58147 Koneoppiminen

luennot keväällä 2005

Jyrki Kivinen

Koneoppiminen ja erityisesti laskennallinen oppimisteoria: perusteita ja viimeaikaisia tuloksia

- laudaturerikoiskurssi algoritmien erikoistumislinjalla, 4 ov, sopii muillekin linjoille sekä jatko-opiskelijoille

- hyödyllisiä esitietoja: matematiikkaa (lineaarialgebra, differentiaalilaskenta, todennäköisyyslaskenta), algoritmianalyysia (Laskennan teoria, Algoritmien suunnittelu ja analyysi)

- muita kursseja laitoksella: Tekoäly, Kolme käsitettä, *Tiedonlouhinnan menetelmät*, *Machine Learning for Bioinformatics*

Opetusmuodot ja suorittaminen:

- perinteinen kurssi: luennot, laskuharjoitukset, tentti
- tentti 50 pistettä, laskuharjoitukset 10 pistettä:

  0% tehtävistä antaa 0 pistettä
  80% tehtävistä antaa 10 pistettä
  välillä interpoloidaan lineaarisesti

- harjoitustehtävissä voi olla jotain pientä tietokoneella kokeilua
- luennoijaa **saa** häiritä kurssiin liittyvillä asioilla
- luennot -13.4., tentti 29.4.; tarkemmin opetusohjelma (tai kurssin kotisivu)

Materiaali:

- ei kurssikirjaa
- tenttissä oletetaan osattavaksi luennoilla ja harjoituksissa käsitelty materiaali sekä mahdollisesti osoitettava lisämateriaali
- luentomuistiinpanot tulevat kurssin edetessä saataville verkkoon ja kurssimappiin

Tavoitteena luoda yleiskuva laskennallisen oppimisteorian erään osa-alueen nykytilaan.

Peruskysymys (johon ei tosin saada kunnollista vastausta): miksi tietyntyyppiset algoritmit oppivat?

Sisältöluonnos:

1. Johdanto: koneoppiminen tieteenalana, koneoppimisen osa-alueet, luokittelijoiden oppimisen peruskäsitteet

2. Askeltava (online) oppiminen: asiantuntijamalli, lineaarinen oppiminen (perceptron, winnow), marginaali ja muut analyysitekniikat

3. Tilastollinen oppiminen (PAC ja yleistykset): perustuloksia, yleistysvirheanalyysia (VCdim, Rademacher), tilastollisen ja askeltavan oppimisen yhteys

4. Tukivektorikoneet (SVM): ydinfunktiot, optimointiongelma, marginaaleihin perustuva yleistysvirheanalyysi

5. Oppimisen tehostaminen (boosting): heikko vs. vahva oppiminen, AdaBoost ja muita aggregointialgoritmeja, marginaaleihin perustuva yleistysvirheanalyysi

1. Johdanto

Luvun tavoitteena on antaa hyvin karkea yleiskuva

- koneoppimisen osa-alueista ja asemasta tekoälyssä ja
- koneoppimisen suhteesta muihin lähitieteenaloihin

ja johdatella kurssin varsinaiseen asiaan eli luokittelijoiden oppimiseen:

- käsitteet ja luokittelijat
- tilastollinen (PAC) oppimismalli
- askeltava (online) oppimismalli

1.1 Koneoppiminen ja sen lähisukulaiset

Tämän kurssin kannalta oppiminen tarkoittaa toimia, joilla (älykäs?) agentti muuntaa toimiaan paremmin ympäristöä vastaaviksi. Koneoppimisessa oppiva agentti on tietokoneohjelma tai algoritmi.

Koneoppimista tarvitaan monentasoisissa tekoälyongelmissa, esim.

- semanttisesti merkityksellisten piirteiden tunnistaminen alhaisen tason havaintodatasta (esim. konenäkö),
- monimutkaisten loogisten käsitteiden rakentaminen yksinkertaisista (esim. asiantuntijajärjestelmän rakentaminen) ja
- oikeiden toimintasuunnitelmien löytäminen (pelit).

Historiallisesti "koneoppiminen" (*machine learning*) tarkoittaa tällaisista sovelluksista noussutta oppimisalgoritmien kehittämistä.

Oppimista on perinteisesti tarkasteltu monilla tieteenaloilla: psykologia, neurobiologia, kognitiotiede.

- tarkastellaan miten ihminen (ja muut organismit) todellisuudessa oppivat
- hyvin vaikea ongelma, vain tiettyjä perusasioita voidaan suoraan mitata laboratoriossa (mikä sekin on jo hyvin vaikeaa)
- tämä on **eri** asia kuin miten oppiminen olisi tehokasta tms.
- tarkasteltavat mallit silti usein relevantteja myös koneoppimisen kannalta (ainakin inspiraation lähteenä)
- tyypillisesti kuitenkaan koneoppimisessa käytetyt algoritmit eivät ole "biologisesti uskottavia" (huolimatta termeistä "neuroverkko" jne.)

Oppivia ohjelmia ja algoritmeja esiintyy myös aloilla, joilla termiä "oppiminen" ei yleensä käytetä:

Signaalinkäsittely:

- spesifit ongelmat ja rajoitteet ratkaisumenetelmille
- pitkälle viety teoria, vakiintuneita teknologioita
- läheinen yhteys *askeltavaan* (online) koneoppimiseen

Tilastotiede:

- matemaattisesti ongelma usein aivan sama kuin perinteisessä koneoppimisessa, tyypillisissä sovelluskohteissa eroja
- usein painopiste jonkin ilmiön esittämisessä ihmisen ymmärrettävässä muodossa
- laskentakapasiteetin kasvu mahdollistaa aivan uudenlaiset menetelmät

Tiedon louhinta (data mining):

- uudempi tulokas kuin edelliset
- lähtökohta tietokantatutkimuksessa: mitä tehdä olemassaoleville valtaville datamäärille?
- tavoitteena yleensä tuottaa ihmisen ymmärrettävää tietoa
- kysymyksenasettelu usein vapaamuotoinen (”mitä kiinnostavaa tästä datasta voi sanoa?”)

Näiden samansukuisten tieteen(osa)alojen kesken ei kannata turhaan etsiä vastakohtaisuuksia:

- kysymys enemmän painotuseroista kuin koulukuntaerimielisyyksistä
- hyväksi havaitut tekniikat siirtyvät alalta toiselle
- erilaiset taustat voivat kuitenkin aiheuttaa pahojakin kommunikointiongelmia

1.2 Ohjattu oppiminen

Peruskäsitteet ovat seuraavat:

Tapausavaruus: joukko X, esim. $X = \mathbf{R}^n$ tai $X = \{0,1\}^n$ jollain n. Alkiot $x \in X$ ovat tapauksia (instance).

Arvojoukko: joukko Y, esim. $Y = \{-1,1\}$, $Y = \{\text{'a'},\ldots,\text{'z'}\}$ tai $Y = \mathbf{R}$. Alkiot $y \in Y$ ovat arvoja (labels).

Opittava funktio on tuntematon funktio $f\colon X \to Y$.

Hypoteesi on oppimisalgoritmin tuottama funktio $h\colon X \to Y$.

Esimerkki on pari $(x,y) \in X \times Y$. Otos (sample) on jono esimerkkejä $((x_1,y_1),\ldots,(x_m,y_m)) \in (X \times Y)^m$ jollain m.

"Ohjattu" (supervised) viittaa siihen, että oppimisalgoritmin tehtäväksi on selvästi asetettu arvojen y muodostaminen tapauksen x perusteella.

Käsitteiden selventämiseksi kannattaa ajatella seuraavaa perustilannetta:

Syöte: otos jossa $y_t = f(x_t)$ opittavalla funktiolla f

Tuloste: hypoteesi h

Tavoite: $h(x) \approx f(x)$ "useimmilla" tapauksilla $x \in X$

Idea on, että otoskoko $\ll$ tapausavaruuden koko, jolloin tavoite edellyttää opittavan funktion approksimoimista myös otospisteiden ulkopuolella. Tällä abstraktiotasolla tehtävänä on siis oleellisesti funktion f ekstrapolointi.

Tavoite on tässä muotoiltu hyvin epämääräisesti, täsmällisempiin muotoiluihin palataan pian.

Käytännössä perusmallia muunnellaan paljon, erityisesti usein ei ole mielekästä olettaa mitään "oikeaa" opittavaa funktiota f.

Ohjattua oppimista jaotellaan edelleen arvojoukon Y mukaan:

Luokittelu: joukossa Y "muutamia" eri arvoja, joilla ei mitään oppimiseen vaikuttavia suhteita; esim. $Y = \{\text{'a'}, \ldots, \text{'z'}\}$ tai $Y = \{\text{ovi}, \text{ikkuna}, \text{seinä}\}$

Käsitteenoppiminen eli binääriluokittelu: edellisen erikoistapaus $Y = \{-1, 1\}$

Regressio: $Y = \mathbf{R}$ (tai $Y = [-R, R]$ jollain $R > 0$, tms.); usein käytetään neliövirhettä $(y - f(x))^2$

Tällä kurssilla keskitytään (binääri)luokitteluun.

Käsite on tapausavaruuden X osajoukko. Samastamme käsitteen $c \subseteq X$ ja binääriluokittelijan f missä

$$f(x) = \begin{cases} -1 & \text{jos } x \notin c \\ 1 & \text{jos } x \in c. \end{cases}$$

Tyypillisiä luokitteluongelmia

Esimerkkejä suosituista benchmark-ongelmista:

Suolisyövän tunnistaminen: Otoksessa 40 esimerkkiä kasvaimista ja 22 esimerkkiä terveestä kudoksesta (siis $m = 62$). Kussakin esimerkissä 2000 geenin ekspressiotaso (siis $X = \mathbf{R}^{2000}$). Saavutettuja virheprosentteja eri menetelmillä: LogitBoost 13%, päätöspuu 16%, tukivektorikone 10% (Dettling and Bühlmann 2002)

Uutissähkeiden luokittelu: Otoksessa noin 7300 uutissähkettä, joita kuvataan 93 eri luokalla (eivät toisensa poissulkevia); keskim. n. 80 sanaa/sähke. Saavutettuja "tarkkuuksia": AdaBoost 93% [3 päivää laskenta-aikaa], k neareast neighbour 92%, lineaarinen luokittelija 90% (Schapire and Singer 2000)

Postinumeroiden tunnistaminen: Otoksessa n. 7300 esimerkkiä, kukin 16×16 pikselimatriisi (siis $X = [-1, 1]^{256}$); luokat '0',. . . ,'9'. Saavutettuja virheprosentteja: ihminen 2,5%, viritetty tukivektorikone 3,2% (laskenta-aika 50 tuntia), neuroverkko 5,0%, nearest neighbour 5,9% (Decoste and Schölkopf 2002)

1.3 Käsiteluokat

Käsiteluokka C on mikä tahansa joukko käsitteitä; siis $C \subseteq \mathcal{P}(X)$ missä $\mathcal{P}$ tarkoittaa potenssijoukkoa.

Käsitteenoppimisongelmia ja -algoritmeja luokitellaan usein käsiteluokkien avulla:

- Voidaan olettaa, että opittava käsite f kuuluu johonkin annettuun käsiteluokkaan C.
- Hieman realistisemmin voidaan olettaa, että jokin annetun luokan C käsite on jossain mielessä "lähellä" opittavaa käsitettä.
- Oppimisalgoritmin hypoteesiluokka koostuu niistä käsitteistä, jotka algoritmi voi palauttaa hypoteesinaan.

Hypoteesiluokan yhteydessä joudutaan ottamaan kantaa myös siihen, miten käsitteet esitetään merkkijonoina. Toisinaan tämä on luokan koostumuksen nojalla ilmeistä, toisinaan ei.

Tarkastelemme seuraavaksi joitain tärkeitä käsiteluokkatyyppejä.

Propositiokäsitteet

Tapauksessa $X = \{ -1, 1 \}^n$ käsitteet voidaan luontevasti esittää propositiologiikan kaavoina (eli Boolen kaavoina).

- Kaava x_i esittää käsitettä $\{ \boldsymbol{x} \in X \mid x_i = 1 \}$.
- Jos f esittää käsitettä c, niin $\overline{f}$ esittää käsitettä $\overline{c} = X - c$.
- Jos f_1 ja f_2 esittävät käsitteitä c_1 ja c_2, niin $f_1 + f_2$ esittää käsitettä $c_1 \cup c_2$ ja $f_1 f_2$ esittää käsitettä $c_1 \cap c_2$

(Tässä perinteiset loogiset merkinnät $\neg f$, $f \vee f'$ ja $f \wedge f'$ on siis esitetty tiiviimmin $\overline{f}$, $f + f'$ ja $f f'$.)

Käytännössä usein samastamme kaavan ja sen esittämän käsitteen. Toisinaan kiinnitämme kuitenkin huomiota siihen, että samalla käsitteellä voi olla useita syntaktisesti erilaisia esityksiä.

Muotoa x_i tai $\overline{x_i}$ olevia kaavoja sanotaan literaaleiksi.

Konjunktio on muotoa $l_1 \ldots l_k$ ja disjunktio muotoa $l_1 + \ldots + l_k$, missä kaavat l_i ovat literaaleja.

Jos kaavassa ei ole lainkaan negaatioita, se on monotoninen.

Kaava on disjunktiivisessa normaalimuodossa (DNF), jos se on muotoa $f_1 + \ldots + f_l$, missä termit f_i ovat konjunktioita. Jos missään termissä ei ole yli k literaalia, em. kaava on l-termi-k-DNF-kaava.

Kaava on konjunktiivisessa normaalimuodossa (CNF), jos se on muotoa $f_1 \ldots f_l$, missä klausuulit f_i ovat disjunktioita. Jos missään klausuulissa ei ole yli k literaalia, em. kaava on l-klausuuli-k-CNF-kaava.

Mikä tahansa propositiokäsite voidaan esittää sekä DNF- että CNF-kaavana, mutta nämä normaalimuotoesitykset voivat olla paljon pitempiä kuin käsitteen lyhin mahdollinen esitys.

Päätöslistat

Jos $c_1, \ldots, c_m$ ovat tapausavaruuden X käsitteitä ja $l_1, \ldots, l_m, l_{m+1}$ luokkia, niin jono $((c_1, l_1), \ldots, (c_m, l_m), (\textbf{true}, l_{m+1}))$ on päätöslista (decision list).

Se esittää luokittelijaa f, missä $f(x)$ on l_i pienimmällä i, jolla $x \in c_i$, eli

if $x \in c_1$ **then** l_1
else if $x \in c_2$ **then** l_2
...
else if $x \in c_m$ **then** l_m
else l_{m+1}.

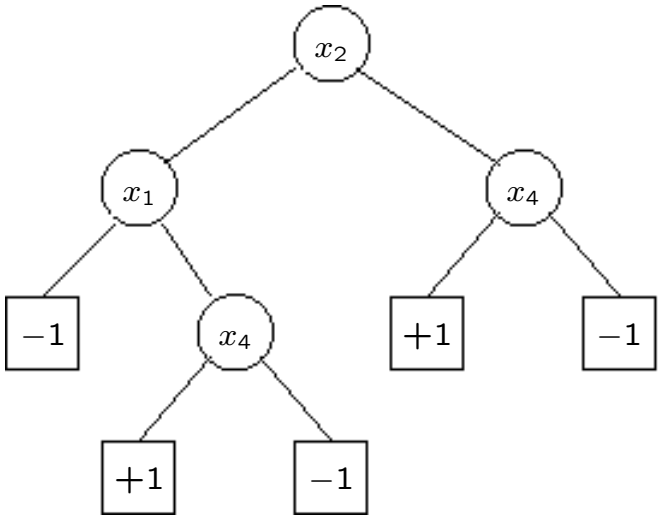
(Merkintä **true** esittää siis käsitettä X.)

Jos $X = \{-1, 1\}^n$ ja jokainen c_i on kork. k-literaalinen konjunktio, niin f on k-päätöslista eli kuuluu luokkaan k-DL.

Päätöslistat ovat erikoistapaus tekoälyssä yleisistä sääntöpohjaisista luokittelijoista.

Päätöspuut

Oheinen päätöspuu esittää käsitettä $x_2\overline{x_1}x_4 + \overline{x_2}x_4$.



Käsittely lähtee liikkeelle juuresta.
Sisäsolmussa testatataan yksittäistä muuttujaa ja mennään vasemmalle (oikealle) jos tosi (epätosi).
Kun päädytään lehteen, voidaan lukea oikea luokitus.

Päätöspuun käsite on helppo yleistää tapaukseen $X = X_1 \times \ldots \times X_n$, X_i äärellisiä, ja usealle luokalla. (Algoritmien yleistäminen ei aina ole triviaalia.)

Päätöspuut ovat toinen tekoälyssä perinteisesti käytetty esitystapa.

Lineaariset luokittelijat

Tässä $X \subseteq \mathbf{R}^n$, mikä sisältää tärkeän erikoistapauksen $X = \{ -1, 1 \}^n$, ja $Y = \{ -1, 1 \}$.

Jos $\boldsymbol{w} \in \mathbf{R}^n - \{ \mathbf{0} \}$ ja $b \in \mathbf{R}$, määritellään lineaarinen luokittelija

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - b)$$

missä $\mathrm{sign}(z) = 1$ jos $z \geq 0$ ja $\mathrm{sign}(z) = -1$ muuten, ja $\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{i=1}^{n} w_i x_i$. Vektoria $\boldsymbol{w}$ sanotaan painovektoriksi ja lukua b kynnysarvoksi (bias).

Luokittelijaa $f_{\boldsymbol{w},0}$ merkitään lyhyesti $f_{\boldsymbol{w}}$.

Otos $s = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (\mathbf{R}^n \times \{ -1, 1 \})^m$ on lineaarisesti erottuva (linearly separable), jos joillain $\boldsymbol{w}$ ja b pätee $y_i = f_{\boldsymbol{w},b}(\boldsymbol{x}_i)$ kaikilla i. Geometrisesti tämä tarkoittaa, että joukot $S_- = \{ x_i \mid y_i = -1 \}$ ja $S_+ = \{ x_i \mid y_i = 1 \}$ ovat hypertason $\{ \boldsymbol{x} \in \mathbf{R}^n \mid \boldsymbol{w} \cdot \boldsymbol{x} = b \}$ eri puolilla. Tällöin kyseisen hypertason (ja luokittelijan) sanotaan erottavan otoksen S.

Parin $(\boldsymbol{w}, b)$ normittamaton marginaali esimerkin $(\boldsymbol{x}, y)$ suhteen on luku $y(\boldsymbol{w} \cdot \boldsymbol{x} - b)$, ja normitettu marginaali on

$$\frac{y(\boldsymbol{w} \cdot \boldsymbol{x} - b)}{\|\boldsymbol{w}\|_2}$$

missä $\|\boldsymbol{w}\|_2 = (\sum_i w_i^2)^{1/2}$ on euklidinen normi.

Marginaali voidaan usein intuitiivisesti tulkita luokittelijan $f_{\boldsymbol{w},b}$ luottamukseksi siihen, että y on oikea luokka tapaukselle $\boldsymbol{x}$.

Huomaa, että (erikoistapausta $\boldsymbol{w} \cdot \boldsymbol{x} = b$ lukuunottamatta) marginaali on positiivinen, jos ja vain jos $y = f_{\boldsymbol{w},b}(x)$, ja normitetun marginaalin itseisarvo on pisteen $\boldsymbol{x}$ etäisyys hypertasosta $\boldsymbol{w} \cdot \boldsymbol{x} = b$.

Normitus on tärkeää myös siksi, että parit $(\boldsymbol{w}, b)$ ja $(\alpha\boldsymbol{w}, \alpha b)$ millä tahansa $\alpha > 0$ esittävät tasan samaa luokittelijaa, mutta jälkimmäisen normittamattomat marginaalit ovat α-kertaiset.

Jatkossa keskitytään tapaukseen $b = 0$, jolloin puhutaan painovektorin $\boldsymbol{w}$ marginaalista jne.

Kynnysarvo b voidaan usein sulauttaa osaksi painovektoria $\boldsymbol{w}$, jolloin sitä ei tarvitse erikseen ottaa huomioon oppimisalgoritmia:

- Oletetaan tunnetuksi algoritmi A, joka oppii muotoa $f_{\boldsymbol{w}}$ olevia luokittelijoita (siis kiinnitetty $b = 0$).
- Halutaan oppia otoksesta $s = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (\mathbf{R}^n \times \{-1, 1\})^m$ muotoa $f_{\boldsymbol{w},b}$ oleva luokittelija, missä **ei** haluta olettaa $b = 0$.
- Kun $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$, määritellään $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_n, 1) \in \mathbf{R}^{n+1}$, ja vastaavasti $\tilde{S} = ((\tilde{\boldsymbol{x}}_1, y_1), \ldots, (\tilde{\boldsymbol{x}}_m, y_m)) \in (\mathbf{R}^{n+1} \times \{-1, 1\})^m$.
- Sovelletaan algoritmia A otokseen $\tilde{S}$, saadaan painovektori $\tilde{\boldsymbol{w}} = (\tilde{w}_1, \ldots, \tilde{w}_n, \tilde{w}_{n+1}) \in \mathbf{R}^{n+1}$.
- Haluttu n-ulotteinen luokittelija $f_{\boldsymbol{w},b}$ saadaan valitsemalla $\boldsymbol{w} = (\tilde{w}_1, \ldots, \tilde{w}_n)$ ja $b = -\tilde{w}_{n+1}$.

Monet algoritmit kuitenkin käsittelevät kynnysarvon b erikseen, koska edelliseen palautukseen liittyy teoreettisia ja käytännöllisiä ongelmia:

- Painovektorin $\tilde{\boldsymbol{w}}$ normitettu marginaali ei ole sama kuin parin $(\boldsymbol{w}, b)$ marginaali, koska kynnysarvon b ei pitäisi olla mukana normituksessa.

- Usein olisi parempi valita $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_n, \alpha)$ jollain $\alpha \neq 1$, mutta oikean arvon α löytäminen ei ole helppoa.

Esimerkki 1.1: Tapausavaruuden $\{-1, 1\}^5$ propositiokäsite $x_1\overline{x_3}x_4$ voidaan esittää lineaarisena luokittelijana $f_{\boldsymbol{w},b}$, missä $\boldsymbol{w} = (1, 0, -1, 1, 0)$ ja $b = 2\frac{1}{2}$. Normitettu marginaali on $(1/2)/\sqrt{3} \approx 0,289$.

