатор от ниво 1 от тези данни, използваните на ниво 0 алгоритми за самообучение могат отново да бъдат приложени вече към *всички* налични данни, като по-този начин ще бъдат породени базовите класификатори от ниво 0 с още по-добро предсказващо поведение.

Основният проблем с описания метод се състои в това, че той лишава модела от ниво 1 от една част от наличните обучаващи данни (тези, които са били използвани за обучение на модели от ниво 0). Решението може да се търси в използване на процедурата за крос-валидация, прилагана към всеки модел от ниво 0. Всеки пример от обучаващото множество участва точно един път в тестовите подмножества и предсказванията на класификатори от ниво 0, генерирани от съответните обучаващите подмножества, се използват за построяване от тях на съответния обучаващ пример за модел от ниво 0. По този начин за всеки обучаващ пример от ниво 0 се генерира съответния обучаващ пример от ниво 1. Естествено тази процедура е доста бавна, тъй като един класификатор от ниво 0 трябва да бъде трениран за всяко едно подмножество от крос-валидацията, но това позволява класификаторът от ниво 1 да използва напълно всички налични обучаващи данни.

Остава последния въпрос – какви алгоритми за самообучение са най-подходящи за използване в качеството на модели от ниво 1? По принцип, за тази цел може да бъде използван всеки алгоритъм за самообучение. Обаче, тъй като повечето от работата е вече свършена от алгоритмите, използвани на ниво 0, класификаторът от ниво 1 е на практика играе роля на арбитър и има смисъл за тази цел да бъде избран един сравнително прост алгоритъм. По думите на откривателя на stacking - David Wolpert, е резонно да се предположи, че алгоритмите за самообучение, създаващи относително глобални, гладки предсказващи модели, ще имат добро поведение в качеството на класификатори от ниво 1. Обикновено линейните модели или дървета с линейните модели в листата работят добре.

Макар че алгоритъм stacking за създаване на ансамбли е бил разработен малко по-рано от алгоритмите bagging и boosting, той не е толкова широко разпространен като последните. Причината за това се крие частично, от едната страна, в това, че stacking е труден за анализиране от теоретичната гледна точка, а от другата – че няма един общо признат най-добър начин за неговата реализация – базовата идея на stacking може да бъда приложена в множество различни вариации.