Toisaalta $x_1\overline{x_3}x_4 = f_{\tilde{\boldsymbol{w}}}(\tilde{\boldsymbol{x}})$, missä $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_5, 1)$ ja $\tilde{\boldsymbol{w}} = (1, 0, -1, 1, 0, -5/2)$. Normitettu marginaali on $(1/2)/\sqrt{11/2} \approx 0,213$.

Valitsemalla $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_5, 5/2)$ saataisiin $\tilde{\boldsymbol{w}} = (1, 0, -1, 1, 0, -1)$, joka tasapainoisempana *voisi* olla oppimisalgoritmille helpompi löytää.

1.4 Käsitteenoppimisen perusmallit

Johdattelevana (epärealistisena) esimerkkinä tilastolliseen oppimiseen tarkastellaan PAC-mallia (probably approximately correct; Valiant 1984) äärellisellä käsiteluokalla.

Oletetaan kiinnitetyksi jokin käsiteluokka $C \subseteq \mathcal{P}(X)$ ja opittava käsite $f_* \in C$ sekä joukon X todennäköisyysmitta P. Kuten tavallista, merkitsemme joukon $\{\, x \in X \mid \phi(x) \,\}$ todennäköisyyttä yksinkertaisesti $P(\phi(x))$.

Hypoteesin h virheeksi määritellään

$$\mathrm{err}_{P,f_*}(h) = P(h(x) \neq f_*(x)).$$

Yleensä P ja f_* ovat yhteydestä selviä ja jätetään merkitsemättä.

Merkinnällä $P(S)$, missä $S \subseteq (X \times Y)^m$, tarkoitetaan todennäköisyyttä tapahtumalle $s \in S$, kun otos $s = ((x_1, y_1), \ldots, (x_m, y_m))$ muodostetaan valitsemalla m tapausta x_i toisistaan riippumatta mitan P mukaan ja asettamalla $y_i = f_*(x_i)$. Jos m on yhteydestä selvä, se jätetään merkitsemättä.

Ajatus siis on, että virhettä mitataan samalla mitalla, jota esimerkit noudattavat.

Määritelmä 1.2 (PAC-oppiminen): Olkoon A jokin oppimisalgoritmi ja $A(s)$ sen hypoteesi otoksella s. Olkoon $0 < \varepsilon, \delta < 1$. Jos kaikilla $f_* \in C$ ja kaikilla P pätee

$$P^m(\text{err}(A(s)) > \varepsilon) \leq \delta,$$

niin sanomme että algoritmi A oppii käsiteluokan C tarkkuudella ε ja luottamuksella δ otoskoon ollessa m.

Määritelmässä on eroteltu kaksi erehtymismahdollisuutta:

- Todennäköisyydellä δ (ts. jos otos on hyvin epätyypillinen) hypoteesi voi olla mitä tahansa.

- Vaikka otos olisikin kunnollinen (todennäk. $1 - \delta$), hypoteesin sallitaan erehtyä pienessä osassa tapauksia (todennäköisyys ε).

Huomaa pahin tapaus käsitteen f_* ja mitan P suhteen.

Toisaan erotetaan vielä aito oppiminen, jossa vaaditaan $A(s) \in C$.

Luokittelija f on yhteensopiva otoksen $s = ((x_1, y_1), \ldots, (x_m, y_m))$ kanssa, jos $y_i = f(x_i)$ kaikilla i. Yhteensopivat käsitteet muodostavat versioavaruuden

$$V_C(s) = \{\, f \in C \mid f \text{ ja } s \text{ yhteensopivat} \,\}.$$

Tästä saadaan seuraava naiivi oppimisalgoritmi:

Algoritmi 1.3 (Versioavaruusalgoritmi):

1. Lue otos $s \in (X \times Y)^m$.
2. Jos $V_C(s) = \emptyset$, palauta **fail**.
3. Muuten palauta mikä tahansa $h \in V_C(s)$.

Lause 1.4: Jos C on äärellinen, niin kaikilla $0 < \varepsilon, \delta < 1$ ja kaikilla

$$m \geq \frac{1}{\varepsilon} \ln \frac{|C|}{\delta}$$

versioavaruusalgoritmi oppii käsiteluokan C tarkkuudella ε ja luottamuksella δ otoskoon ollessa m.

Jatkossa sanomme lyhyemmin "versioavaruusalgoritmi oppii luokan C otosvaativuudella (sample complexity) $(1/\varepsilon) \ln(|C|/\delta)$".

Todistus: Koska oletuksen mukaan $f_* \in C$, niin $f_* \in V_C(s)$ kaikilla s, joten algoritmi ei koskaan palauta **fail**.

Olkoon $h \in C$ yksi luokittelija, jolla $\mathrm{err}(h) > \varepsilon$. Jos $h \in V_C(s)$, niin otokseen s on sattunut m sellaista tapausta x_i, joilla $h(x_i) = y_i = f_*(x_i)$. Koska $P(h(x) = f_*(x)) < 1 - \varepsilon$, tämän todennäköisyys on korkeintaan $(1 - \varepsilon)^m$.

Koska sellaisia h, joilla $\mathrm{err}(h) > \varepsilon$, on korkeintaan $|C|$ kappaletta, niin todennäköisyys että jokin niistä on versioavaruudessa $V_C(s)$ on korkeintaan

$$|C|(1 - \varepsilon)^m.$$

Käyttämällä epäyhtälöä $(1 - x) \leq \mathrm{e}^{-x}$, joka pätee kaikille $x \in \mathbf{R}$, ja sijoittamalla otoskoolle m annettu alaraja, saadaan tämän ylärajaksi

$$|C|\mathrm{e}^{-\varepsilon m} \leq |C|\mathrm{e}^{\ln(\delta/|C|)} = \delta.$$

□

Esimerkki 1.5: Olkoon $X = \{-1, 1\}^n$ ja $C = \mathcal{P}(X)$. Siis kaikki käsitteet sallitaan. Koska $|C| = 2^{2^n}$, otosvaativuudeksi tulee

$$\frac{1}{\varepsilon} \ln \frac{2^{2^n}}{\delta} = \frac{1}{\varepsilon} \left((\ln 2) 2^n + \ln \frac{1}{\delta} \right)$$

eli tapausavaruuden koon suhteen lineaarinen. Tämä on yhteensopivaa sen intuition kanssa, että jos mitään ei oleteta, niin oppiminen ei ole mahdollista.

Esimerkki 1.6: Olkoon $X = \{-1, 1\}^n$ ja C kaikkien k-CNF-kaavojen joukko. Erilaisia k literaalin klausuuleja on korkeintaan $(2n)^k$. Siis niistä saatavia konjunktioita on korkeintaan $2^{(2n)^k}$. Otosvaativuudeksi tulee

$$\frac{1}{\varepsilon} \left((\ln 2)(2n)^k + \ln \frac{1}{\delta} \right)$$

eli muuttujien lukumäärän n suhteen polynominen, jos k on vakio.

Huom. käytännössä oletus "k on vakio" ei yleensä ole mielekäs, ja muutenkin tämäntyyppiset otoskokoarviot ovat toivottoman pessimistisiä.

Jotta tulokset olisi käytännön kannalta relevantteja, mallia pitää tarkentaa monin tavoin:

- Yleensä ei voida olettaa, että otos noudattaa tarkalleen jotain opittavaa käsitettä $f_* \in C$.
- Käsitteiden lukumäärä $|C|$ on liian karkea mitta käsiteluokan monimutkaisuudelle.
- Yksittäisessä käytännön sovelluksessa ei yleensä esiinny mitan P tai käsitteen f_* pahinta tapausta.
- Miten käsitellään moniluokkainen tapaus?

Palaamme näihin kysymyksiin jatkossa.

Jos halutaan hyödyllisiä numeerisia otoskokoarvioita todellisille sovelluksille, myös erilaiset yksityiskohdat pitää viilata huolellisesti. (Ja silti tämäntyyppisten arvioiden hyödyllisyys nykytekniikoilla on marginaalista.) Koska kysymys on todennäköisyyksistä, pelkät suuruusluokka-arviot eivät ole yhtä hyödyllisiä kuin algoritmianalyysissa yleensä.

Vaihtoehtona tilastolliselle oppimisella tarkastellaan askeltavaa (online) oppimista.

Oppiminen tapahtuu jonona kokeita (trial), joissa algoritmi vuorovaikuttaa ympäristön kanssa ja päivittää hypoteesia $h_t \colon X \to Y$.

Kokeessa numero t (tai "ajanhetkellä t") algoritmi

1. saa syötteenä tapauksen $x_t \in X$,
2. tulostaa ennusteen $\widehat{y}_t = h_t(x_t) \in Y$,
3. saa syöttenä oikean tuloksen $y \in Y$ ja
4. muodostaa seuraavan hypoteesin h_{t+1}.

Siis hypoteesi h_t määräytyy aiempina ajanhetkinä saadusta syötteestä $x_1, y_1, \ldots, x_{t-1}, y_{t-1}$. Lähtökohtana on jokin alkuhypoteesi h_1

Jos $\widehat{y}_t \neq y_t$, algoritmi teki virheen kokeessa t. Tavoitteena on luonnollisesti minimoida virheiden määrä sopivien oletusten vallitessa.

Erotuksena tilastolliseen malliin oppiminen ja testaaminen tapahtuvat limittäin, ei peräkkäin.

Analogisesti tilastollisen oppimisen versioavaruusalgoritmin kanssa lähdemme liikkeelle seuraavasta naiivista algoritmista annetulle käsiteluokalle C:

Algoritmi 1.7 (Puolitusalgoritmi):

Alusta $V_1 := C$.
Toista arvoilla $t = 1, \ldots, T$:

1. Lue $x_t \in X$.
2. $V^- := \{ h \in V_t \mid h(x_t) = -1 \}$ ja
   $V^+ := \{ h \in V_t \mid h(x_t) = 1 \}$
3. Jos $|V^-| > |V^+|$ niin $\widehat{y}_t = -1$
   muuten $\widehat{y}_t = 1$.
4. Lue y_t.
5. Jos $y_t = -1$ niin $V_{t+1} := V^-$, muuten $V := V^+$.

Lause 1.8: Jos jollakin $f_* \in C$ pätee $f_*(x_t) = y_t$ kaikilla t, niin puolitusalgoritmi tekee korkeintaan $\log_2 |C|$ virhettä.

Todistus: Algoritmin ennuste $\widehat{y}_t$ on sama kuin ainakin puolella versioavaruuden V_t hypoteeseista.

Siis jos algoritmi tekee virheen, ainakin puolet versioavaruudesta pyyhkiytyy pois ja $|V_{t+1}| \leq \frac{1}{2}|V_t|$.

Oletuksen mukaan $f_* \in V_t$ kaikilla t, joten $|V_t| \geq 1$. Siis virhe voi tapahtua korkeintaan $\log_2 |V_1| = \log_2 |C|$ kertaa. $\square$

Huom. sama virheraja pätee myös algoritmin konservatiiviselle versiolle, joka muuttaa joukkoa V_t vain, jos $y_t \neq \widehat{y}_t$.

Tähän tarkasteluun liittyy samoja ongelmia kuin versioavaruusalgoritmiin:

- Entä jos mikään $f \in C$ ei ole aina oikeassa?
- Miten voidaan ottaa huomioon käsiteluokan C koon lisäksi sen muu rakenne?
- Miten käsitellään moniluokkainen tapaus ja regressio?
- Kuinka hyvin pahimman tapauksen rajat kuvaavat algoritmin todellista käyttäytymistä?
- Miten algoritmi toteutetaan laskennallisesti tehokkaasti?

Lisäksi pitää miettiä, mikä on tilastollisen ja askeltavan oppimismallin suhde toisiinsa.

Palaamme näihin kysymyksiin pian.

1.5 Muunlaisia oppimismalleja

Ohjatun oppimisen lisäksi tärkeitä malleja ovat mm. ohjaamaton oppiminen (unsupervised learning) ja vahvistusoppiminen (reinforcement learning).

Ohjaamaton oppiminen: Syötteenä on jono tapauksia $(x_1, \ldots, x_m) \in X^m$, siis nyt ilman luokkia tms. Tyypillisesti $X \subseteq \mathbf{R}^n$, ja tavoitteena esim.

- ryvästäminen (clustering) eli pisteiden jakaminen ryppäisiin siten, että kukin rypäs sijaitsee pienellä alueella mutta ryppäät ovat kaukana toisistaan,
- dimension pienentäminen, jossa yritetään löytää dataan mahdollisimman hyvin sopiva alempiulotteinen osa-avaruus (esim. pääkomponenttianalyysi),
- löytää "pieni" osajoukko $S \subset X$ jolla kuitenkin on "suuri" todennäköisyys tai
- yleisemmin koko esimerkkien noudattaman todennäköisyysmitan approksimoiminen

Palauteoppiminen: Protokolla muistuttaa jossain määrin askeltavaa oppimista: hetkellä t algoritmi

1. on aluksi tilassa s_t,
2. tekee toiminnon a_t,
3. saa palautteen $r_t = R(s_t, a_t) \in \mathbf{R}$ ja
4. siirtyy tilaan $s_{t+1} = S(s_t, a_t)$.

Tavoitteena on maksimoida kokonaispalaute $\sum_{t=1}^{T} r_t$ pitkän ajanjakson yli.

Palautefunktio R ja siirtymäfunktio S voivat sisältää satunnaisuutta. Tila s_t ei välttämättä ole tiedossa, vaan siitä voi olla vain osittaisia havaintoja.

Ajatellaan esim. robotin navigointia sokkelossa.

Robotilla on jonkinlainen käsitys omasta sijainnistaan ja siitä, miten sen toiminnot tähän vaikuttavat.

Palaute voisi olla $+1$, kun robotti poistuu sokkelosta, ja 0 aina muulloin. Palaute voi siis oikeastaan koskea hyvinkin pitkää toimintajonoa, ja (eräs) ongelma on päättää, mitkä toiminnot todella edistivät tavoitteen saavuttamista.

2. Online learning

We study some of the basic algorithms of online learning:

- predicting with expert advice: Weighted Majority, Aggregating Algorithm
- linear prediction: perceptron, winnow

and relative loss bounds and related proof techniques:

- convergence analysis using potential function
- error estimates based on margins.

2.1 Predicting with expert advice

Suppose that for solving some online learning task we have available N experts $\mathcal{E}_i$, $i = 1, \ldots, N$. They can be any learning algorithms.

High-level solution by combining experts: at each time step,

1. Input $z_t \in X$, give to experts.
2. For all i let $x_{t,i}$ be the prediction of $\mathcal{E}_i$.
3. Predict with $\widehat{y}_t = h_t(\boldsymbol{x})$.
4. Input the correct answer y_t, give to experts.

The problem is to learn the right method $h_t(\cdot)$ of combining the expert predictions, when we do not know in advance which experts (if any) are actually good.

Example 2.1: The model includes learning a finite concept class as special case: for $C = \{ f_1, \ldots, f_k \}$, choose $N = k$ and $\mathcal{E}_i(z) = f_i(z)$ for all i.

Example 2.2: The task is to predict, whether it will rain in the afternoon. The experts might include forecasts in radio and newspapers, horoscope, birds' flight etc.

- The idea is to throw lots of different experts to the pool and hope that at least one performs well.
- The overall method will not be too sensitive to the presence of many bad experts.
- Since the goal is to just combine experts, we can from now on ignore the "original" input $z_t \in X$ (which we would not even know in the case of e.g. weather prediction).

(Almost) realistic examples

Task: managing a stock portfolio
Experts: "invest all money in stock x", for different stocks x

Task: disk spin-down in a laptop
Experts: "spin down after x seconds idle" for a set of values x

Task: virtual memory paging
Experts: RAND, FIFO, LIFO, LRU, MRU, ...

("Almost realistic" means that the algorithms we shall soon see have worked well in realistic simulations.)

Start with the case $Y = \{-1, 1\}$. Assume also expert predictions are binary: $x_{t,i} \in \{-1, 1\}$.

Algorithm 2.3 (Weighted Majority, WM): At time t, expert i has weight $w_{t,i} \in [0, 1]$. The algorithm has learning rate $0 < \beta < 1$ as parameter.

Initialise $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Get expert predictions $x_{t,i}$.
2. Calculate $W_t^- = \sum_{x_{t,i}=-1} w_{t,i}$ and $W_t^+ = \sum_{x_{t,i}=1} w_{t,i}$.
3. If $W_t^- > W_t^+$ then $\hat{y}_t = -1$, else $\hat{y}_t = 1$
4. For $i = 1, \ldots, n$:
   if $x_{t,i} = y_t$ then $w_{t+1,i} = w_{t,i}$
   else $w_{t+1,i} = \beta w_{t,i}$.

Hence, the prediction of the master algorithm is the weighted majority vote of the experts. Each mistaken expert loses a fraction $1 - \beta$ of its weight. (Case $\beta = 0$ gives the Halving Algorithm.)

Consider the potential function $P_t = c \ln W_t$, where $W_t = W_t^+ + W_t^- = \sum_i w_{t,i}$ it the total remaining weight and $c > 0$ a constant to be fixed later.

Define the discrete or 0-1 loss function:

$$L_{0-1}(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise.} \end{cases}$$

This will help to unify notation.

Lemma 2.4: If

$$c = \left(\ln \frac{2}{1+\beta} \right)^{-1},$$

then for all t we have

$$L_{0-1}(y_t, \hat{y}_t) \leq c \ln \frac{W_t}{W_{t+1}} = -(P_{t+1} - P_t).$$

That is, whenever the master algorithm makes a mistake, the potential decreases by at least 1.

Proof of Lemma: Since always $W_t \geq W_{t+1}$, it is sufficient to consider the case $y_t \neq \widehat{y}_t$.

Take for example $\widehat{y}_t = -1$ and $y_t = 1$. Then $W_t^+ \leq W_t/2$, and we get

$$\begin{aligned} W_{t+1} &= W_t^+ + \beta W_t^- \\ &= (1-\beta)W_t^+ + \beta(W_t^+ + W_t^-) \\ &\leq (1-\beta)\frac{W_t}{2} + \beta W_t, \end{aligned}$$

and the claim follows. □

We see directly from the algorithm that $P_1 = c \ln N$, and P_t never increases. If additionally we make assumptions about the experts, we can also lower bound the potential (similar to amortized analysis of algorithms).

Define the total loss both for the algorithm and the experts:

$$L_{0-1}(\mathrm{WM}) = \sum_{t=1}^{T} L_{0-1}(y_t, \widehat{y}_t) \quad \text{and} \quad L_{0-1}(\mathcal{E}_i) = \sum_{t=1}^{T} L_{0-1}(y_t, x_{t,i}).$$

(Later we shall apply the same definition to other algorithms and loss functions.)

Theorem 2.5: For all experts $\mathcal{E}_i$ we have

$$L_{0-1}(\mathrm{WM}) \leq c\eta L_{0-1}(\mathcal{E}_i) + c \ln N,$$

where

$$c = \left(\ln \frac{2}{1+\beta}\right)^{-1} \quad \text{and} \quad \eta = \ln \frac{1}{\beta}.$$

Naturally the bound is tightest if we compare against the best expert, i.e. choose $i = \arg\min_j L_{0-1}(\mathcal{E}_j)$.

We can optimise the bound by tuning β:

- slow learning:

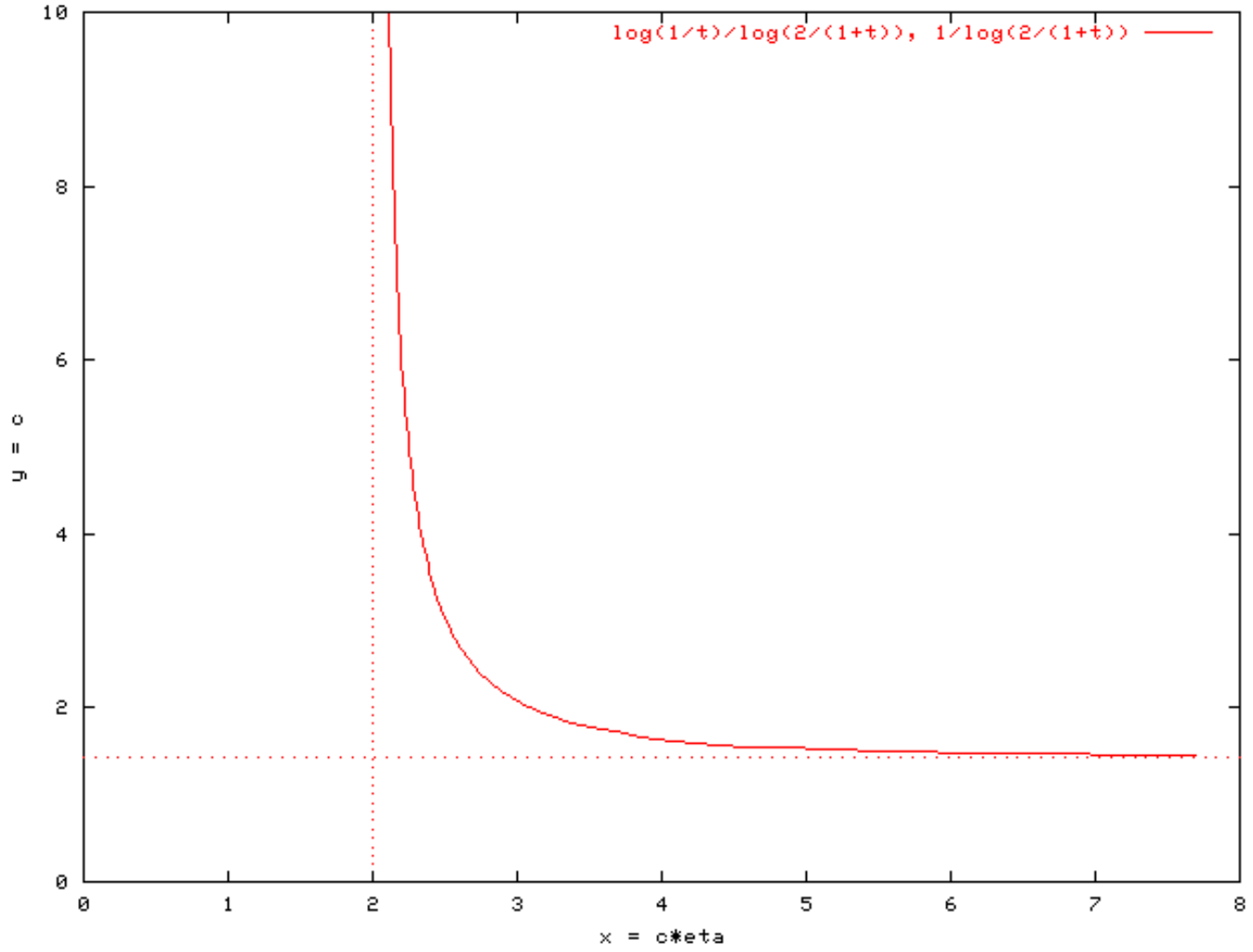
$$\lim_{\beta \to 1-} c\eta = 2 \quad \text{and} \quad \lim_{\beta \to 1-} c = \infty.$$

- fast learning:

$$\lim_{\beta \to 0+} c\eta = \infty \quad \text{and} \quad \lim_{\beta \to 0+} c = \frac{1}{\ln 2} \approx 1{,}44.$$

$\Rightarrow$ if all experts are bad, choose $\beta \approx 1$;
in one expert is really good, choose $\beta \approx 0$

Choosing the optimal β requires either (usually unrealistic) advance knowledge (how much is $\min_i L_{0-1}(\mathcal{E}_i)$) or complicated tuning "on the fly". We shall return to this.



Curve $(x, y) = (c\eta, c)$, $0 < \beta < 1$; asymptotes $x = 2$ and $y = 1/\ln 2$.

Proof of Theorem: By summing the estimates from previous lemma for $t = 1, \ldots, T$ we get

$$\begin{aligned} L_{0-1}(\mathrm{WM}) &\leq \sum_{t=1}^{T} -(P_{t+1} - P_t) \\ &= P_1 - P_{T+1} \\ &= c \ln N - c \ln W_{T+1}. \end{aligned}$$

The claim follows, since for all i we have

$$\begin{aligned} W_{T+1} &\geq w_{t+1,i} \\ &= \beta^{L_{0-1}(\mathcal{E}_i)} \\ &= \exp(-\eta L_{0-1}(\mathcal{E}_i)). \end{aligned}$$

□

The bound above is valid even if a malicious adversary chooses

- experts $\mathcal{E}_i$ and
- correct answers y_t

so that the task becomes as hard as possible.

Under such assumptions, the bound cannot be subtantially improved:

Theorem 2.6:

For any algorithm A there are arbitrarily large values N and M for which it is possible that

- there are N experts and
- the best expert makes at most M mistakes but
- the algorithm A makes at least

$$2M + \log_2 \frac{N}{2} = 2M + \frac{1}{\ln 2} \ln N - 1$$

mistakes.

Proof: Let $N = 2^{k+1}$ for some $k \geq 1$, and $T = 2M + k$. Choose T predictions $x_{t,i}$ for N experts as follows:

- For $1 \leq t \leq k$ let $x_{t,i} = x_{t,i+2^k} = 1$ if bit t in the binary representation of i is 1, and $x_{t,i} = x_{t,i+2^k} = -1$ otherwise.
- For $k+1 \leq t \leq T$ let $x_{t,i} = 1$ and $x_{t,i+2^k} = -1$.

Now for any sequence of answers $(y_1, \ldots, y_T)$

- there is one i such that both $\mathcal{E}_i$ and $\mathcal{E}_{i+2^k}$ predict the first k answers correctly and
- for all i at least one of $\mathcal{E}_i$ or $\mathcal{E}_{i+2^k}$ get at least half the answers $k+1, \ldots, T$ right.

Hence, the best expert makes at most M mistakes.

On the other hand, for given A and $\mathcal{E}_i$ we can always pick an answer sequence such that A always makes a mistake (take $y_t = -\widehat{y}_t$). $\square$

Example 2.7: $k = 2$, $n = 2^{k+1} = 8$, $M = 3$; $T = k + 2M = 8$.

The predictions of the experts are as follows:

t	1	2	3	4	5	6	7	8
$\mathcal{E}_1$	-1	-1	1	1	1	1	1	1
$\mathcal{E}_2$	1	-1	1	1	1	1	1	1
$\mathcal{E}_3$	-1	1	1	1	1	1	1	1
$\mathcal{E}_4$	1	1	1	1	1	1	1	1
$\mathcal{E}_5$	-1	-1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_6$	1	-1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_7$	-1	1	-1	-1	-1	-1	-1	-1
$\mathcal{E}_8$	1	1	-1	-1	-1	-1	-1	-1

Suppose the correct answers are

$$(y_1, \ldots, y_8) = (1, -1, 1, -1, -1, -1, -1, 1).$$

After 2 rounds, $\mathcal{E}_2$ and $\mathcal{E}_6$ have no mistakes.

After the whole sequence, $\mathcal{E}_6$ has done $2 \leq M$ mistakes (and $\mathcal{E}_2$ $4 = 2M - 2$ mistakes).

Thus, our upper bound is fairly tight in worst case. How realistic is "worst case"?

Intuitively the term $(\ln 2)^{-1} \ln N = \log_2 N$ is right: to find the best out of N experts we need at least $\log_2 N$ bits of information.

On the other hand, the factor 2 in $2L_{0-1}(\mathcal{E}_i)$ seems extraneous. In our lower bound it clearly is a result of looking very hard for a worst-case y_t.

For the learning result, the factor 2 implies that even with large amounts of data the algorithm is not guaranteed to converge towards the best expert, which is bad.

To address this issue, we introduce some continuous-valued loss functions that enable more subtle techniques.

Continuous-valued loss functions

Square loss typically used in regression:

$$L_{\mathsf{sq}}(y, \widehat{y}) = (y - \widehat{y})^2.$$

Logarithmic loss which is central in information theory:

$$L_{\mathsf{log}}(y, \widehat{y}) = \frac{1-y}{2} \ln \frac{1-y}{1-\widehat{y}} + \frac{1+y}{2} \ln \frac{1+y}{1+\widehat{y}}.$$

Hellinger loss that is useful for some statistical estimates but here mainly as an unusual example:

$$L_{\mathsf{H}}(y, \widehat{y}) = \frac{1}{2} \left((\sqrt{1-y} - \sqrt{1-\widehat{y}})^2 + (\sqrt{1+y} - \sqrt{1+\widehat{y}})^2 \right).$$

Absolute loss is (as we shall soon see) a probabilistic version of the discrete loss:

$$L_{\mathsf{abs}}(y, \widehat{y}) = \frac{1}{2} |y - \widehat{y}| .$$

Here always $-1 \leq y, \widehat{y} \leq 1$, and as usual $0 \log(1/0) = 0 \log 0 = 0$.

All loss functions on previous page are obtained as follows:

- Interpret the prediction $\widehat{y} \in [-1, 1]$ as a probability measure P over $\{-1, 1\}$, with $P(1) = (1 + \widehat{y})/2$ and $P(-1) = (1 - \widehat{y})/2$.
- Similarly interpret y as measure Q.
- Pick some distance measure d used for probability measure and take $L(y, \widehat{y}) = d(Q, P)$.

In particular, $L_{\text{abs}}(y, \widehat{y})$ gives the expected number of mistakes if we allow randomised predictions:

1. The algorithm gives a prediction $\widehat{y} \in [-1, 1]$.
2. The correct answer $y \in \{-1, 1\}$ is fixed.
3. The final prediction of the algorithm is picked from $\{-1, 1\}$ according to the measure P defined above.

Algoritmi 2.8 (Weighted Average, WA): Given: loss function L, learning rate $\eta > 0$

Initialize $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Receive expert predictions $x_{t,i}$.
2. For all i let $v_{t,i} = w_{t,i}/W_t$ where $W_t = \sum_{i=1}^{N} w_{t,i}$.
3. Predict $\widehat{y}_t = \boldsymbol{v}_t \cdot \boldsymbol{x}_t$.
4. For all i update $w_{t+1,i} = w_{t,i} \exp(-\eta L(y_t, x_{t,i}))$.

As for WM (with $\beta = e^{-\eta}$ and $L = L_{0-1}$), we have

$$w_{t,i} = w_{1,i} \exp\left(-\eta \sum_{j=1}^{t-1} L(y_j, x_{j,i})\right).$$

The weights also have a Bayesian interpretation (but we shall not use that in what follows).

Suppose we have a parametric density $P(y \mid x)$. Choose $L(y, x) = -\eta \ln P(y \mid x)$. Let $\mathcal{E}_i$ be the expert (i.e. model) whose prediction for y_t is distributed according to $P(\cdot \mid x_{t,i})$. Pick a prior $P(\mathcal{E}_i)$ over the experts. Now

$$\begin{aligned} P(\mathcal{E}_i \mid y_1, \ldots, y_t) &\propto P(y_1, \ldots, y_t \mid \mathcal{E}_i) P(\mathcal{E}_i) \\ &= P(\mathcal{E}_i) \prod_{j=1}^{t} P(y_j \mid x_{j,i}) \\ &= P(\mathcal{E}_i) \exp\left(-\eta \sum_{j=1}^{t} L(y_j, x_{j,i}) \right) \end{aligned}$$

so the normalised weights $v_{t,i}$ are directly the posterior probabilities of the experts, assuming we initialise $w_{i,1} \propto P(\mathcal{E}_i)$.

But not all loss functions can be represented as log-likelihoods.

We derive a loss bound in two steps:

1. Find $\eta > 0$ and $c > 0$ such that always
$$L(y_t, \widehat{y}_t) \leq c \ln \frac{W_t}{W_{t+1}}.$$
2. Sum over $t = 1, \ldots, T$.

Step 2 is easy and essentially the same as for WM.

Step 1 is technical and depends on the specifics of each loss function. Fortunately, there is a general solution for a large class of loss functions.

Remark Step 2 does not assume $\widehat{y}_t = \boldsymbol{v}_t \cdot \boldsymbol{x}_t$. In fact, a different choice of $\widehat{y}$ may implement Step 1 with better constants. This leads to the final version of the algorithm, the Aggregating Algorithm.

Theorem 2.9: Let $\eta >$ and $c > 0$ be such that

$$L(y_t, \boldsymbol{v}_t \cdot \boldsymbol{x}_t) \leq c \ln \frac{W_t}{W_{t+1}}. \qquad (*)$$

holds for all t. Then

$$L(\mathrm{WA}) \leq c\eta \min_{1 \leq i \leq N} L(\mathcal{E}_i) + c \ln N.$$

Proof: As for the WM algorithm, we get

$$L(\mathrm{WA}) \leq \sum_{t=1}^{T} c(\ln W_t - \ln W_{t+1}) = c \ln W_1 - c \ln W_{T+1}.$$

Since we chose $w_{1,i} = 1$ for all i, we can then estimate

$$\begin{aligned} -\ln W_{T+1} &\leq -\ln \left(\max_{1 \leq i \leq N} w_{t+1,i} \right) \\ &= \min_{1 \leq i \leq N} (-\ln w_{1,i} \exp(-\eta L(\mathcal{E}_i))) \\ &= \eta \min_{1 \leq i \leq N} L(\mathcal{E}_i). \end{aligned}$$

□

Our goal is to show that for certain continuous-valued L we can make the condition above hold for values c that are about half of that for L_{0-1}.

Define $L_y(x) = L(y, x)$; in particular $L'_y(x) = \partial L(y, x)/\partial x$. We always assume that L_{-1} is increasing and L_1 is decreasing.

Lemma 2.10: Assume that L is convex and twice differentiable. Then if $c \geq \tilde{c}_L$ where

$$\tilde{c}_L = \max_{y \in \{-1,1\}} \sup_{-1 \leq x \leq 1} \frac{L'_y(x)^2}{L''_y(x)},$$

the WA algorithm satisfies the condition $(*)$ for $\eta = 1/c$.

Remarks:

- If we replace $\max_{y \in \{-1,1\}}$ by $\sup_{-1 \leq y \leq 1}$, the bound holds for $-1 \leq y_t \leq 1$ (and not just $y_t \in \{-1, 1\}$). The value $\tilde{c}_L$ usually remains the same.
- By choosing $\eta = 1/\tilde{c}_L$ we get $L(\text{WA}) \leq \min_{1 \leq i \leq N} L(\mathcal{E}_i) + \tilde{c}_L \ln N$.
- The absolute loss does **not** satisfy the conditions. For that we need the more general Aggregating Algorithm.

The value $\tilde{c}_L$ is usually easy to calculate.

Example 2.11: Take $L = L_{\mathsf{sq}}$, so $L_y(x) = (y-x)^2$. We get

$$\frac{L'_y(x)^2}{L''_y(x)} = \frac{(2(x-y))^2}{2} = 2(x-y)^2,$$

so clearly

$$\max_{y \in \{-1,1\}} \sup_{-1 \le x \le 1} \frac{L'_y(x)^2}{L''_y(x)} = \sup_{-1 \le y \le 1} \sup_{-1 \le x \le 1} \frac{L'_y(x)^2}{L''_y(x)} = 8.$$

We can even allow $x, y \in [-R, R]$ for any $R > 0$, and the constant becomes $8R^2$.

Example 2.12: For logarithmic loss we have $\tilde{c}_L = 1$, for the Hellinger loss $\tilde{c}_L = 2$ (proof left as exercise). This is also the case if we take sup over $-1 \le y \le 1$. (In the case $-1 \le y \le 1$, log loss is often called relative entropy or Kullback-Leibler divergence.)

Proof of Lemma: Let $\eta = 1/c$. Rewrite the claim as

$$L(y_t, \boldsymbol{v}_t \cdot \boldsymbol{x}_t) \leq -c \ln \frac{\sum_{i=1}^{N} w_{t,i} \exp(-L(y_t, x_{t,i})/c)}{W_t} = -c \ln \sum_{i=1}^{N} v_{t,i} \exp(-L(y_t, x_{t,i})/c),$$

or equivalently

$$\exp(-L(y_t, \boldsymbol{v}_t \cdot \boldsymbol{x}_t)/c) \geq \sum_{i=1}^{N} v_{t,i} \exp(-L(y_t, x_{t,i})/c).$$

Defining $f(x) = \exp(-L(y_t, x)/c)$, this becomes

$$f\left(\sum_{i=1}^{N} v_{t,i} x_{t,i}\right) \geq \sum_{i=1}^{N} v_{t,i} f\left(x_{t,i}\right).$$

We show that f is concave, so this follows from Jensen's inequality (see below).

To complete the proof, we show $f''(x) \leq 0$:

$$\begin{aligned} f'(x) &= -\frac{1}{c}\frac{\partial L(y_t, x)}{\partial x} f(x) \\ f''(x) &= \left(-\frac{1}{c}\frac{\partial^2 L(y_t, x)}{\partial x^2} + \left(-\frac{1}{c}\frac{\partial L(y_t, x}{\partial x} \right)^2 \right) f(x) \end{aligned}$$

and since $f(x)$ and $L''(x)$ are positive, $f''(x)$ is negative if

$$c \geq \frac{\left(\frac{\partial L(y_t, x)}{\partial x}\right)^2}{\frac{\partial^2 L(y_t, x)}{\partial x^2}}.$$

Thus f is concave for $c \geq \tilde{c}_L$. □

The proof also shows that if $c < \tilde{c}_L$, then for $\eta = 1/c$ the bound $(*)$ fails for some combination of y_t, $\boldsymbol{v}_t$ and $\boldsymbol{x}_t$.

Theorem 2.13 (Jensen's Inequality): Let X be an arbitrary real-valued random variable and f a convex real-valued function defined on an interval that includes the range of X (which need not be bounded). Then

$$f(\mathrm{E}X) \leq \mathrm{E}f(X)$$

where E denotes expectation.

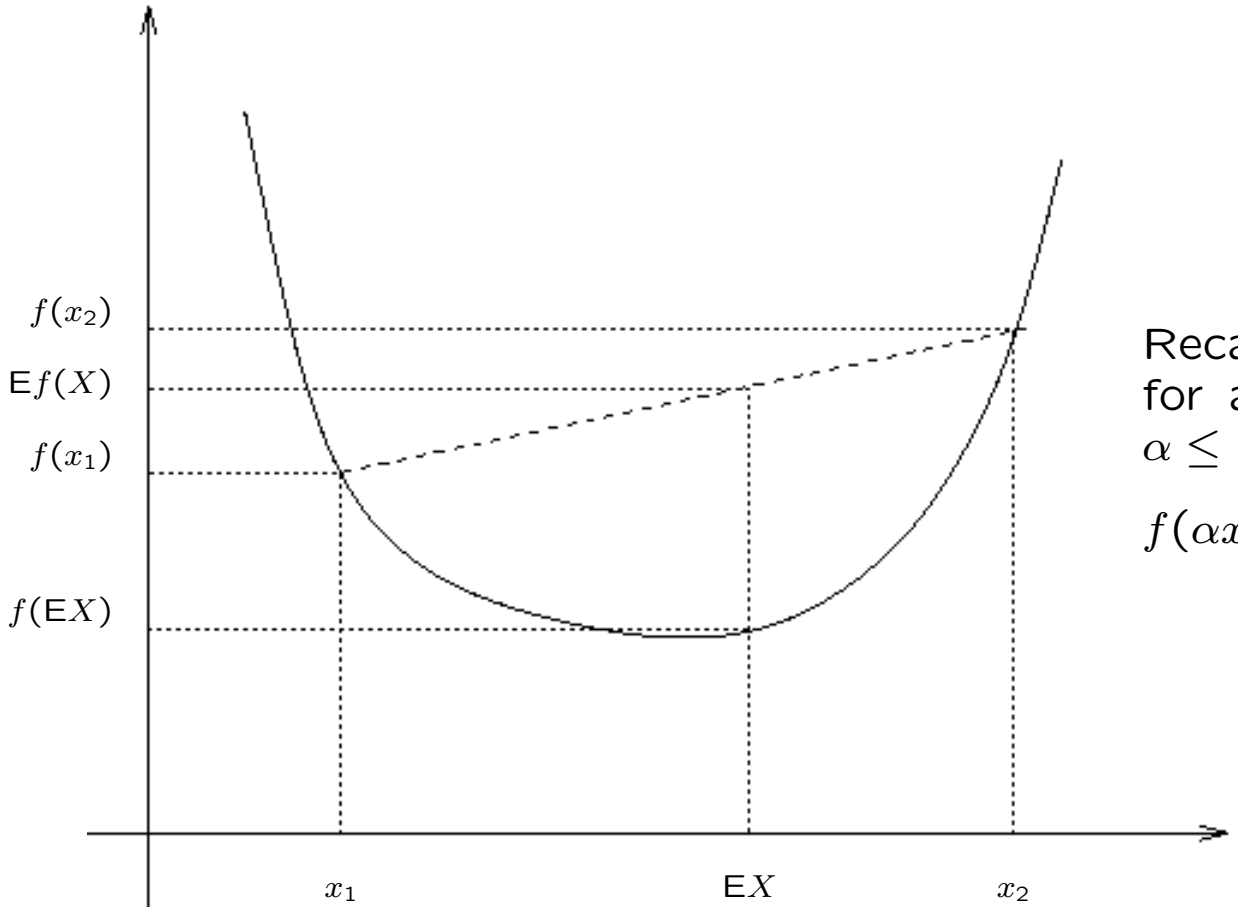
"Proof": The case where X has two possible values x_1 and x_2 follows directly from the definition of convexity. (See picture on next page.) More general case by induction and continuity. □

Corollary 2.14: If f is convex then

$$f\left(\sum_i v_i x_i\right) \leq \sum_i v_i f(x_i)$$

for all $\boldsymbol{x}$ and $\boldsymbol{v}$ such that $\sum_i v_i = 1$ and $v_i \geq 0$. □

If f is concave, the inequality flips the other way (and the best way to remember the right way is to remember the picture on next page).



Recall that f is convex if for all x_1 and x_2 and $0 \leq \alpha \leq 1$ we have

$$f(\alpha x_1 + (1-\alpha)x_2) \leq \alpha f(x_1) + (1-\alpha) f(x_2).$$

Corollary 2.15: Assume L that is convex and twice differentiable. Then for $\eta = 1/\tilde{c}_L$ the Weighted Average algorithm achieves

$$L(\mathrm{WA}) \leq \min_{1 \leq i \leq N} L(\mathcal{E}_i) + \tilde{c}_L \ln N.$$

For discrete loss, WM has

$$L_{0-1}(\mathrm{WM}) \leq c\eta \min_{1 \leq i \leq N} L_{0-1}(\mathcal{E}_i) + c \ln N$$

where $c\eta > 2$. To bridge this gap, we use absolute loss and the Aggregating Algorithm.

Algorithm 2.16 (Aggregating Algorithm, AA):

Given: loss function L, learning rate $\eta > 0$, "fudge factor" $c > 0$

Initialise $w_{1,i} = 1$ for all i.
Repeat for $t = 1, \ldots, T$:

1. Receive all expert predictions $x_{t,i}$.
2. Let $W_t = \sum_{i=1}^{N} w_{t,i}$ and, for $y \in \{-1, 1\}$, $W_t(y) = \sum_{i=1}^{N} w_{t,i} e^{-\eta L(y, x_{t,i})}$.
3. Predict with any $\widehat{y}_t$ satisfying $L(y, \widehat{y}_t) \leq c \ln(W_t / W_{t+1}(y))$ both for $y = -1$ and $y = 1$. (If no such $\widehat{y}_t$ exists, **fail**.)
4. Receive y_t and for all i let $w_{t+1,i} = w_{t,i} \exp(-\eta L(y_t, x_{t,i}))$.

The rule for choosing $\widehat{y}_t$ can easily be written out explicitly for any (sensible) L (example: square loss on next page).

As before, we have $w_{t,i} = w_{1,i} \exp\left(-\eta \sum_{j=1}^{t-1} L(y_j, x_{j,i})\right)$. However there is more freedom in choosing $\widehat{y}_t$.

Like with WM and WA, *if* the algorithm never fails we get

$$L(\mathrm{AA}) \leq c\eta \min_{1 \leq i \leq N} L(\mathcal{E}_i) + c \ln N.$$

Example 2.17: Aggregating Algorithm for square loss: Let $L = L_{\text{sq}}$.

First compute

$$\Delta(y) = c \ln \frac{W_t}{W_{t+1}(y)} = -c \sum_{i=1}^{N} \ln \frac{w_{t,i}}{W_t} \exp(-\eta(y - x_{t,i})^2)$$

for $y = -1$ and $y = 1$.

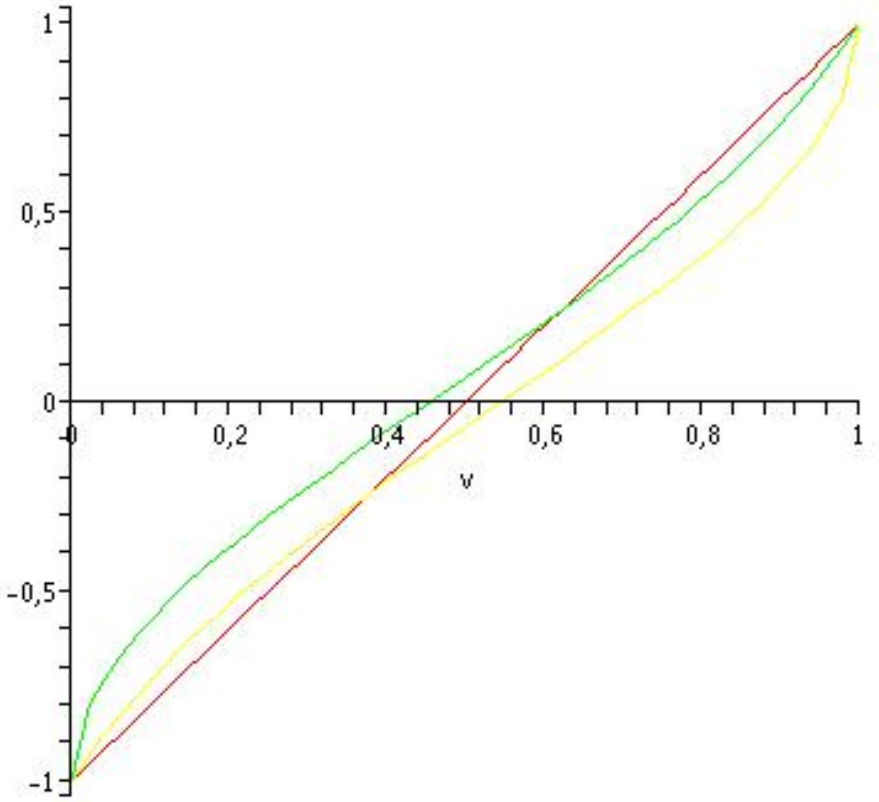
Since $L(-1, z) = (1 + z)^2$ and $L(1, z) = (1 - z)^2$, we require $(1 + z)^2 \leq \Delta(-1)$ and $(1 - z)^2 \leq \Delta(1)$, which becomes

$$1 - \sqrt{\Delta(1)} \leq z \leq -1 + \sqrt{\Delta(-1)}.$$

Thus a natural choice would be

$$\frac{1}{2}\left(\sqrt{\Delta(-1)} - \sqrt{\Delta(1)}\right)$$

assuming we have $\sqrt{\Delta(-1)} + \sqrt{\Delta(1)} \geq 2$. The tricky part is determining η and c such that this is guaranteed for all $\boldsymbol{x}_t$ and $\boldsymbol{w}_t$. It turns out that $\eta = 1/2$, $c = 2$ is a good choice.



Bounds for allowed predictions for square loss, $\eta = 1/2$, $c = 2$, $\boldsymbol{x} = (-1, 1)$, $\boldsymbol{w} = (1-v, v)$, for $v \in [0, 1]$.

green curve: upper bound
yellow curve: lower bound
red line: WA prediction

Notice that in general the WA prediction $\boldsymbol{v} \cdot \boldsymbol{x}$ is *not* in the legal range.

Lemma 2.18: Assume L is convex and twice differentiable and define

$$c_L = \sup_{0 \le x \le 1} \frac{L'_{-1}(x) L'_1(x)^2 - L'_1(x) L'_{-1}(x)^2}{L'_{-1}(x) L''_1(x) - L'_1(x) L''_{-1}(x)}.$$

Then AA with $c \ge c_L$ and $\eta = 1/c$ will never fail.

(Proof omitted.)

Corollary 2.19: For $c = c_L$ and $\eta = 1/c_L$ the Aggregating Algorithm achieves

$$L(\mathrm{AA}) \le \min_{1 \le i \le N} L(\mathcal{E}_i) + c_L \ln N.$$

Remark: The bound is optimal in the sense that there are sequences such that for *any* algorithm A we have

$$L(A) \ge \min_{1 \le i \le N} L(\mathcal{E}_i) + c_L \ln N - o(1)$$

where $o(1)$ goes to zero as T and N increase.

For "nice" loss functions we now have the choice of WA and AA, the latter with a more complicated prediction but better constant c_L instead of $\tilde{c}_L$:

square loss:	$c_L = 2$	$\tilde{c}_L = 8$
log loss:	$c_L = 1$	$\tilde{c}_L = 1$
Hellinger loss:	$c_L = \sqrt{2}$	$\tilde{c}_L = 2$

In general $c_L \leq \tilde{c}_L$ can be seen from the inequality

$$\frac{a+b}{a'+b'} \leq \max\left\{\frac{a}{a'}, \frac{b}{b'}\right\}$$

that holds for all positive a, b, a', b'.

So what about absolute loss?

Lemma 2.20: Choose any $\eta > 0$ and

$$c \geq \left(2 \ln \frac{2}{1 + e^{-\eta}} \right)^{-1}.$$

Then the Aggregating Algorithm for absolute loss never fails.

Corollary 2.21: The Aggregating Algorithm for absolute loss achieves

$$L_{\text{abs}}(\text{AA}) \leq \frac{\eta}{2 \ln 2/(1 + e^{-\eta})} \min_{1 \leq i \leq N} L_{\text{abs}}(\mathcal{E}_i) + \frac{1}{2 \ln 2/(1 + e^{-\eta})} \ln N.$$

Comparing this to

$$L_{0-1}(\text{WM}) \leq \frac{\eta}{\ln 2/(1 + e^{-\eta})} \min_{1 \leq i \leq N} L_{0-1}(\mathcal{E}_i) + \frac{1}{\ln 2/(1 + e^{-\eta})} \ln N$$

and remembering the interpretation of absolute loss as expected number of mistakes, we see that allowing randomised predictions saves a factor 2 in the loss bound.

Proof of Lemma: Write $\beta = e^{-\eta}$. Since the function $z \mapsto \beta^z$ is convex, for $0 \le z \le 1$ we have

$$\beta^{(1-z)a+zb} \le (1-z)\beta^a + z\beta^b$$

for all $a, b \in \mathbf{R}$. Choosing $a = 0$ and $b = 1$ gives us

$$\beta^z \le 1 - z + z\beta.$$

Define $\Delta(-1) = c\ln(W_t/W_{t+1}(y))$ and write $\boldsymbol{v} = \boldsymbol{w}_t/W_t$. Then $\sum_i v_i = 1$.

Since $L(-1, x) = (1+x)/2 \le 1$ for $-1 \le x \le 1$, we have

$$\begin{aligned}
\Delta(-1) &= -c\ln\sum_{i=1}^{N} v_i\beta^{(1+x_{t,i})/2} \\
&\ge -c\ln\sum_{i=1}^{N} v_i(1 + \frac{1+x_{t,i}}{2}(\beta-1)) \\
&= -c\ln(1 + r(\beta-1))
\end{aligned}$$

where $r = (1 + \boldsymbol{v}\cdot\boldsymbol{x}_t)/2$. As $L(1, x) = 1 - L(-1, x)$, the same estimate gives us

$$\Delta(1) \ge -c\ln(1 + (1-r)(\beta-1)).$$

We need $L(y,\widehat{y}) \leq \Delta(y)$ for $y \in \{-1,1\}$, which means

$$\frac{1+\widehat{y}}{2} \leq \Delta(-1) \quad \text{and} \quad \frac{1-\widehat{y}}{2} \leq \Delta(1).$$

This becomes $1-2\Delta(1) \leq \widehat{y} \leq -1+2\Delta(-1)$, so a valid prediction exists if $\Delta(-1)+\Delta(1) \geq 1$. Using the above estimates, and then Jensen, we get

$$\begin{aligned}
\Delta(-1)+\Delta(1) &\geq -2c\frac{\ln(1+r(\beta-1))+\ln(1+(1-r)(\beta-1))}{2} \\
&\geq -2c\ln\frac{1+r(\beta-1)+1+(1-r)(\beta-1)}{2} \\
&= 2c\ln\frac{2}{1+\beta}
\end{aligned}$$

so it is sufficient to have

$$c \geq \left(2\ln\frac{2}{1+\beta}\right)^{-1}.$$

□

Choosing a good η is subtle.

Theorem 2.22: Define $h(z) = 1 + 2\sqrt{z} + z/\ln 2$. If K is such that $\min_i \mathcal{E}_i \leq K$ and $\eta = \ln h((\ln N)/K)$, then

$$L_{\text{abs}}(\text{AA}) \leq \min_{1 \leq i \leq N} L_{\text{abs}}(\mathcal{E}_i) + \sqrt{K \ln N} + \frac{\ln N}{2 \ln 2}.$$

(Proof: Plug in value of η to previous corollary and crank; details omitted.)

Notice that η needs to be fixed at the beginning, so we cannot simply pick $K = \min_i \mathcal{E}_i$.

If T is known in advance, we can take $K = T$, but this is very pessimistic.

Thus, we would prefer a scheme that tunes η dynamically as time goes by. Such schemes exist, but they tend to be complicated to analyse.

Intuitively we want a large learning rate if N is "large" or $\min_i \mathcal{E}_i$ is "small".

The traditional way of dealing with not knowing the value $\min_i L_{\text{abs}}(\mathcal{E}_i)$ in advance is the so-called "doubling trick":

Choose suitable $a > 0$ and $c > 1$.
for $z := 0$ **to** ∞ **do**
  $k_z := ac^z$
  $b_z := k_z + \sqrt{k_z \ln N} + \ln N/(2 \ln 2)$
  $\eta_z := \ln h((\ln N)/k_z)$
  Set all expert weights to 1. Set loss count to 0.
  repeat
    predict with AA using $\eta = \eta_z$
  until loss count exceeds b_z
od

The idea is to make exponentially increasing guesses k_z about what $\min_i L_{\text{abs}}(\mathcal{E}_i)$ might be.

By the previous theorem, if the loss during any stage exceeds b_z, then all experts made loss larger than k_z during that stage.

Thus, after the outer loop has run for Z iterations, we know that

$$\min_{1 \le i \le N} L_{\text{abs}}(\mathcal{E}_i) \ge \sum_{z=0}^{Z} k_z.$$

Doing the "doubling trick" carefully gets us around the problem of having to know $\min_i L_{\text{abs}}(\mathcal{E}_i)$ in advance at the cost of somewhat worse constants.

Theorem 2.23: For a suitable choice of a and c, the "doubling AA" of previous page achieves

$$L_{\text{abs}}(\text{dAA}) \leq K_* + p\sqrt{K_* \ln N} + q \ln N$$

where $K_* = \min_i L_{\text{abs}}(\mathcal{E}_i)$ and p and q are absolute constants.

(Proof is straightforward but a bit involved; omitted.)

A similar analysis can be done for Weighted Majority and 0-1 loss.

The practical value of this result is unclear. The algorithm throws away lots of information by resetting the weights after each phase.

A more promising approach is to use at time t learning rate $\eta_t = \ln h((\ln N)/K_t)$ where

$$K_t = \min_{1 \leq 1 \leq N} \sum_{j=1}^{t-1} L_{\text{abs}}(y_j, x_{j,i})$$

is the current estimate of the loss of the best expert.

Even with the best tuning, the bound for absolute loss has an additional $\sqrt{\min_i L(\mathcal{E}_i)}$ term compared to square etc. losses. This is unavoidable:

Theorem 2.24: For any algorithm A there are sequences of expert predictions $\boldsymbol{x}_t$ and answers y_t such that

$$L_{\text{abs}}(A) \geq \min_{1 \leq i \leq N} L_{\text{abs}}(\mathcal{E}_i) + \Omega(\sqrt{T}).$$

Proof sketch: We construct a probability measure over $\boldsymbol{x}_t$ and y_t for which

$$\mathrm{E} L_{\text{abs}}(A) \geq \mathrm{E}\left(\min_{1 \leq i \leq N} L_{\text{abs}}(\mathcal{E}_i)\right) + \Omega(\sqrt{T})$$

where E denotes expectation w.r.t. this measure. This certainly implies the claim.

We make each $x_{t,i}$ and y_t to be -1 with probability $1/2$ and 1 with probability $1/2$, all independent. Clearly for any algorithm, and any given expert i, we have $\mathrm{E} L_{\text{abs}}(A) = \mathrm{E} L_{\text{abs}}(\mathcal{E}_i) = T/2$.

More specifically, $L_{\text{abs}}(\mathcal{E}_i)$ is a binomial random variable with mean $T/2$ and variance $\sqrt{T}/2$. Define a normalised version

$$F_i = \frac{L_{\text{abs}}(\mathcal{E}_i) - T/2}{\sqrt{T}/2}.$$

By Central Limit Theorem, for large T the random variable F_i has approximately normal distribution with mean 0 and variance 1.

A known result about a minimum of N independent normal random variables implies

$$\mathsf{E}\left(\min_{1\le i\le N} F_i\right) \approx -\sqrt{2\ln N}.$$

Thus

$$\mathsf{E}\left(\min_{1\le i\le N} L_{\text{abs}}(\mathcal{E}_i)\right) \approx T/2 - \sqrt{T\ln N/2}$$

from which the claim follows. "□"

Notice that for square loss we would have $\mathsf{E}L_{\text{sq}}(\mathcal{E}_i) = 2T$ for the above distribution, whereas $\mathsf{E}L_{\text{sq}}(A) = T$ for $\widehat{y}_t = 0$, so this would not work.

To summarise the results, suppose $L(\mathcal{E}_i) = \sigma_i T$ for all i. Thus, expert i incurs loss at an average rate σ_i. Let $\sigma_* = \min_i \sigma_i$.

For square, log, Hellinger etc. "nice" loss functions, we have

$$\frac{L(\mathrm{AA})}{T} \leq \sigma_* + \frac{c \ln N}{T}.$$

Thus, the average loss of AA convergest to the best average loss at rate $O(T^{-1})$. The same holds for WA with a worse constant c.

For absolute loss, AA has convergence rate $O(T^{-1/2})$:

$$\frac{L(\mathrm{AA})}{T} \leq \sigma_* + \frac{p\sqrt{\sigma_* \log N}}{\sqrt{T}} + \frac{q \ln N}{T}.$$

For 0-1 loss, WM converges to at most *twice* the optimal at rate $O(T^{-1/2})$:

$$\frac{L(\mathrm{WM})}{T} \leq 2\sigma_* + \frac{p\sqrt{\sigma_* \log N}}{\sqrt{T}} + \frac{q \ln N}{T}.$$

Unfortunately the theoretically nice loss functions are not always the ones we are really interested in.

2.2 Linear online classification

Here $X \subseteq \mathbf{R}^n$ for some n, $Y = \{-1, 1\}$, and the hypothesis of the learner at time t consist of a weight vector $\boldsymbol{w}_t \in \mathbf{R}^n$ and a threshold value (or bias) $b_t \in \mathbf{R}$.

Thus, the general scheme for a linear online prediction algorithm is as follows.

Set some initial values $\boldsymbol{w}_1$ and b_1.
For $t = 1, \ldots, T$ do the following:

1. Get the instance $\boldsymbol{x}_t \in \mathbf{R}^n$.
2. Predict $\widehat{y}_t = \mathrm{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t - b_t) \in \{-1, 1\}$.
3. Get the correct answer $y_t \in \{-1, 1\}$.
4. Update $(\boldsymbol{w}_t, b_t)$ to $(\boldsymbol{w}_{t+1}, b_{t+1})$.

As noted earlier, we can avoid handling the bias separately by replacing each $\boldsymbol{x}_t$ by $(x_{t,1}, \ldots, x_{t,n}, 1) \in \mathbf{R}^{n+1}$. Thus, without (much) loss of generality, we take $b_t = 0$ for all t (unless we specifically say othewise).

Weight $\boldsymbol{w}_t$ may depend on anything that has happened at times $1, \ldots, t-1$. The initial weight vector $\boldsymbol{w}_1$ is typically $\mathbf{0} = (0, \ldots, 0)$, or sometimes $\mathbf{1} = (1, \ldots, 1)$.

The quantity $y_t(\boldsymbol{w}_t \cdot \boldsymbol{x}_t - b_t)$ is the (unnormalised) margin of the algorithm at time t. Define

$$\sigma_t = \begin{cases} 0 & \text{if } y_t(\boldsymbol{w}_t \cdot \boldsymbol{x}_t - b_t) > 0 \\ 1 & \text{otherwise.} \end{cases}$$

If $\sigma_t = 1$, we say the algorithm made a mistake at time t.

Strictly speaking the prediction is correct also when $\boldsymbol{w}_t \cdot \boldsymbol{x}_t - b_t = 0$ and $y_t = 1$. Considering this special case separately would not actually improve our theoretical mistake bounds. In practice, it would be very unwise to trust on having the margin exactly zero.

Thus, we aim at having strictly positive margins, and analyse algorithms in terms of the number of mistakes

$$\sum_{t=1}^{T} \sigma_t.$$

The Perceptron Algorithm

The Perceptron Algorithm is attributed to Rosenblatt [1957] who was interested in learning in biological neurons.

In our setting, the Perceptron Algorithm is given by the update rule

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \sigma_t y_t \boldsymbol{x}_t.$$

We initialise $\boldsymbol{w}_1 = \mathbf{0}$. (Again, here $b_t = 0$ is fixed.)

If there was no mistake, there is no change in weights.

If there was a mistake, the update increases the margin (assuming $\boldsymbol{x}_t \neq \mathbf{0}$):

$$\begin{aligned} y_t \boldsymbol{w}_{t+1} \cdot \boldsymbol{x}_t &= y_t(\boldsymbol{w}_t + y_t \boldsymbol{x}_t) \cdot \boldsymbol{x}_t \\ &= y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t + \|\boldsymbol{x}_t\|_2^2 \\ &> y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \end{aligned}$$

where $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x} \cdot \boldsymbol{x}}$ is the Euclidean norm. Thus, were the same example encountered immediately again, the algorithm would at least be "less wrong".

A more careful analysis gives a mistake bound originally due to Novikoff [1962].

Define the normalised margin of a weight vector $\boldsymbol{u}$ with respect to $(\boldsymbol{x}_t, y_t)$ as

$$\gamma_t(\boldsymbol{u}) = \frac{y_t \boldsymbol{u} \cdot \boldsymbol{x}_t}{\|\boldsymbol{u}\|_2},$$

and let $\gamma(\boldsymbol{u}) = \min_t \gamma_t(\boldsymbol{u})$. Thus if $\gamma(\boldsymbol{u}) > 0$, then an algorithm using $\boldsymbol{w}_t = \boldsymbol{u}$ for all t would make no mistakes.

Theorem 2.25 (Perceptron Convergence Theorem): Let $X > 0$ be such that $\|\boldsymbol{x}_t\|_2 \leq X$ for all t, and let $\alpha > 0$. If $\gamma(\boldsymbol{u}) \geq \alpha$ for some $\boldsymbol{u} \in \mathbf{R}^n$, then the number of mistakes made by the Perceptron Algorithm has the upper bound

$$\sum_{t=1}^{T} \sigma_t \leq \frac{X^2}{\alpha^2}.$$

Notice that the bound does not depend on T. We could even allow infinite sequences.

Before the proof, consider briefly the implications.

Suppose we have a sample of m examples

$$s = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \in (\mathbf{R}^n \times \{-1, 1\})^m.$$

For $k = 1, 2, 3, \ldots$, let s_k be the sample of mk examples obtained by taking k copies of s and putting them one after another.

Running the Perceptron on s_k, if there ever are m consecutive time steps during which no mistake was made, the algorithm has learned to predict all the examples correctly. Hence there will be no more updates, and the algorithm has converged.

Suppose some $\boldsymbol{u}$ has margin at least $\alpha > 0$ on s. The margin is of course the same for s_k, for all k. If

$$r = \frac{\max_t \|\boldsymbol{x}_t\|_2^2}{\alpha^2},$$

then for any k the Perceptron Algorithm makes at most r mistakes on s_k.

In particular, if we take $k = r + 1$, we know that in s_k there must be some sequence of m consecutive examples during which there were no mistakes.

Hence, after repeating the sample at most $r + 1$ times the algorithm has converged to a consistent hypothesis ($\boldsymbol{w}_t$ such that $y_j \boldsymbol{w}_t \cdot \boldsymbol{x}_j > 0$ for all j).

The previous remark is related to applying online algorithms to the statistical learning setting. We shall return to this later in more detail. For now, just some brief remarks:

- Linear classifiers in high-dimensional spaces are a very rich concept class. In modern machine learning applications it is common to have $n \gg m$. In this case running the Perceptron through the sample sufficiently many times **will** produce a consistent hypothesis, **no matter what** the actual data is (barring some pathological special cases). This is an example of overfitting.

- Various methods exist to avoid overfitting. They include early stopping and keeping the weights "small" by regularisation.

- Algorithmically, if the margin is small then the problem of finding a consistent linear classifier may be more efficiently solved by linear programming.

Proof of the Perceptron Convergence Theorem: Without loss of generality we can take $\|\boldsymbol{u}\|_2 = 1$. Then $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \alpha$ for all t.

The idea is to show that $\boldsymbol{w}_t$ converges towards $c\boldsymbol{u}$ where $c > 0$ is a suitable constant. (Recall that $\boldsymbol{u}$ and $c\boldsymbol{u}$ for $c > 0$ define the same classifier.)

We define a potential function

$$P_t = \frac{1}{2}\|c\boldsymbol{u} - \boldsymbol{w}_t\|_2^2,$$

where $c > 0$ is to be fixed later. Initially $P_1 = c^2\|\boldsymbol{u}\|_2^2/2 = c^2/2$. Always $P_t \geq 0$.

Next we lower bound the drop of potential at time t as

$$P_t - P_{t+1} \geq \left(c\alpha - \frac{X^2}{2}\right)\sigma_t.$$

For $\sigma_t = 0$ the claim is clear. Assume $\sigma_t = 1$, so $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq 0$.

By simply writing the squared norms as dot products it is easy to verify

$$\frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}\|_2^2 - \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}'\|_2^2 = (\boldsymbol{w}' - \boldsymbol{w}) \cdot (\boldsymbol{u} - \boldsymbol{w}) - \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{w}'\|_2^2.$$

for all $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{w}' \in \mathbf{R}^n$. (Notice that since the dot product on the right-hand side can be negative, this shows that the squared Euclidean distance does not satisfy the triangle inequality.)

By plugging in $\boldsymbol{u} := c\boldsymbol{u}$, $\boldsymbol{w} := \boldsymbol{w}_t$ and $\boldsymbol{w}' := \boldsymbol{w}_{t+1}$, and noticing $\boldsymbol{w}_{t+1} - \boldsymbol{w}_t = y_t \boldsymbol{x}_t$, we get

$$P_t - P_{t+1} = y_t \boldsymbol{x}_t \cdot (c\boldsymbol{u} - \boldsymbol{w}_t) - \frac{1}{2}\|\boldsymbol{x}_t\|_2^2.$$

Since $\|\boldsymbol{x}_t\|_2 \leq X$, $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \alpha$ and $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq 0$, we get

$$P_t - P_{t+1} \geq c\alpha - \frac{X^2}{2}.$$

By summing over t we get

$$\begin{aligned}\frac{c^2}{2} &\geq P_1 - P_{T+1} \\ &\geq \sum_{t=1}^{T} (P_t - P_{t+1}) \\ &\geq \left(c\alpha - \frac{X^2}{2}\right) \sum_{t=1}^{T} \sigma_t\end{aligned}$$

which for $c \geq X^2/(2\alpha)$ becomes

$$\sum_{t=1}^{T} \sigma_t \leq \frac{c^2}{2c\alpha - X^2}.$$

We get the desired bound by choosing $c = X^2/\alpha$. (A straightforward differentiation shows that this choice of c maximises

$$\frac{2c\alpha - X^2}{c^2}$$

and thus gives the best bound.) $\square$

To get some geometrical intuition, denote by $\varphi(\boldsymbol{u}, \boldsymbol{x})$ the angle between vectors $\boldsymbol{u}$ and $\boldsymbol{x}$. That is,

$$\cos\varphi(\boldsymbol{u}, \boldsymbol{x}) = \frac{\boldsymbol{u} \cdot \boldsymbol{x}}{\|\boldsymbol{u}\|_2 \|\boldsymbol{x}\|_2}.$$

Do the standard simplification trick of replacing $(\boldsymbol{x}_t, y_t)$ by $(\tilde{\boldsymbol{x}}_t, 1)$ where $\tilde{\boldsymbol{x}}_t = y_t \boldsymbol{x}_t$. The condition $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \alpha$ then becomes $\boldsymbol{u} \cdot \tilde{\boldsymbol{x}}_t \geq \alpha$.

For simplicity, consider the special case $\|\boldsymbol{x}_t\|_2 = 1$ for all t. The condition becomes $\cos\varphi(\boldsymbol{u}, \tilde{\boldsymbol{x}}_t) \geq \alpha$. Thus for all $\tilde{\boldsymbol{x}}_t$ we have

$$\varphi(\boldsymbol{u}, \tilde{\boldsymbol{x}}_t) \leq \arccos\alpha = \frac{\pi}{2} - \theta$$

for some constant $\theta > 0$. All the $\tilde{\boldsymbol{x}}_t$ are in a certain cone opening around the vector $\boldsymbol{u}$ in angle $\pi/2 - \theta$.

Suppose we now simply want to find any $\boldsymbol{w}$ such that $\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_t > 0$ for all t. Thus any positive margin is acceptable. Then we can pick any $\boldsymbol{w}$ in the interior of the cone opening in an angle θ around $\boldsymbol{u}$.

The idea of the Perceptron Convercence Theorem is that every mistake twists $\boldsymbol{w}_t$ towards $\boldsymbol{u}$ by a fixed amount, and after a finite number of mistakes $\boldsymbol{w}_t$ is in the cone and no further mistakes occur.

Example 2.26: Let g be a k-literal conjunction over n variables. That is,

$$g(\boldsymbol{x}) = \tilde{x}_{i_1} \dots \tilde{x}_{i_k},$$

where $\tilde{x}_j$ is either x_j or $\overline{x_j}$.

For $j = 1, \dots, k$, define $u_{i_j} = 1$ if $\tilde{x}_j = x_j$ and $u_{i_j} = -1$ if $\tilde{x}_j = \overline{x_j}$. Further, introduce an extra variable x_{n+1} that is always set to 1, and let $u_{n+1} = -k + 1$. For all other i we set $u_i = 0$. Then

$$g(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n+1} u_i x_i\right)$$

and this linear classifier has unnormalised margin 1. Since $\|\boldsymbol{u}\|_2 = \sqrt{k \cdot 1^2 + (n-k) \cdot 0^2 + (k-1)^2}$, the normalised margin is $(2k^2 - 2k + 1)^{-1/2}$.

Assume now that the sequence $((\boldsymbol{x}_t, y_t)) \in (\{-1, 1\}^n \times \{-1, 1\})^T$ is such that $y_t = g(\boldsymbol{x}_t)$ for all t. Then $\|\boldsymbol{x}_t\|_2^2 = n + 1$ for all t. Plugging this and the margin to the Perceptron Convergence Theorem, we get

$$\sum_{i=1}^{T} \sigma_t \leq n(2k^2 - 2k + 1).$$

In practical applications, we of course never know that the target is a k-literal conjunction. Nevertheless bounds like this do give useful intuition about what affects the performance of the algorithm.

Notice in particular that the mistake bound is linear in N even if the target is extremely simple, say $k = 1$. By experimenting, we can see that this is how the algorithm actually behaves, not just some loose upper bound. □(Example)

Feature mappings

Consider learning classifiers over instance base X. In principle a feature map is any function $\boldsymbol{\psi}\colon X \to \mathbf{R}^r$, for some r. The r components $\psi_i(x)$ are called features of x, and $\mathbf{R}^r$ is the feature space.

Given a feature map ψ and a linear prediction algorithm A (e.g. the Perceptron), we can learn classifiers $X \to \{-1, 1\}$ by doing the following at time t:

1. Get input x_t.
2. Give $\boldsymbol{\psi}(x_t)$ as input to A.
3. Get prediction $\widehat{y}_t$ from A and give it as your prediction.
4. Get correct answer y_t and give to A.

Thus, the linear algorithm A sees a transformed sequence $((\boldsymbol{\psi}(x_t), y_t)) \in (\mathbf{R}^r \times \{-1, 1\})^T$. Ideally, we hope to choose $\boldsymbol{\psi}$ such that this sequence is linearly separable with large margin.

The original instance base X need not be a subset of $\mathbf{R}^n$ for any n. However, having $X = \mathbf{R}^n$ with $n \ll r$ is an important special case.

Example 2.27: Consider learning k-clause l-CNF formulas over n variables. We choose as features all disjunctions of at most l literals. Thus, for $n = 4$ and $l = 2$ we have

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{x}) \;=\; (&\mathbf{false}, \mathbf{true}, x_1, x_2, x_3, x_4, \overline{x_1}, \overline{x_2}, \overline{x_3}, \overline{x_4},\\ &x_1 + x_2, x_1 + x_3, x_1 + x_4, x_1 + \overline{x_2}, x_1 + \overline{x_3}, x_1 + \overline{x_4},\\ &\overline{x_1} + x_2, \overline{x_1} + x_3, \overline{x_1} + x_4, \overline{x_1} + \overline{x_2}, \overline{x_1} + \overline{x_3}, \overline{x_1} + \overline{x_4},\\ &x_2 + x_3, x_2 + x_4, x_2 + \overline{x_3}, x_2 + \overline{x_4}, \overline{x_2} + x_3, \overline{x_2} + x_4, \overline{x_2} + \overline{x_3}, \overline{x_2} + \overline{x_4},\\ &x_3 + x_4, x_3 + \overline{x_4}, \overline{x_3} + x_4, \overline{x_3} + \overline{x_4}).\end{aligned}$$

(The ordering of the features is of course unimportant.) Here $r = 34$. In general we can roughly estimate estimate $r \leq (2n + 1)^l$.

Assume now $y_t = g(\boldsymbol{x}_t)$ where $\boldsymbol{x}_t \in \{-1, 1\}^n$ and g is a k-clause l-CNF formula. Then $y_t = \tilde{g}(\boldsymbol{\psi}(\boldsymbol{x}_t))$ where $\tilde{g}$ is a k-literal conjunction. Based on the previous example, we know that using the Perceptron Algorithm with this feature map will make at most

$$r(2k^2 - 2k + 1) \leq (2n + 1)^l(2k^2 - 2k + 1)$$

mistakes. □

Example 2.28: A monomial over (real-valued) variables $x_1, \ldots, x_n$ is a product $x_1^{k_1} \ldots x_n^{k_n}$. The degree of the monomial is $k_1 + \cdots + k_n$. Let r be the number of degree k monomials over n variables, and let ψ_i, $i = 1, \ldots, r$, be these monomials. A degree k polynomial over $\boldsymbol{x}$ is a sum

$$\sum_{i=1}^{r} a_i \psi_i(\boldsymbol{x})$$

where $a_i \in \mathbf{R}$. A degree k polynomial classifier is a classifier g that can be represented as $g(\boldsymbol{x}) = \text{sign}(p(\boldsymbol{x}))$ for some degree k polynomial p. The feature map $\boldsymbol{\psi}$ reduces learning degree k polynomial classifiers to learning linear classifiers in r dimensions.

It can be shown that $r = \binom{n+k}{k}$. As a useful estimate, we have

$$\left(\frac{n}{k} + 1\right)^k \leq \binom{n+k}{k} \leq \left(\frac{\mathrm{e}n}{k} + \mathrm{e}\right)^k.$$

As a practical application, consider classifying images given as pixel vectors. Using a linear classifier would allow us to give weights to individual pixels, which is often not very informative. Using degree 2 polynomials allows us to consider correlations between pair of pixels. In general, degree k polynomials consider interactions between groups of k pixels. □

The Kernel Trick

This is an important technique that makes feature mapping particularly attractive for linear learning.

The feature spaces $\mathbf{R}^r$ of interesting feature mappings tend to have **very** large r. This is a computational problem, since it looks like we would need to do a lot of manipulation of r-dimensional vectors.

A kernel function for feature map ψ is a function $k\colon X^2 \to \mathbf{R}$ such that $k(x,z) = \psi(x) \cdot \psi(z)$ for all $x, z \in X$. Here $\cdot$ is the dot product in the r-dimensional feature space. Often the kernel is much simpler to compute than the actual feature map (examples follow).

To apply to the Perceptron, notice that with feature map ψ we have $\boldsymbol{w}_t = \sum_{j=1}^{t-1} \sigma_j y_j \psi(x_j)$. Therefore

$$\boldsymbol{w}_t \cdot \psi(\boldsymbol{x}_t) = \sum_{j=1}^{t-1} \sigma_j y_j \psi(x_j) \cdot \psi(x_t) = \sum_{j=1}^{t-1} \sigma_j y_j k(x_j, x_t).$$

We get the following algorithm:

Algorithm 2.29 (Kernelised Perceptron):

For $t = 1, \ldots, T$ do the following:

1. Get the instance $x_t \in X$.
2. Let $p_t = \sum_{j=1}^{t-1} \sigma_j y_j k(x_j, x_t)$. Predict $\widehat{y}_t = \mathrm{sign}(p_t)$.
3. Get the correct answer $y_t \in \{-1, 1\}$.
4. If $y_t p_t \leq 0$, set $\sigma_t = 1$ and store y_t and x_t.
   Otherwise $\sigma_t = 0$ and x_t can be discarded.

Instead of storing (and manipulating) an explicit feature space weight vector $\boldsymbol{w}_t$, which would have r components, we store $O(T)$ instances x_t and coefficients σ_t. For large enough T, this can be a computational problem, too.

Theorem 2.30: Let $\boldsymbol{\psi}: X \to \mathbf{R}^r$ be a feature map with kernel k, and write $X^2 = \max_t k(x_t, x_t)$. If there is a vector $\boldsymbol{u} \in \mathbf{R}^r$ such that $\|\boldsymbol{u}\|_2 = 1$ and $y_t \boldsymbol{u} \cdot \boldsymbol{\psi}(x_t) \geq \gamma > 0$ for all t, then the Kernelised Perceptron makes at most

$$\frac{X^2}{\gamma^2}$$

mistakes on sequence $((x_t, y_t))$.

Proof: This is a direct corollary the the original Perceptron Convergence Theorem applied to the sequence $((\boldsymbol{\psi}(x_t), y_t))$. Notice that $\|\boldsymbol{\psi}(x)\|_2^2 = \boldsymbol{\psi}(x) \cdot \boldsymbol{\psi}(x) = k(x, x)$. □

The vector $\boldsymbol{u}$ in the theorem can be any vector in the feature space. We do *not* require $\boldsymbol{u} = \boldsymbol{\psi}(z)$ for some $z \in X$.

Next we consider some basic examples of kernels.

Example 2.31 (Monomial kernel): Consider $X \subseteq \mathbf{R}^n$, and take as features all the monomials of degree exactly q, for some $q \geq 2$. The feature space has dimension $r = \binom{n+q-1}{q}$ which is $\Theta(n^q)$ for constant q. We have

$$k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z})^q$$

which can be seen as follows:

$$\begin{aligned}
(\boldsymbol{x} \cdot \boldsymbol{z})^q &= (x_1 z_1 + \cdots + x_n z_n)^q \\
&= \sum_{j=1}^{r} \psi_j((x_1 z_1, \ldots, x_n z_n)) \\
&= \sum_{j=1}^{r} \psi_j((x_1, \ldots, z_n)) \psi_j((z_1, \ldots, z_n)) \\
&= \boldsymbol{\psi}(\boldsymbol{x}) \cdot \boldsymbol{\psi}(\boldsymbol{z}).
\end{aligned}$$

We have reduced computing the dot product in $\Theta(n^k)$ dimensional feature space to computing the dot product in n dimensional space and taking a power which does not depend on n. □

Example 2.32 (Polynomial kernel): The degree q polynomial kernel, again for $X \subseteq \mathbf{R}^n$, is given by

$$k_q(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x} \cdot \boldsymbol{z} + c)^q$$

where $c > 0$ is some suitable constant. It can be shown that the dimension of the feature space is $r = \binom{n+q}{q}$ and each feature is one of the r monomials of degree at most q multiplied by some constant.

The values of these constants can be determined by writing

$$k_q(\boldsymbol{x}, \boldsymbol{z}) = \sum_{j=0}^{q} \binom{q}{j} c^{q-j} (\boldsymbol{x} \cdot \boldsymbol{z})^j$$

and rewriting $(\boldsymbol{x} \cdot \boldsymbol{z})^j$ as in previous example. For practical purposes this is not really interesting:

> We do not really care about the details of the feature map, since we never need to compute it and the kernel tells us all we need to know.

The expansion above does indicate that the larger c, the less weight is given to high-degree monomials. □

Example 2.33 (All subsets kernel): Again $X \subseteq \mathbf{R}^n$. We take as features the functions ψ_A for all $A \subseteq \{1, \ldots, n\}$ where

$$\psi_A(\boldsymbol{x}) = \prod_{i \in A} x_i.$$

In other words, we have all monomials where each individual variable may have degree at most 1. There are 2^n such monomials, and the kernel can be written as

$$k(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{n} (1 + x_i z_i).$$

This has an interesting application to Boolean functions. We encode **true** as 1 and **false** as 0 (instead of the ± 1 encoding we have been using). Then for $\boldsymbol{x} \in \{0, 1\}^n$, the features ψ_A are exactly the monotone conjunctions over n variables. Further, we can include non-monotone ones by using

$$k'(\boldsymbol{x}, \boldsymbol{z}) = \prod_{i=1}^{n} (1 + x_i z_i)(1 + (1 - x_i)(1 - z_i))$$

or equivalently replacing $\boldsymbol{x} \in \{0, 1\}^n$ by $(x_1, \ldots, x_n, 1 - x_1, \ldots, 1 - x_n) \in \{0, 1\}^{2n}$.

Denote the feature map corresponding to k' by $\boldsymbol{\psi}'$. It has 4^n features that for $\boldsymbol{x} \in \{0,1\}^n$ become all the conjunctions, with **false** having several representations.

An arbitrary l-term DNF formula can be represented as $\text{sign}(\boldsymbol{u} \cdot \boldsymbol{\psi}'(\boldsymbol{x}))$ where $\|\boldsymbol{u}\|_2^2 = l$ and the unnormalised margin is $1/2$. Thus, it would seem that with this kernel the Perceptron could learn arbitrary Boolean formulas with $O(n)$ time per update.

Unfortunately, this will not work, since $\max_{\boldsymbol{x}} k'(\boldsymbol{x}, \boldsymbol{x}) = 2^n$ so the mistake bound becomes

$$\frac{2^n}{1/(2\sqrt{l})^2} = l2^{n+2}$$

which is larger than $|X| = 2^n$. Since this also means that the number of non-zero σ_t can be $\Omega(2^n)$, also the computational efficiency becomes questionable. □

Example 2.34 (Gaussian kernel): For $X \subseteq \mathbf{R}^n$, we define

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_2^2}{2\sigma^2}\right)$$

where $\sigma > 0$ is a suitably chosen parameter. (The "suitable" values depend on application and may not be trivial to find.) This is also known as the radial basis function (RBF) kernel.

In this case the feature space is actually not $\mathbf{R}^n$ for any finite n but an infinite dimensional Hilbert space. Still, the computations can be done in finite time using kernels and the mistake bound analysis can be done using some inner product in some feature space. We ignore the formal details for now.

Since now $k(\boldsymbol{x}, \boldsymbol{x}) = 1$ for all $\boldsymbol{x} \in \mathbf{R}^n$, we can interpret the kernel as a similarity measure:

$$\begin{aligned}
\|\boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z})\|_H^2 &= \langle \boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z}), \boldsymbol{\psi}(\boldsymbol{x}) - \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H \\
&= \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{x}) \rangle_H - 2 \langle \boldsymbol{\psi}(\boldsymbol{x}), \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H + \langle \boldsymbol{\psi}(\boldsymbol{z}), \boldsymbol{\psi}(\boldsymbol{z}) \rangle_H \\
&= k(\boldsymbol{x}, \boldsymbol{x}) - 2k(\boldsymbol{x}, \boldsymbol{z}) + k(\boldsymbol{z}, \boldsymbol{z}) \\
&= 2 - 2k(\boldsymbol{x}, \boldsymbol{z})
\end{aligned}$$

where $\|\cdot\|_H$ and $\langle \cdot, \cdot \rangle_H$ refer to the norm and inner product in the Hilbert space.

A large number of easy-to-compute kernels are known, and more are being developed. We shall not try to list them here, but it should be noted that kernels exist for a wide variety of instance classes X (trees, graphs, strings, text documents, . . .).

The recent interest in kernels is largely due to the success of Support Vector Machines (SVMs) that are a statistical learning algorithm based on kernels and large margins. Kernels themselves are an old idea in mathematics, and their use in "machine learning" goes back to 1960's.

We return to SVMs later, and consider some theoretical issues we ignored here (such as, given $k(\cdot, \cdot)$, is it really a kernel for some feature map).

Despite its simplicity, the Kernelised Perceptron is surprisingly good. More sophisticated algorithms (such as SVM) can be more accurate, but often not by much, and they are computationally much more expensive.

2.3 Linear online classification, part II

The Winnow algorithm appears similar to the Perceptron but has subtly different performance characteristic. The name refers to the algorithm's goal of quickly ignoring unimportant components of the input vectors ("chaff").

Algorithm 2.35 (Winnow): as parameter, learning rate $\eta > 0$

Initialize $\boldsymbol{w}_1 = \mathbf{1}$.
Repeat for $t = 1, \ldots, T$:
1. Receive input $\boldsymbol{x}_t \in \mathbf{R}^n$.
2. Predict $\widehat{y}_t = \text{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$.
3. Receive correct answer $y_t \in \{-1, 1\}$.
   Let $\sigma_t = 1$ if $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq 0$ and $\sigma_t = 0$ otherwise.
4. Update $w_{t+1} = w_{t,i} \exp(\eta \sigma_t y_t x_{t,i})$ for $i = 1, \ldots, n$.

Due to the multiplicative update, the weights remain always positive. We shall soon see how the algorithm can be made to learn classifiers with negative weights.

To understand the difference between Winnow and the Perceptron, we need to consider different norms in $\mathbf{R}^n$.

Given $p \geq 1$, define the p-norm by

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Case $p = 2$ gives the familiar Euclidean norm. We shall also need the 1-norm and the limiting case

$$\|\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| .$$

If $1/p + 1/q = 1$, we say that p and q-norms are dual. Thus 2-norm is dual with itself. The 1-norm is dual with the ∞-norm. Any pair of dual norms satisfies *Hölder's inequality*

$$|\boldsymbol{w} \cdot \boldsymbol{x}| \leq \|\boldsymbol{w}\|_p \|\boldsymbol{x}\|_q.$$

It is fairly easy to see that $\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1$ for all $\boldsymbol{x}$.

Given $\boldsymbol{u}$ and an example $(\boldsymbol{x}_t, y_t)$, we define the p-normalised margin of $\boldsymbol{u}$ with respect to $(\boldsymbol{x}_t, y_t)$ as

$$\frac{y_t \boldsymbol{u} \cdot \boldsymbol{x}_t}{\|\boldsymbol{u}\|_p}.$$

The geometrical interpretation is given by the fact that $|\boldsymbol{u} \cdot \boldsymbol{x}_t| / \|\boldsymbol{u}\|_p$ is the distance of $\boldsymbol{x}$ from the hyperplane $\{\, \boldsymbol{x} \mid \boldsymbol{x} \cdot \boldsymbol{u} = 0 \,\}$ as measured by q-norm where $1/p + 1/q = 1$.

Theorem 2.36: Assume that $\boldsymbol{u}$ is such that $u_i \geq 0$ for all i and $\boldsymbol{u}$ has 1-normalised margin at least α with respect to all examples (x_t, y_t). Assume $X > 0$ is such that $\|\boldsymbol{x}_t\|_\infty \leq X$ for all t. Then Winnow with learning rate $\eta = \alpha / X^2$ has mistake bound

$$\sum_{i=1}^{T} \sigma_t \leq \frac{2X^2 \ln n}{\alpha^2}.$$

Before the proof, we introduce a standard trick that allows us to use negative weights, and then compare the result with Perceptron.

Given $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$, let $\tilde{\boldsymbol{x}} = (x_1, \ldots, x_n, -x_1, \ldots, -x_n) \in \mathbf{R}^{2n}$.

Further, given $\boldsymbol{u} \in \mathbf{R}^n$, define $\boldsymbol{u}' \in [0, \infty)^{2n}$ by setting for $i = 1, \ldots, n$

$$\begin{aligned} u'_i = u_i \quad \text{and} \quad u'_{i+n} = 0 \qquad & \text{if } u_i \geq 0 \\ u'_i = 0 \quad \text{and} \quad u'_{i+n} = -u_i \qquad & \text{if } u_i < 0. \end{aligned}$$

Then $\boldsymbol{u}' \cdot \tilde{\boldsymbol{x}} = \boldsymbol{u} \cdot \boldsymbol{x}$, and $\|\boldsymbol{u}'\|_p = \|\boldsymbol{u}\|_p$ for all p.

Hence, if $((\boldsymbol{x}_t, y_t)) \in (\mathbf{R}^n \times \{-1, 1\})^T$ can be separated with margin α, then also $((\tilde{\boldsymbol{x}}_t, y_t)) \in (\mathbf{R}^{2n} \times \{-1, 1\})^T$ can be separated with margin α, and this can be done using only positive weights.

Denote by Winnow$^{\pm}$ the algorithm that on sequence $((\boldsymbol{x}_t, y_t))$ gives the same predictions that Winnow gives on sequence $((\tilde{\boldsymbol{x}}_t, y_t))$.

Corollary 2.37: Assume that $\boldsymbol{u}$ has 1-normalised margin at least α and $X > 0$ is such that $\|\boldsymbol{x}_t\|_\infty \leq X$ for all t. Then Winnow$^{\pm}$ with learning rate $\eta = \alpha/X^2$ has mistake bound

$$\sum_{i=1}^{T} \sigma_t \leq \frac{2X^2 \ln 2n}{\alpha^2}.$$

Caveat: We are assuming an optimal learning rate based on knowing the margin. In practice, we do not know the margin, and various empirical methods can be used to choose η. We shall not get into variants of Winnow that tune the learning rate automatically.

To compare the Perceptron and Winnow bounds, assume that some $\boldsymbol{u}$ has $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq 1$ for all t Then the 2-normalised margin is $1/\|\boldsymbol{u}\|_2^2$ and the 1-normalised margin $1/\|\boldsymbol{u}\|_1^2$.

Perceptron has mistake bound

$$\|\boldsymbol{u}\|_2^2 \max_t \|\boldsymbol{x}_t\|_2^2.$$

Winnow has mistake bound

$$2\|\boldsymbol{u}\|_1^2 \max_t \|\boldsymbol{x}_t\|_\infty^2 \ln 2n.$$

Since $\|\boldsymbol{u}\|_2 \leq \|\boldsymbol{u}\|_1$ and $\|\boldsymbol{x}_t\|_2 \geq \|\boldsymbol{x}_t\|_\infty$, the bounds are uncomparable. Perceprons wins on the $\boldsymbol{u}$ part, Winnow on the $\boldsymbol{x}_t$ part.

This becomes clearer by considering two extreme examples.

Example 2.38: Suppose $\boldsymbol{u} \in \{-1, 1\}^n$, and $\boldsymbol{x}_t \in -1, 0, 1^n$ are such that each $\boldsymbol{x}_t$ has exactly k non-zero components. Then $\|\boldsymbol{u}\|_2^2 = n$, $\|\boldsymbol{u}\|_1^2 = n^2$, $\|\boldsymbol{x}_t\|_2^2 = k$ and $\|\boldsymbol{x}_t\|_\infty = 1$. Perceptron gets kn, Winnow $2n^2 \ln 2n$, so for $k \ll n$, Perceptron is the clear winner. □

Example 2.39: Suppose $\boldsymbol{u} \in \{-1, 0, 1\}^n$ has exactly k non-zero components, and $\boldsymbol{x}_t \in \{-1, 1\}^n$. Then $\|\boldsymbol{u}\|_2^2 = k$, $\|\boldsymbol{u}\|_1^2 = k^2$, $\|\boldsymbol{x}_t\|_2^2 = n$ and $\|\boldsymbol{x}_t\|_\infty = 1$. Perceptron gets kn, Winnow $2k^2 \ln 2n$, so for $k \ll n$ Winnow is clear winner.

We say that variable (or attribute) x_i is irrelevant if $u_i = 0$. This example shows that Winnow is attribute-efficient: the mistake bound depends polynomially on the number of relevant attributes, but only logarithmically on the number of irrelevant attributes. □

Strictly speaking this comparison is meaningless, since we are comparing upper bounds without knowing how tight they are. However, simple experiments show that our conclusions are at least qualitatively correct.

Like any algorithm, Winnow can of course be used in combination with a feature map. However, in general the kernel trick does **not** work. We have

$$\boldsymbol{\psi}(x) \cdot \boldsymbol{w}_t = \sum_{i=1}^{r} \psi_i(x) \exp\left(\eta \sum_{j=1}^{t-1} \sigma_t y_t \psi_i(x_j)\right)$$

and this sum cannot be written in terms of $\boldsymbol{\psi}(x) \cdot \boldsymbol{\psi}(x_t)$ except in rare special cases.

Example 2.40: In Example 2.33 we noticed that (a minor modification of) the all subsets kernel allows learning DNF formulas with the Perceptron algorithm with polynomial time per update. Unfortunately the mistake bound was exponential in n.

By using the same linear representation, we see that the 1-normalised margin for l-term DNF is $1/(2l)$. Since $\|\boldsymbol{\psi}'(\boldsymbol{x})\|_\infty = 1$ and $\boldsymbol{\psi}'(\boldsymbol{x}) \in \{0,1\}^{4^n}$, Winnow gets the mistake bound $8k^2(2n+1)\ln 2$ which is polynomial in the size n of inputs and $kn \log n$ of the target formula. Unfortunately there is no known method of implementing the algorithm in poly(n) time per update. (For this particular kernel it has been shown that such an implementation does not exist if P $\neq$ NP.) $\square$

Proof of Winnow mistake bound: Assume that $\sum_i u_i = 1$, $u_i \geq 0$ for all i and $\boldsymbol{u}$ has unnormalised margin at least α on all the examples. We show that the normalised weights $\boldsymbol{v}_t = \boldsymbol{w}_t / \sum_i w_{t,i}$ converge towards $\boldsymbol{u}$. As potential function we use the relative entropy (or Kullback-Leibler divergence)

$$P_t = d_{\mathsf{re}}(\boldsymbol{u}, \boldsymbol{v}_t) = \sum_{i=1}^{n} u_i \ln \frac{u_i}{v_{t,i}}$$

where $0 \ln 0 = 0 \ln(0/0) = 0$ and otherwise $\ln(1/0) = \infty$ as usual.

Our starting point is the identity

$$d_{\mathsf{re}}(\boldsymbol{u}, \boldsymbol{v}_t) - d_{\mathsf{re}}(\boldsymbol{u}, \boldsymbol{v}_{t+1}) = (\boldsymbol{u} \cdot \boldsymbol{z} - \boldsymbol{v}_t \cdot \boldsymbol{z}) - d_{\mathsf{re}}(\boldsymbol{v}_t, \boldsymbol{v}_{t+1})$$

that holds for $w_{t+1,i} = w_{t,i} e^{z_i}$. Taking $\boldsymbol{z} = \eta \sigma_t y_t \boldsymbol{x}_t$ and using the known estimate

$$d_{\mathsf{re}}(\boldsymbol{v}_t, \boldsymbol{v}_{t+1}) \leq \frac{\|\boldsymbol{z}\|_\infty^2}{2}$$

we get

$$P_t - P_{t+1} \geq \eta \sigma_t y_t (\boldsymbol{u} \cdot \boldsymbol{x}_t - \boldsymbol{v}_t \cdot \boldsymbol{x}_t) - \sigma_t \frac{\eta^2 X^2}{2}.$$

We use $y_t \boldsymbol{u} \cdot \boldsymbol{x}_t \geq \alpha$ and $\sigma_t y_t \boldsymbol{v}_t \cdot \boldsymbol{x}_t \leq 0$ and sum over $t = 1, \ldots, T$ to get

$$\begin{aligned} P_1 - P_{t+1} &= \sum_{t=1}^{T} (P_t - P_{t+1}) \\ &\geq \left(\eta\alpha - \frac{\eta^2 X^2}{2} \right) \sum_{t=1}^{T} \sigma_t. \end{aligned}$$

We choose $\eta = \alpha / X^2$ to maximise the right-hand side. We can show $d_{\mathsf{re}}(\boldsymbol{u}, \boldsymbol{v}_t) \geq 0$ using Jensen's inequality. Since the initial weight vector is uniform, we have

$$d_{\mathsf{re}}(\boldsymbol{u}, \boldsymbol{v}_1) = \sum_{i=1}^{n} u_i \ln \frac{u_i}{1/n} = \ln n - \sum_{i=1}^{n} u_i \ln \frac{1}{u_i} \leq \ln n.$$

Hence, the inequality becomes

$$\ln n \geq \frac{\alpha^2}{2X^2} \sum_{t=1}^{T} \sigma_t$$

from which the claim follows. □

Remark The similarity between the formulas for $d_{\mathrm{re}}(\boldsymbol{u}, \boldsymbol{v}_t) - d_{\mathrm{re}}(\boldsymbol{u}, \boldsymbol{v}_{t+1})$ above and $\frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_t\|_2^2 - \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|_2^2$ in the Perceptron proof is no coincidence. The relative entropy and squared Euclidean distance are examples of so-called Bregman divergences.

The idea of these two algorithms and proofs can be generalised to give for any $p \geq 2$ an algorithm that has mistake bound

$$O(p\|\boldsymbol{u}\|_q^2 \max_t \|\boldsymbol{x}_t\|_p^2)$$

where $1/p + 1/q = 1$ and $\boldsymbol{u}$ has unnormalised margin at least 1. This algorithm is called the p-norm Perceptron. Case $p = 2$ gives the usual Perceptron, while $p = \Theta(\log n)$ gives a bound similar to Winnow. □

Often there is no $\boldsymbol{u}$ with positive margin on all the examples:

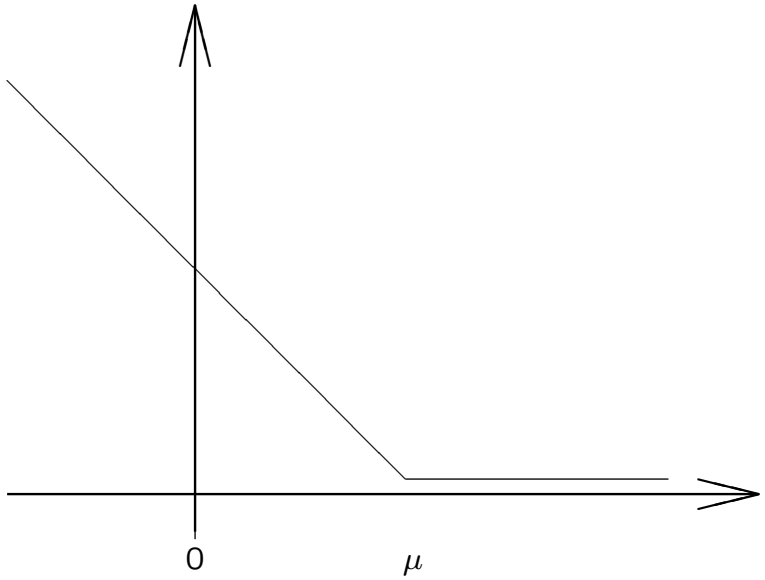
- the examples may contain errors
- the target classifier may be non-linear
- a target classifier may not exist at all (or the target is "probabilistic")

We generalise our bounds for this scenario by considering the hinge loss (or soft margin loss).

For any margin parameter $\mu \geq 0$ we define the hinge loss as

$$L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) = \max\{0, \mu - y\boldsymbol{u} \cdot \boldsymbol{x}\}.$$

If $y\boldsymbol{u} \cdot \boldsymbol{x} \leq 0$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) \geq \mu$.
If $y\boldsymbol{u} \cdot \boldsymbol{x} \geq \mu$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) = 0$.



Given some sequence $((\boldsymbol{x}_t, y_t)) \in (\mathbf{R}^n \times \{-1, 1\})^T$, define for any linear prediction algorithm A and any fixed weight vector $\boldsymbol{u} \in \mathbf{R}^n$

$$L_\mu(A) = \sum_{t=1}^{T} L_\mu(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t) \quad \text{and} \quad L_\mu(\boldsymbol{u}) = \sum_{t=1}^{T} L_\mu(\boldsymbol{u}, \boldsymbol{x}_t, y_t).$$

Define similarly the discrete losses $L_{0-1}(A)$ and $L_{0-1}(\boldsymbol{u})$.

If $y\boldsymbol{u} \cdot \boldsymbol{x} \leq 0$, then $L_\mu(\boldsymbol{u}, \boldsymbol{x}, y) \geq \mu$. Therefore

$$L_{0-1}(\boldsymbol{u}) \leq \frac{1}{\mu} L_\mu(\boldsymbol{u}).$$

Ideally, we would wish to have bounds of the form

$$L_{0-1}(A) \leq a L_{0-1}(\boldsymbol{u}) + b$$

for some constants a and b (that might depend on norms of $\boldsymbol{x}_t$ etc.). However, this seems very difficult. We settle for weaker bounds of the form

$$L_{0-1}(A) \leq \frac{a}{\mu} L_\mu(\boldsymbol{u}) + b.$$

We obtain the desired bound for Perceptron by analysing the following more general algorithm in terms of hinge loss and then considering some special cases.

Algorithm 2.41 (Marginalised Perceptron (MP)):
parameters: learning rate $\eta > 0$, margin parameter $\rho \geq 0$.

Initialise $\boldsymbol{w}_1 = \mathbf{0} \in \mathbf{R}^n$.
Repeat for $t = 1, \ldots, T$:
1. Get input $\boldsymbol{x}_t \in \mathbf{R}^n$.
2. Predict $\widehat{y}_t = \mathrm{sign}(\boldsymbol{w}_t \cdot \boldsymbol{x}_t)$.
3. Get correct answer $y_t \in \{-1, 1\}$.
4. If $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq \rho$ then $\tilde{\sigma}_t = 1$.
   Otherwise $\tilde{\sigma}_t = 0$.
5. Let $\boldsymbol{w}'_{t+1} = \boldsymbol{w}_t + \eta \tilde{\sigma}_t y_t \boldsymbol{x}_t$.
6. If $\|\boldsymbol{w}'_{t+1}\|_2 > 1$ then $\boldsymbol{w}_{t+1} = \boldsymbol{w}'_{t+1} / \|\boldsymbol{w}'_{t+1}\|_2$.
   Otherwise $\boldsymbol{w}_{t+1} = \boldsymbol{w}'_{t+1}$.

We get the usual Perceptron by taking $\rho = 0$ and omitting the normalisation (step 6). The value η is then irrelevant.

If $y_t \boldsymbol{w}_t \cdot \boldsymbol{x}_t \leq \rho$, we say that a margin error occurred, which we denote by $\tilde{\sigma}_t = 1$. Step 6 forces the hypothesis to stay inside the unit ball.

We start by a bound on margin errors.

Theorem 2.42: Assume $\|\boldsymbol{u}\|_2 \leq 1$ and $\|\boldsymbol{x}_t\|_2 \leq X$ for all t. Then for any $0 \leq \rho < \mu$ and $0 < \eta < 2(\mu - \rho)/X^2$ the Marginalised Perceptron has

$$\sum_{t=1}^{T} \tilde{\sigma}_t \leq \frac{L_\mu(\boldsymbol{u})}{\mu - \rho - \eta X^2/2} + \frac{1}{2\eta(\mu - \rho - \eta X^2/2)}.$$

Intuitively μ is our idea of the margin with which the examples "should" be linearly separable. If the examples actually are not separable with margin μ, we want to consider the difference as "noise" that makes learning harder but ideally should not affect the end result.

Then ρ is the goal we set to our algorithm. We would wish to set ρ as close to μ as possible, but this will cost more margin errors.

After we have decided our goal by fixing ρ, the learning rate η can be tuned to get the fastest learning.

This becomes clearer by considering two extreme examples.

Example 2.43: Choose $\rho = 0$, so $\tilde{\sigma}_t = \sigma_t$. By straightforward calculus, we can find η such that the bound is minimised. The optimised bound is

$$\begin{aligned}\sum_{t=1}^{T} \sigma_t &\leq \frac{L_\mu(\boldsymbol{u})}{\mu} + \frac{X^2}{2\mu^2} + \left(\frac{2L_\mu(\boldsymbol{u})}{\mu} + \frac{X^2}{2\mu^2}\right)^{1/2} \left(\frac{X^2}{2\mu^2}\right)^{1/2} \\ &= \frac{L_\mu(\boldsymbol{u})}{\mu} + O\left(\sqrt{L_\mu(\boldsymbol{u})}\right)\end{aligned}$$

if we consider X and μ as constants. This gives a bound for how much the performance of the algorithm degrades, if the data are not linearly separable.

If we take $\rho = 0$ and omit the normalisation step, we get a bound for the usual Perceptron when the data is not linearly separable. The fact that our bound would seem to suggest a particular value η is an artefact of our proof technique. □

Example 2.44: Assume that $\boldsymbol{u}$ actually separates $((\boldsymbol{x}_t, y_t))$ with margin μ, so $L_\mu(\boldsymbol{u}) = 0$. Choose $\rho = (1 - \varepsilon)\mu$ for some $0 < \varepsilon \leq 1$.

The bound is minimised for $\eta = (\mu - \rho)/X^2$, giving

$$\sum_{t=1}^{T} \tilde{\sigma}_t \leq \frac{X^2}{\varepsilon^2 \mu^2}.$$

Setting $\varepsilon = 1$ gives the familiar Perceptron mistake bound.

More generally, if we know the optimal margin μ, we can force the algorithm to find a hypothesis with margin within ε of optimal, but the number of updated needed for that grows as $O(1/\varepsilon^2)$.

Knowing μ in advance is often not realistic. There are more sophisticated algorithms that can approximate the optimal margin without knowing it in advance. □

Proof of Marginalised Perceptron bound: We consider the potential

$$P_t = \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_t\|_2^2.$$

This time we use the learning rate η to take care of proper scaling.

We decompose the potential decrease as

$$\begin{aligned} P_t - P_{t+1} &= \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_t\|_2^2 - \frac{1}{2}\left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 \\ &\quad + \frac{1}{2}\left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 - \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|_2^2. \end{aligned}$$

Since $\boldsymbol{u}$ is within the unit ball $B = \{\, \boldsymbol{w} \mid \|\boldsymbol{w}\|_2 \leq 1 \,\}$, and $\boldsymbol{w}_{t+1}$ is the projection of $\boldsymbol{w}'_{t+1}$ onto that ball, we can estimate

$$\frac{1}{2}\left\|\boldsymbol{u} - \boldsymbol{w}'_{t+1}\right\|_2^2 - \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{w}_{t+1}\|_2^2 \geq 0.$$

As in the Perceptron convergence proof, we have

$$\begin{aligned}\frac{1}{2}\|\boldsymbol{u}-\boldsymbol{w}_t\|_2^2-\frac{1}{2}\left\|\boldsymbol{u}-\boldsymbol{w}'_{t+1}\right\|_2^2 &= (\boldsymbol{w}'_{t+1}-\boldsymbol{w}_t)\cdot(\boldsymbol{u}-\boldsymbol{w}_t)-\frac{1}{2}\left\|\boldsymbol{w}_t-\boldsymbol{w}'_{t+1}\right\|_2^2\\ &= \eta\tilde{\sigma}_t y_t(\boldsymbol{u}-\boldsymbol{w}_t)\cdot\boldsymbol{x}_t-\frac{1}{2}\eta^2\tilde{\sigma}_t\|\boldsymbol{x}_t\|_2^2.\end{aligned}$$

We estimate $\tilde{\sigma}_t y_t \boldsymbol{w}_t\cdot\boldsymbol{x}_t\le\tilde{\sigma}_t\rho$ and $\tilde{\sigma}_t y_t\boldsymbol{u}\cdot\boldsymbol{x}_t\ge\tilde{\sigma}_t\mu-L_\mu(\boldsymbol{u},\boldsymbol{x}_t,y_t)$. Combining with the previous, we get

$$P_t-P_{t+1}\ge-\eta L_\mu(\boldsymbol{u},\boldsymbol{x}_t,y_t)+\tilde{\sigma}_t\eta\left(\mu-\rho-\eta X^2/2\right).$$

The result follows by summing over $t=1,\dots,T$ and noticing $P_1=\frac{1}{2}\|\boldsymbol{u}\|_2^2$ and $P_{T+1}\ge 0$. □

Online prediction: summary

We were interested in relative loss bounds of the form

$$L(A) \leq (1 + o(1)) \min_{f \in F} L(f)$$

where F is some comparison class of predictors (e.g., a set of experts, or all norm-bounded linear predictors). We did not always get exactly this form, in particular with discrete loss.

The algorithms are in general simple to implement and run fast.

Analysis is based on potential functions. Mathematical tools include Jensen's inequality and basic properties of Bregman divergences (squared Euclidean norm, relative entropy).

Mistake bounds for linear prediction can be given in terms of margins and different norms, which turn out to be important in other learning models, too.

3. Statistical learning theory

We introduce the main types of error bound for statistical (or "batch") learning:

- test set bound,
- Occam's Razor bound for finite hypothesis spaces and
- VC and Rademacher bounds for infinite hypothesis spaces

and also connect to online learning via

- conversion to get statistical bounds from online mistake bounds.

The goal is to

- understand the basic concepts and their proper application and
- get some idea of the underlying mathematical techniques.

3.1 Basic setting

The basic ingredients of the learning model are

X	instance space
Y	label space
$h\colon X \to Y$	classifier, or hypothesis
H	hypothesis space, a collection of classifiers
$(x, y) \in X \times Y$	example
P	probability measure on $X \times Y$
$S \in (X \times Y)^m$	sample, *i.e.*, a sequence of examples
$m = \lvert S \rvert$	sample size.

Here we mainly consider binary classification, $Y = \{-1, 1\}$.

We write $\Pr_{(x,y)\sim P}(\cdot)$ to denote the probability of an event when (x, y) is drawn from P, and $\Pr_{S\sim P^m}(\cdot)$ to denote the probability of an event when S consists of m examples drawn independenty from P. Such examples are called i.i.d. (independent, identically distributed). Usually the meaning is clear and we just write $\Pr(\cdot)$.

The basic idea is that P is unknown, but given a sample $S \sim P^m$ the learner should be able to find out something useful about P.

Example 3.1 (Noise-free PAC model): We consider a fixed collection of binary classifiers C called the concept class. We also assume there is an unknown probability measure P_X on X and an unknown target concept $f_* \in C$. The measure P on $X \times Y$ is now obtained by setting

$$P((x,y)) = \begin{cases} P_X(x) & \text{if } y = f_*(x) \\ 0 & \text{if } y \neq f_*(x). \end{cases}$$

The goal is to come up with a hypothesis h such that $\Pr_{(x,y)\sim P}(h(x) \neq y)$ is small. □

Example 3.2 (Random classification noise): The assumptions about C and f_* are as above, and so is the goal. However, now there is a fixed probability $0 \leq \eta \leq 1/2$ for each example to have its label flipped:

$$P((x,y)) = \begin{cases} (1-\eta)P_X(x) & \text{if } y = f_*(x) \\ \eta P_X(x) & \text{if } y \neq f_*(x). \end{cases}$$

□

However, from now on we do **not** want to make such (unrealistic) assumptions about a target f_* from a known class C.

Given a function f, denote its expectation by

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[f(x,y)] = \sum_{(x,y)\in X\times Y} f(x,y)P(x,y)$$

(with summation to be replaced by the appropriate integral if P is not discrete) and sample mean by

$$\mathop{\mathsf{E}}_{(x,y)\sim S}[f(x,y)] = \frac{1}{m}\sum_{t=1}^{m} f(x_t,y_t).$$

It is well known that expectations can be estimated by sample means. As sample size increases, the sample means converge, in some stochastical sense, towards the expectation. Studying the convergence properties is a well-established brach of mathematical statistics.

Statistical learning theory is largely concerned with studying such properties in the special case that $f(x,y) = L_{0-1}(y,h(x))$ for some hypothesis h.

We define the true error of h as

$$\mathrm{err}(h; P) = \mathop{\mathsf{E}}_{(x,y)\sim P}[L_{0-1}(y, h(x))] = \mathop{\mathsf{Pr}}_{(x,y)\sim P}(y \neq h(x))$$

and the empirical error of h as

$$\widehat{\mathrm{err}}(h; S) = \mathop{\mathsf{E}}_{(x,y)\sim S}[L_{0-1}(y, h(x))] = \frac{1}{m}\left|\{\, t \in \{\, 1, \ldots, m \,\} \mid h(x_t) \neq y_t \,\}\right|.$$

Usually P and S are clear from the context, and we write $\mathrm{err}(h)$ and $\widehat{\mathrm{err}}(h)$.

Since the sample is given but P is unknown, $\widehat{\mathrm{err}}(h)$ is directly observable but $\mathrm{err}(h)$ is not. However, $\mathrm{err}(h)$ is what we are really interested in.

1. For a given h, is $\widehat{\mathrm{err}}(h)$ a reliable estimate of $\mathrm{err}(h)$?

2. Is minimising $\widehat{\mathrm{err}}(h)$ a good way to find h such that $\mathrm{err}(h)$ is small?

The answer is basically yes, under some assumptions, and we may need to include some correction terms. We next study question 1 in detail.

3.2 Test set bounds

Suppose someone gives us a sample S from some unknown P and asks us to produce

1. a hypothesis h, and
2. an estimate of its true error $\mathrm{err}(h)$.

The basic procedure is as follows:

1. Split S into a training set S_{train} and test set S_{test}. A typical choice is $|S_{\mathrm{train}}| \approx 0.7m$ and $|S_{\mathrm{test}}| \approx 0.3m$, but this is basically a guess.
2. Run your favourite learning algorithm on S_{train} to obtain h.
3. Use $\widehat{\mathrm{err}}(h; S_{\mathrm{test}})$ as estimate for $\mathrm{err}(h)$.

Since S_{train} and S_{test} are independent, $\widehat{\mathrm{err}}(h, S_{\mathrm{test}})$ is an unbiased estimate of $\mathrm{err}(h)$ (*i.e.*, $\mathsf{E}[\widehat{\mathrm{err}}(h, S_{\mathrm{test}})] = \mathrm{err}(h)$). The "test set bound" is a bound for error in that estimate.

Notice: It is necessary to keep the training and test sets separate. Typically $\widehat{\mathrm{err}}(h; S_{\mathrm{train}})$ is a **bad underestimate** of the true error.

Example 3.3: Many learning algorithms are based on empirical risk minimisation (ERM): The algorithm has some fixed hypothesis class H, and gives the hypothesis

$$h_* = \arg\min_{h \in H} \widehat{\mathrm{err}}(h, S).$$

that has the least empirical risk. (The minimisation problem is intractable for many interesting classes H, like linear classifiers, so in practice various heuristics are used instead of exact minimisation.)

Clearly $\widehat{\mathrm{err}}(h_*)$ is an underestimate of $\mathrm{err}(h_*)$. Suppose the data are random noise, so $\mathrm{err}(h) = 1/2$ for all h. Still on any given sample S, if H is large enough there will be some h_* that happens to get $\widehat{\mathrm{err}}(h_*; S) < 1/2$ for that particular S. □

Suppose now that we have several learning algorithms $A_1, \ldots, A_n$ we wish to try. This could be n entirely different algorithms, or just the Kernelised Perceptron with the Gaussian kernel using n different kernel widths σ.

The basic procedure is the following:

1. Split S into training set S_{train}, validation set S_{valid} and test set S_{test}.

2. For $i = 1, \ldots, n$, run A_i on S_{train} to get a hypothesis h_i.

3. Choose as the final hypothesis

$$h_* = \arg\min_{1 \le i \le n} \widehat{\text{err}}(h_i; S_{\text{valid}}).$$

4. Estimate the true error by $\widehat{\text{err}}(h_*; S_{\text{test}})$.

Again, do **not** evaluate a hypothesis using the data it was trained on (unless you know how to apply an appropriate correction, as we see later).

A more sophisticated error estimation method is k-fold cross validation:

1. Split S into k folds $S_1, \ldots, S_k$ of roughly equal size. Let
$$S_{(-i)} = \bigcup_{j \neq i} S_j.$$
2. For $i = 1, \ldots, k$, run A on $S_{(-i)}$ to obtain hypothesis h_i.
3. For $i = 1, \ldots, k$, let $\varepsilon_i = \widehat{\mathrm{err}}(h_i; S_i)$. Let $\varepsilon = \frac{1}{k} \sum_i \varepsilon_i$.
4. Run A on S to obtain the final hypothesis h. Return ε as estimate for $\mathrm{err}(h)$.

Cross validation is widely used but not well understood theoretically.

Leave-one-out cross validation is the case $k = m$, *i.e.*, one example per fold. This can be computationally very expensive.

For small k, cross validation can be very sensitive to how the splitting to folds is done.

The assumption that the data are i.i.d. is often suspect. In such situation, the following theoretical results have no validity. The empirical test methods can still be used, but care must be taken.

The order of examples in S may be non-random, so shuffling S might be a good idea.

As a particular example, consider classification of images into cars and non-cars. The sample may consist several images of the same car from slightly different angles etc. It might be a good idea to split the data so that images of the same car are either all in the training set or all in the test set.

We now consider estimating $\mathrm{err}(h)$ in terms of $\widehat{\mathrm{err}}(h; S)$. As explained above, the context is usually that h has been obtained by running some learning algorithm on a sample that is independent of S.

Let $\mathrm{Bin}(k, m, p)$ be the probability of obtaining no more than k heads in m independent coin tosses each with probability p of coming up heads:

$$\mathrm{Bin}(k, m, p) = \sum_{i=0}^{k} \binom{m}{i} p^i (1-p)^{m-i}.$$

It is intuitively clear, and easy to verify by calculus, that $\mathrm{Bin}(k, m, p)$ is continuous and strictly decreasing in p. Therefore, we can define the inverse

$$\overline{\mathrm{Bin}}(k, m, \delta) = p \quad \text{such that} \quad \mathrm{Bin}(k, m, p) = \delta.$$

Then $\overline{\mathrm{Bin}}(k, m, \delta)$ is continuous and strictly decreasing in δ. We have

$$\begin{aligned} \mathrm{Bin}(k, m, p) \leq \delta &\quad \Leftrightarrow \quad p \geq \overline{\mathrm{Bin}}(k, m, \delta) \\ \mathrm{Bin}(k, m, p) \geq \delta &\quad \Leftrightarrow \quad p \leq \overline{\mathrm{Bin}}(k, m, \delta) \end{aligned}$$

We use various approximations of Bin in order to get asymptotic estimates. However, to obtain actual numerical values, one should usually compute the exact sums, which is easy enough on modern computers.

Theorem 3.4 (Test Set Bound): Let h be an arbitrary classifier and $0 < \delta \leq 1$. Define $\hat{\varepsilon}(S) = \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h; S), m, \delta)$. Then

$$\Pr_{S \sim P^m}(\mathrm{err}(h) \geq \hat{\varepsilon}(S)) \leq \delta.$$

The interpretation of the theorem is as follows.

We use the random variable $\hat{\varepsilon}$ as estimate for $\mathrm{err}(h)$. We do not mind if $\mathrm{err}(h)$ is actually even better than our estimate, but if the estimate is too optimistic, we consider that an error.

The theorem says that errors occur with probability at most δ. We call $1 - \delta$ the confidence parameter.

Since $\overline{\mathrm{Bin}}(k, m, \delta)$ is decreasing in δ, increasing confidence means that also the estimate goes up, *i.e.*, becomes more conservative.

Proof of Test Set Bound: This basically follows directly from definitions. Write $p = \mathrm{err}(h)$, and let $\tilde{k} = \max\{\, k \mid \mathrm{Bin}(k, m, p) \leq \delta \,\}$. Notice that $\mathrm{Bin}(k, m, p)$ is increasing in k.

Since for any k we have $\Pr_{S \sim P^m}(m\widehat{\mathrm{err}}(h; S) \leq k) = \mathrm{Bin}(k, m, p)$, we get

$$\Pr_{S \sim P^m}(m\widehat{\mathrm{err}}(h; S) \geq \tilde{k}) \geq 1 - \delta.$$

On the other hand, by the definition of $\tilde{k}$ we have

$$\begin{aligned} m\widehat{\mathrm{err}}(h; S) \geq \tilde{k} &\Leftrightarrow \mathrm{Bin}(m\widehat{\mathrm{err}}(h; S), m, p) \geq \delta \\ &\Leftrightarrow \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h; S), m, \delta) \geq p \\ &\Leftrightarrow \hat{\varepsilon}(S) \geq p. \end{aligned}$$

Hence $\hat{\varepsilon}(S) \geq p$ with probability at least $1 - \delta$. □

The proof also shows that the bound is tight.

As mentioned above, to get a good bound in a practical situation, one should compute the actual value $\overline{\mathrm{Bin}}(\ldots)$ numerically. However, next we check some approximations that show how the bound behaves asymptotically.

We start with the special case where the hypothesis makes no mistakes.

Theorem 3.5 (Realisable Test Set Bound): If

$$\mathrm{err}(h) \geq \frac{1}{m} \ln \frac{1}{\delta},$$

we have $\widehat{\mathrm{err}}(h; S) > 0$ with probability at least $1 - \delta$.

In other words, if we happen to get zero empirical error, we have confidence $1 - \delta$ that the true error is at most $\frac{1}{m} \ln \frac{1}{\delta}$.

Proof: Let $\hat{\varepsilon} = \overline{\mathsf{Bin}}(0, m, \delta)$ be the estimate from previous theorem when $\widehat{\mathsf{err}}(h) = 0$. Then $\mathsf{Bin}(0, m, \hat{\varepsilon}) = \delta$.

We have

$$\mathsf{Bin}(0, m, \hat{\varepsilon}) = (1 - \hat{\varepsilon})^m \leq e^{-m\hat{\varepsilon}}$$

where we have used $x = -\hat{\varepsilon}$ in the inequality $1 + x \leq e^x$ that holds for all x. Therefore

$$\delta \leq e^{-m\hat{\varepsilon}},$$

which is equivalent with

$$\hat{\varepsilon} \leq \frac{1}{m} \ln \frac{1}{\delta}.$$

Hence, $\frac{1}{m} \ln \frac{1}{\delta}$ is a conservative estimate of the "true" $\hat{\varepsilon}$. $\square$

There are a number of ways of approximating $\mathrm{Bin}(k, m, p)$ for $k \neq 0$. Here we consider the following simple estimate:

Lemma 3.6 (Hoeffding Bound): Let $k/m = r \leq p$. We then have

$$\mathrm{Bin}(k, m, p) \leq \exp\left(-2m(p-r)^2\right).$$

□

Corollary 3.7: Let h be an arbitrary classifier and $0 < \delta \leq 1$. Then

$$\Pr_{S \sim P^m}\left(\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h; S) + \sqrt{\frac{1}{2m}\ln\frac{1}{\delta}}\right) \geq 1 - \delta.$$

Notice that in this general case, the true error is with high confidence within $O(m^{-1/2})$ of the empirical error, whereas in the realisable case it was within $O(m^{-1})$. This is a general feature: we get faster convergence if the true error is close to zero.

Proof of Corollary: Again, let $\hat{\varepsilon} = \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h;S), m, \delta)$ be the estimate we get from the exact Test Set Bound. We claim

$$\hat{\varepsilon} \leq \widehat{\mathrm{err}}(h;S) + \sqrt{\frac{1}{2m} \ln \frac{1}{\delta}}.$$

If $\hat{\varepsilon} \leq \widehat{\mathrm{err}}(h;S)$, the claim is clear. Otherwise the Hoeffding Bound gives us

$$\delta = \mathrm{Bin}(m\widehat{\mathrm{err}}(h;S), m, \hat{\varepsilon}) \leq \exp\left(-2m(\hat{\varepsilon} - \widehat{\mathrm{err}}(h;S))^2\right)$$

from which the claim follows. □

Next we consider what kind of error estimates are possible without using a separate test set.

3.3 Occam's Razor Bound

In machine learning, Occam's Razor refers to a specific type of error bound.

Theorem 3.8 (Occam's Razor): Let H be a finite hypothesis space, and for all $h \in H$ write $\hat{\varepsilon}_h = \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h;S), m, \delta/|H|)$. Then for all $0 < \delta \leq 1$ we have

$$\Pr_{S\sim P^m}(\forall h \in H : \mathrm{err}(h) \leq \hat{\varepsilon}_h) \geq 1 - \delta.$$

The theorem says that if in the Test Set Bound estimate $\hat{\varepsilon}$ we replace δ by $\delta/|H|$, we have confidence $1-\delta$ that the bound holds uniformly for all $h \in H$.

This become more easy to visualise in the version with Hoeffding approximation (we get to the proofs soon):

Corollary 3.9: Let H be a finite hypothesis space and $0 < \delta \leq 1$. Then with probability at least $1-\delta$ we have

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h;S) + \sqrt{\frac{1}{2m}\ln\frac{|H|}{\delta}}$$

for all $h \in H$.

The Occam's Razor bound is a training set bound: it allows us to get error estimates without having a separate test set. The procedure is as follows:

1. Fix a hypothesis space H. Let A be any algorithm that always chooses its hypotesis from H.

2. Receive a sample S.

3. Run A on sample S to obtain a hypothesis $h \in H$.

4. Give h as the hypothesis and $\widehat{\varepsilon}_h$ as error estimate.

We can now use the same data for learning and testing, but we must fix H in advance. The Test Set Bound is actually the case where H consists of just one hypothesis. The cost of having $|H| > 1$ is that δ must be replaced by $\delta/|H|$, leading to bigger error estimates.

The basic Occam's Razor bound follows directly from the following by choosing $Q(h) = 1/|H|$ for all $h \in H$:

Theorem 3.10 (Non-uniform Occam's Razor): Let H be a countable hypothesis space, and let Q be a probability measure over H. Then for all $0 < \delta \leq 1$ we have

$$\Pr_{S \sim P^m} (\forall h \in H : \mathrm{err}(h) \leq \hat{\varepsilon}_h) \geq 1 - \delta$$

where $\hat{\varepsilon}_h = \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h; S), m, Q(h)\delta)$.

For applying the bound, notice that choosing Q corresponds to fixing H in the procedure on the previous slide, and must be done in advance.

The test set bound corresponds to setting $Q(h_0) = 1$ for some $h_0 \in H$ and $Q(h) = 0$ for $h \in H - \{ h_0 \}$. This means that the bound becomes vacuous for $h \neq h_0$.

Proof: For $h \in H$, let R_h be the event that $\mathrm{err}(h) > \widehat{\varepsilon}_h$. We need to show that

$$\Pr_{S \sim P^m}\left(\bigcup_{h \in H} R_h\right) \leq \delta.$$

We use the union bound that holds for any countable set of events $\{ E_i \mid i \in I \}$:

$$\Pr\left(\bigcup_{i \in I} E_i\right) \leq \sum_{i \in I} \Pr(E_i).$$

By applying the Test Set Bound with δ replaced by $Q(s)\delta$ we see that $\Pr_{S \sim P^m}(R_h) \leq Q(h)\delta$ holds for any given h. Therefore

$$\begin{aligned} \Pr_{S \sim P^m}\left(\bigcup_{h \in H} R_h\right) &\leq \sum_{h \in H} \Pr(R_h) \\ &\leq \delta \sum_{h \in H} Q(h) \\ &= \delta. \end{aligned}$$

□

We also have the non-uniform Hoeffding version:

Corollary 3.11: Let H be a countable hypothesis space, and let Q be a probability measure over H. Then for all $0 < \delta \leq 1$, with probability at least $1 - \delta$ we have

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h; S) + \sqrt{\frac{1}{2m}\left(\ln \frac{1}{Q(h)} + \ln \frac{1}{\delta}\right)}$$

for all $h \in H$.

Proof: Exactly like for the Test Set Bound, we can estimate

$$\hat{\varepsilon}_h \leq \widehat{\mathrm{err}}(h; S) + \sqrt{\frac{1}{2m}\left(\ln \frac{1}{Q(h)\delta}\right)}$$

in the previous theorem. □

The quantity $\ln(1/Q(h))$ has an important information theoretic interpretation as code length for hypothesis h.

Let W be some (countable) set of events. We wish to communicate a sequence of events $w_1, w_2, w_3, \ldots$, where $w_i \in W$, over some channel by encoding it as a string of bits, transmitting those bits over the channel, and decoding at the other end.

Any mapping $C\colon W \to \{0,1\}^*$ that maps each event to some finite bit string is called a code. For $w \in W$, the string $C(w)$ is called the codeword of w. Some additional assumptions are needed to guarantee that decoding is possible.

If in particular C has the property that no codeword is the prefix of any other codeword, we call C a prefix code. If a prefix code is used, decoding can take place instantaneously. As soon as we have received all the bits of a codeword, we know that no other interpretation of those bits is possible.

Theorem 3.12 (Kraft Inequality): Let C be a prefix code with codeword lengths $|C(w)| = l(w)$, $w \in W$. Then

$$\sum_{w \in W} 2^{-l(w)} \leq 1. \qquad (*)$$

Conversely, for any natural numbers $l(w)$ that satisfy $(*)$ there is a prefix code with codeword lengths $l(w)$. $\square$

Assume now that we have a prefix code for the hypotheses $h \in H$ with codeword lengths $l(h)$. Write $Z = \sum_{h \in H} 2^{-l(h)}$, and define a probability measure by $Q(h) = 2^{-l(h)}/Z$. Since $0 < Z \leq 1$, we have

$$\ln \frac{1}{Q(h)} = (\ln 2) l(h) + \ln Z \leq (\ln 2) l(h).$$

Thus we can estimate the errors as

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h; S) + \sqrt{\frac{1}{2m}\left((\ln 2) l(h) + \ln \frac{1}{\delta}\right)}.$$

Conversely, given Q, we can pick code word lenghts $l(h) = \lceil (\ln 2)^{-1} \ln(1/Q(h)) \rceil$ and the above bound holds.

Up to now, we have only worried about our error estimates being too optimistic. It is straightforward to also get guarantees against being too pessimistic.

Corollary 3.13: Let H be a finite hypothesis space and $0 < \delta \leq 1$. Then with probability at least $1 - \delta$ we have

$$\left|\mathrm{err}(h; P) - \widehat{\mathrm{err}}(h; S)\right| \leq \sqrt{\frac{1}{2m} \ln \frac{2|H|}{\delta}}$$

for all $h \in H$.

Proof sketch: An argument completely symmetrical with the proof of Test Set Bound (Theorem 3.4) gives us

$$\Pr_{S \sim P^m}(\mathrm{err}(h) \leq \overline{\mathrm{Bin}}(m\widehat{\mathrm{err}}(h; S), m, 1 - \delta)) \leq \delta.$$

We then apply the Hoeffding bound and replace δ by $\delta/(2|H|)$ to apply the union bound to the $|H|$ upper bounds and $|H|$ lower bounds simultaneously. □

Now that we have a bound for both error directions, we can address Question 2 on page 125.

Theorem 3.14: Let

$$h_* = \arg\min_{h \in H} \operatorname{err}(h; P)$$

be the (unknown) optimal hypothesis in a finite class H, and let

$$\hat{h} = \arg\min_{h \in H} \widehat{\operatorname{err}}(h; S)$$

be the empirical risk minimiser. Then for any $0 < \varepsilon, \delta \leq 1$ and for

$$m \geq \frac{2}{\varepsilon^2} \ln \frac{2|H|}{\delta}$$

we have

$$\operatorname{err}(\hat{h}; P) \leq \operatorname{err}(h_*; P) + \varepsilon$$

with probability at least $1 - \delta$.

In other words, the Empirical Risk Minimiser is an agnostic PAC learner with sample complexity $(2/\varepsilon^2) \ln(2|H|/\delta)$ for hypothesis class H.

Proof: Since $\widehat{h}$ is the empirical risk minimiser, we have

$$\begin{aligned} \mathrm{err}(\widehat{h}) - \mathrm{err}(h_*) &= (\mathrm{err}(\widehat{h}) - \widehat{\mathrm{err}}(\widehat{h})) - (\mathrm{err}(h_*) - \widehat{\mathrm{err}}(\widehat{h})) \\ &\leq (\mathrm{err}(\widehat{h}) - \widehat{\mathrm{err}}(\widehat{h})) - (\mathrm{err}(h_*) - \widehat{\mathrm{err}}(h_*)). \end{aligned}$$

For m as stated, we have

$$\sqrt{\frac{1}{2m} \ln \frac{2|H|}{\delta}} \leq \frac{\varepsilon}{2}.$$

Corollary 3.13 then gives, with probability at leat $1 - \delta$,

$$(\mathrm{err}(\widehat{h}) - \widehat{\mathrm{err}}(\widehat{h})) - (\mathrm{err}(h_*) - \widehat{\mathrm{err}}(h_*)) \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

□

Our proof of Theorem 3.14 was based on Corollary 3.13 which is a uniform convergence result: it gives a bound for $\left|\mathrm{err}(h) - \widehat{\mathrm{err}}(h)\right|$ that is the same for all h.

Uniform convergence is an unnecessarily strong requirement, because we only need to bound $\left|\mathrm{err}(h) - \widehat{\mathrm{err}}(h)\right|$ for the two hypotheses $\widehat{h}$ and h_*. Furthermore, since actually the errors converge faster for hypotheses that have small error, we may get a lot of slack in the bound by requiring convergence for all $h \in H$.

However, currently the most practical bounds are all based on uniform convergence. This leads us to the question of what can be said of uniform convergence for an infinite hypothesis space H.

3.4 Infinite hypothesis classes

We say that a hypothesis class H has uniform convergence with sample size $s(\varepsilon, \delta)$ if for all $0 < \varepsilon, \delta \leq 1$ and for $m \geq s(\varepsilon, \delta)$ we have

$$\Pr_{S \sim P^m} \left(\sup_{h \in H} \left| \mathrm{err}(h) - \widehat{\mathrm{err}}(h) \right| > \varepsilon \right) \leq \delta.$$

The proof of Theorem 3.14 shows that uniform convergence with sample size $s(\varepsilon, \delta)$ guarantees that empirical risk minimisation gives agnostic PAC learning with sample complexity $s(\varepsilon/2, \delta)$.

Corollary 3.13 shows that for finite H we have uniform convergence with sample complexity $(1/2\varepsilon^2) \ln(2|H|/\delta)$. However, $|H|$ is a very crude measure for the "expressive power" of H, in particular since many interesting hypothesis classes are infinite.

It turns out that a better measure is the Vapnik-Chervonenkis (VC) dimension.

Let H be a class of binary classifiers over an infinite instance space X.

For any finite $D \subseteq X$, let $H_{\restriction D}$ be H restricted to D. That is,

$$H_{\restriction D} = \{\, h_{\restriction D} \mid h \in H \,\}$$

where $h_{\restriction D}$ is the function from D to $\{-1, 1\}$ with $h_{\restriction D}(x) = h(x)$ for all $x \in D$.

Further, let $\Pi_H(D) = |H_{\restriction D}|$ be the number of different ways we can classify the elements of D using classifiers from H. Clearly $\Pi_H(D) \leq 2^{|D|}$ for all H and D. If $\Pi_H(D) = 2^{|D|}$, we say that H shatters D. The Vapnik-Chervonenkis dimension of H is the largest cardinality of a shattered set:

$$\mathrm{VCdim}(H) = \max\{\, m \mid |H_{\restriction D}| = 2^m \text{ for some } D \text{ with } |D| = m \,\}.$$

If H shatters arbitrarily large sets, we write $\mathrm{VCdim}(H) = \infty$.

Example 3.15: Given a set $A \subseteq X$, let f_A be its indicator function:

$$f_A(x) = \begin{cases} 1 & \text{if } x \in A \\ -1 & \text{otherwise.} \end{cases}$$

Let $H_1 = \{ f_A \mid A \subseteq H \}$. Clearly H_1 shatters every $D \subset X$, so $\text{VCdim}(H_1) = \infty$.

Let $H_2 = \{ f_A \mid |A| \leq k \}$ for some k. Then H_2 shatters any D with $|D| \leq k$, and for $|D| > k$ we have

$$\Pi_{H_2}(D) = \sum_{i=1}^{k} \binom{|D|}{i} < 2^{|D|}.$$

Hence $\text{VCdim}(H_2) = k$. □

Example 3.16: Let $X = \mathbf{R}^n$ and H the class of linear classifiers. We show that $\text{VCdim}(H) = n + 1$.

To see $\text{VCdim}(H) \geq n + 1$, let $\boldsymbol{x}_{n+1} = \mathbf{0}$, and for $i = 1, \ldots, n$ let $\boldsymbol{x}_i$ be the ith unit vector: $x_{ij} = \delta_{ij}$ where δ_{ij} is Kronecker delta,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

We claim that H shatters $D = \{ \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1} \}$. To see that, let $(y_1, \ldots, y_{n+1}) \in \{ -1, 1 \}^{n+1}$ be arbitrary. By defining $w_i = y_i$ for $i = 1, \ldots, n$, and $b = -y_{n+1}/2$, we get

$$\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) = y_i$$

for all i.

To see VCdim$(H) \leq n+1$, suppose for contradiction that H shatters $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+2}\} \subset \mathbf{R}^n$. Then, by the familiar reduction, we see that there is a set $\{\tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_{n+2}\} \subset \mathbf{R}^{n+1}$ that is shattered by linear classifiers with bias 0. At least one $\tilde{\boldsymbol{x}}_i$ is a linear combination of the others. Without loss of generality, assume

$$\tilde{\boldsymbol{x}}_{n+2} = \sum_{i=1}^{n+1} a_i \boldsymbol{x}_i.$$

Choose now $y_i = \text{sign}(a_i)$ for $i = 1, \ldots, n+1$, and $y_{n+2} = -1$. Let $\boldsymbol{w}$ be such that $\text{sign}(\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_i) = y_i$ for $1 \leq i \leq n+1$. Then

$$\text{sign}(\boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_{n+2}) = \text{sign}\left(\sum_{i=1}^{n+1} a_i \boldsymbol{w} \cdot \tilde{\boldsymbol{x}}_i\right) = 1 \neq y_{n+2}.$$

Thus the labeling $(y_1, \ldots, y_{n+2})$ cannot be realised by a linear classifier. □

Although we will not prove it here, we remark in passing the key combinatorial result that makes the VC dimension important also outside learning theory. Let

$$\Pi_H(m) = \max_{|D|=m} \Pi_H(D).$$

If $\mathrm{VCdim}(H) = \infty$, we have $\Pi_H(m) = 2^m$ for all m. On the other hand, if $\mathrm{VCdim}(H)$ is finite, then $\Pi_H(m)$ is only polynomial in m:

Lemma 3.17 (Sauer): If $\mathrm{VCdim}(H) = d < \infty$, we have

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{\mathrm{e}m}{d}\right)^d.$$

□

We are now ready to state one of the main results in computational learning theory.

Theorem 3.18 (Vapnik and Chervonenkis, 1971): Assume $\mathrm{VCdim}(H) = d < \infty$. There is an absolute constant C such that

$$\Pr_{S \sim P^m} \left(\sup_{h \in H} \left| \mathrm{err}(h) - \widehat{\mathrm{err}}(h) \right| > \varepsilon \right) \leq \delta$$

holds whenever

$$m \geq \frac{C}{\varepsilon^2} \left(d \ln \frac{2}{\varepsilon} + \ln \frac{2}{\delta} \right).$$

(We have omitted some measurability assumptions that technically should be in the theorem.)

The original bound by Vapnik and Chervonenkis actually had an extra $\log d$ factor, which has since been removed by considerable technical effort. Nevertheless the bound is still too loose to give practically useful numerical error estimates.

Despite its looseness, the bound gives useful intuition: in order to avoid overfitting, one should control the VC dimension of the hypothesis class.

Let H be some class of functions $X \to Y$, and $L\colon H \times X \times Y \to [0, \infty)$. We define the loss class $L(H)$ as

$$L(H) = \{\, \ell_h \mid h \in H \,\}$$

where $\ell_h\colon X \times Y \to [0, \infty)$ is given by $\ell_h(x, y) = L(h, x, y)$.

In particular, let H be some class of binary classifiers and $\mathcal{F} = L_{0-1}(H)$. Then the uniform convergence requirement

$$\sup_{h \in H} \left| \mathrm{err}(h) - \widehat{\mathrm{err}}(h) \right| \leq \varepsilon$$

can be written equivalently

$$\sup_{f \in \mathcal{F}} \left| \mathop{\mathsf{E}}_{(x,y) \sim P} [f(x, y)] - \mathop{\mathsf{E}}_{(x,y) \sim S} [f(x, y)] \right| \leq \varepsilon. \qquad (*)$$

We now study uniform convergence of type $(*)$ for general classes $\mathcal{F}$ of functions $Z \to \mathbf{R}$, keeping in mind that the intended application is $Z = X \times Y$ and $\mathcal{F} = L(H)$.

Let r_i, $i = 1, \ldots, m$, be independent Rademacher random variables, *i.e.*, $r_i \in \{-1, 1\}$ and $\Pr(r_i = -1) = \Pr(r_i = 1) = 1/2$.

For a class $\mathcal{F}$ of functions $Z \to \mathbf{R}$ and a sample $S = (z_1, \ldots, z_m) \in Z^m$, define the empirical Rademacher complexity as

$$\hat{R}_m(\mathcal{F}, S) = \mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m} \left[\sup_{f \in \mathcal{F}} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right| \right].$$

The sum $\sum_i r_i f(z_i)$ is a measure of correlation between f and the labels $\boldsymbol{r}$. If $\hat{R}_m(\mathcal{F}, S)$ is high, then a random labelling of points $z_1, \ldots, z_m$ can be accurately matched by choosing a function from $\mathcal{F}$.

This means that $\mathcal{F}$ is rich enough that we are likely to find spurious regularities in the sample even if in reality there are none to be found. Thus we should not trust our empirical estimates too much.

Finally, given a probability measure P over Z, define the Rademacher complexity of $\mathcal{F}$ as

$$R_m(\mathcal{F}) = \mathop{\mathsf{E}}_{S\sim P^m}\left[\hat{R}_m(\mathcal{F}, S)\right].$$

Notice that unlike the VC dimension, the Rademacher complexity depends on P.

This leads to more accurate estimates when P is "easy", but makes it usually impossible to determine $R_m(\mathcal{F})$ exactly in closed form.

Fortunately, for large m we have with high probability

$$\hat{R}_m(\mathcal{F}, S) \approx R_m(\mathcal{F})$$

so in practice it is sufficient to determine the empirical Rademacher complexity for a sample S from P^m.

The main result is the following.

Theorem 3.19: Let $\mathcal{F}$ be a class of functions $Z \to [0,1]$, and let P be a probability measure on Z. Let $0 < \delta \leq 1$. For $S = (z_1, \ldots, z_m)$ drawn from P^m, with probability at least $1 - \delta$ we have

$$\sup_{f \in \mathcal{F}} \left| \mathop{\mathsf{E}}_{z \sim P}[f(z)] - \mathop{\mathsf{E}}_{z \sim S}[f(z)] \right| \leq R_m(\mathcal{F}) + \sqrt{\frac{1}{2m} \ln \frac{2}{\delta}}.$$

Assuming we know how to evaluate $R_m(\mathcal{F})$ where $\mathcal{F} = L(H)$ is the loss class we are interested in, we can use this to get a bound for the accuracy of empirical risk minimisation, as in Theorem 3.14.

Alternatively, we can simply interpret this as a training set bound:

$$\mathrm{err}(h) \leq \widehat{\mathrm{err}}(h) + R_m(\mathcal{F}) + \sqrt{\frac{1}{2m} \ln \frac{2}{\delta}}$$

holds with probability $1 - \delta$, for whichever hypothesis $h \in H$ we decided to choose.

Consider the case where $\mathcal{F}$ contains all functions from Z to $\{0,1\}$. Then $R_m(\mathcal{F}) = 1$, and the bound is vacuous (as expected).

However, we see later that for many interesting classes $\mathcal{F}$ we have $R_m(\mathcal{F}) = O(m^{-1/2})$. Then we can apply the following.

Corollary 3.20: Assume that $R_m(\mathcal{F}) \leq cm^{-1/2}$ for all m. Then for

$$m \geq \frac{1}{\varepsilon^2}\left(2c^2 + \ln\frac{2}{\delta}\right)$$

we have

$$\sup_{f\in\mathcal{F}}\left|\mathop{\mathsf{E}}_{z\sim P}[f(z)] - \mathop{\mathsf{E}}_{z\sim S}[f(z)]\right| \leq \varepsilon$$

with probability at least $1-\delta$.

Proof: Directly from the previous theorem using

$$\frac{1}{2}(\sqrt{a}+\sqrt{b}) \leq \sqrt{\frac{a+b}{2}}$$

(Jensen). □

Our proof of Theorem 3.19 is based on the following.

Theorem 3.21 (McDiarmid): Let $X_1, \ldots, X_m$ be independent random variables taking values in some set A. Assume that $f: A^m \to \mathbf{R}$ is such that for $i = 1, \ldots, m$ we have a constant c_i such that

$$\left| f(x_1, \ldots, x_m) - f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, x_m) \right| \leq c_i$$

for all $x_1, \ldots, x_m, x_i' \in A$. Write $\bar{f} = \mathsf{E}[f(X_1, \ldots, X_m)]$. Then for $\varepsilon > 0$ we have

$$\Pr(f(X_1, \ldots, X_m) - \bar{f} \geq \varepsilon) \leq \exp\left(\frac{-2\varepsilon^2}{\sum_{i=1}^m c_i^2} \right).$$

□

McDiarmid's inequality is an important example of a concentration inequality: it shows that under some assumptions, the distribution a random variable is closely concentrated around its expectation.

As an example, we derive another important concentration inequality as a special case of McDiarmid's. This is a more general version of the Hoeffding bound stated earlier.

Corollary 3.22 (Hoeffding): For $i = 1, \ldots, m$, let X_i be independent random variables with $a_i \leq X_i \leq b_i$ and $\mathsf{E}[X_i] = \bar{X}_i$. Let $S = \sum_{i=1}^{m} X_i$. Then

$$\Pr\left(\left|S - \sum_{i=1}^{m} \bar{X}_i\right| \geq \varepsilon\right) \leq 2 \exp \frac{-2\varepsilon^2}{\sum_{i=1}^{m} (b_i - a_i)^2}.$$

Proof: We apply McDiarmid separately to S and $-S$. The union bound gives the factor 2. □

Proof of Theorem 3.19: For $f \in \mathcal{F}$ we have

$$\mathop{\mathsf{E}}_{z\sim P}[f(x)] \leq \mathop{\mathsf{E}}_{z\sim S}[f(z)] + p(S)$$

where

$$p(S) = \sup_{g\in\mathcal{F}} \left(\mathop{\mathsf{E}}_{z\sim P}[g(z)] - \mathop{\mathsf{E}}_{z\sim S}[g(z)] \right).$$

We apply McDiarmid's inequality to function p. Since $g(z) \in [0,1]$, changing one z_i in S can change $\mathsf{E}_{z\sim S}[g(z)]$ by at most $1/m$. We solve for ε such that

$$\exp\left(-2m\varepsilon^2\right) = \frac{\delta}{2}$$

and plug into McDiarmid to get

$$p(S) \leq \mathop{\mathsf{E}}_{S\sim P^m}[p(S)] + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}$$

with probability at least $1 - \delta/2$.

The key to the whole proof is now estimating $\mathsf{E}_{S\sim P^m}[p(S)]$.

First we express $\mathsf{E}_{z\sim P}[g(z)]$ in terms of a hypothetical sample $S' = (z'_1, \ldots, z'_m)$:

$$\begin{aligned}
\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] &= \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in\mathcal{F}}\left(\mathop{\mathsf{E}}_{z\sim P}[g(z)] - \mathop{\mathsf{E}}_{z\sim S}[g(z)]\right)\right] \\
&= \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in\mathcal{F}}\left(\mathop{\mathsf{E}}_{S'\sim P^m}\left[\frac{1}{m}\sum_{i=1}^m g(z'_i)\right] - \frac{1}{m}\sum_{i=1}^m g(z_i)\right)\right].
\end{aligned}$$

We then apply the fact $\sup_f \mathsf{E}[f] \leq \mathsf{E}[\sup_f f]$ to get

$$\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] \leq \mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^m (g(z'_i) - g(z_i))\right)\right].$$

Let now $r_1, \ldots, r_m$ be independent Rademacher random variables. Since $g(z'_i)$ and $g(z'_i)$ have same distribution, $g(z'_i) - g(z_i)$ has the same distribution as $r_i(g(z'_i) - g(z_i))$, and we get

$$\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] \leq \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\mathop{\mathsf{E}}_{S,S'\sim P^m}\left[\sup_{g\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^m r_i(g(z'_i) - g(z_i))\right)\right].$$

Now the triangle inequality gives

$$\mathop{\mathsf{E}}_{S\sim P^m}[p(S)] \le 2 \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m} \mathop{\mathsf{E}}_{S\sim P^m}\left[\sup_{g\in\mathcal{F}}\left|\frac{1}{m}\sum_{i=1}^{m} r_i g(z_i)\right|\right] = R_m(\mathcal{F}).$$

By applying the same argument to $-f$ and then taking the union bound we see that

$$\left|\mathop{\mathsf{E}}_{z\sim P}[f(x)] - \mathop{\mathsf{E}}_{z\sim S}[f(z)]\right| \le R_m(\mathcal{F}) + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}$$

holds with probability $1-\delta$. □

To apply the bound, we of course need to be able to estimate $R_m(\mathcal{F})$.

Theorem 3.23: For a fixed $S \in Z^m$ we have

$$R_m(\mathcal{F}) \leq \hat{R}_m(\mathcal{F}, S) + \sqrt{\frac{2}{m} \ln \frac{1}{\delta}}$$

with probability at least $1 - \delta$ over the random choice of $\boldsymbol{r}$.

We also have

$$R_m(\mathcal{F}) \leq \sup_{f \in \mathcal{F}} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right| + \sqrt{\frac{8}{m} \ln \frac{1}{\delta}}$$

with probability at least $1 - \delta$ over the random choice of $\boldsymbol{r}$ and S.

This shows that instead of calculating expectations, we can just solve a single maximisation and have an upper bound with high confidence.

Proof: We prove the second estimate. The first one is proven similarly.

Let $q_i = (r_i, z_i) \in \{-1, 1\} \times Z$, and define

$$p(q_1, \ldots, q_m) = \sup_{f \in \mathcal{F}} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f(z_i) \right|.$$

Since we assume $f(z_i) \in [0, 1]$, we can apply McDiarmid's inequality with $c_i = 4/m$. Solving for ε such that

$$\exp\left(-m\varepsilon^2/8\right) = \delta$$

gives

$$\varepsilon = \sqrt{\frac{8}{m} \ln \frac{1}{\delta}}.$$

□

Assume now that $\mathcal{F} = L_{0-1}(H)$ is the discrete loss class for some class H of classifiers, and H is closed under complementation (*i.e.*, if $h \in H$ then $-h \in H$). We show how empirical risk minimisation can be used to calculate $R_m(\mathcal{F})$.

Let $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times Y)^m$ and $\boldsymbol{r} \in \{-1, 1\}^m$ be fixed. Write $S' = ((x_1, r_1 y_1), \ldots, (x_m, r_m y_m))$. Since $L_{0-1}(h, x, y) = \frac{1}{2}(1 - yh(x))$, we have

$$\begin{aligned}
2\sum_{i=1}^{m} r_i L_{0-1}(h, x_i, y_i) &= \sum_{i=1}^{m} r_i - \sum_{i=1}^{m} r_i y_i h(x_i) \\
&= \sum_{i=1}^{m} r_i - \sum_{i=1}^{m} (1 - 2L_{0-1}(h, x_i, r_i y_i)) \\
&= \sum_{i=1}^{m} r_i - m + 2\sum_{i=1}^{m} (1 - L_{0-1}(-h, x_i, r_i y_i))) \\
&= \sum_{i=1}^{m} r_i + m - 2m\widehat{\mathrm{err}}(-h, S').
\end{aligned}$$

Similarly

$$-2\sum_{i=1}^{m} r_i L_{0-1}(h, x_i, y_i) = -\sum_{i=1}^{m} r_i + m - 2m\widehat{\mathrm{err}}(h, S').$$

Write $\hat{\varepsilon} = \inf_{h\in H} \widehat{\mathrm{err}}(h; S')$, which by our assumption means also $\hat{\varepsilon} = \inf_{h\in H} \widehat{\mathrm{err}}(-h; S')$. We have

$$\sup_{f\in\mathcal{F}} \left| \frac{2}{m} \sum_{i=1}^{m} r_i f((x_i, y_i)) \right| = 1 - 2\hat{\varepsilon} + \frac{1}{m} \left| \sum_{i=1}^{m} r_i \right|.$$

Thus, we get an estimate for $R_m(\mathcal{F})$ by computing $\hat{\varepsilon}$ by empirical risk minimisation and plugging the above to Theorem 3.23.

Unfortunately, empirical risk minimisation is computationally intractable for most interesting classes H. Also in practice, the problem of evaluating $R_m(\mathcal{F})$ seriously restricts the applicability of Rademacher complexities.

Rademacher complexities can be crudely estimated in terms of the VC dimension. This of course gives only an upper bound that is worst case with respect to P.

Theorem 3.24: Let $\mathcal{F} = L_{0-1}(H)$ where $\mathrm{VCdim}(H) = d < \infty$, and let $m \geq d$. Then

$$R_m(\mathcal{F}) \leq 5\sqrt{\frac{d+1}{m}\left(\ln\frac{m}{d}+1\right)}.$$

Remark: A stronger result $R_m(\mathcal{F}) = O(\sqrt{d/m})$ is also known. Plugging that into Corollary 3.20 gives the sample complexity

$$m = O\left(\frac{1}{\varepsilon^2}\left(d + \ln\frac{1}{\delta}\right)\right).$$

Here we only give the proof for the weaker bound of Theorem 3.24 which illustrates the connection between combinatorial and statistical parameters.

Proof of Theorem 3.24: For $0 < \delta \leq 1$, let

$$\rho_\delta = \sqrt{\frac{8}{m}\left(d\left(\ln\frac{m}{d}+1\right)+\ln\frac{2}{\delta}\right)}.$$

Write also

$$Q(S,\boldsymbol{r}) = \sup_{f\in\mathcal{F}}\left|\frac{2}{m}\sum_{i=1}^{m} r_i f(z_i)\right|.$$

We show that with probability at least $1-\delta$ over random choice of $S \sim P^m$ and $\boldsymbol{r} \in \{-1,1\}^m$, we have

$$Q(S,\boldsymbol{r}) \leq \rho_\delta.$$

Since always $Q(S,\boldsymbol{r}) \leq 2$, choosing $\delta = d/m$ will then give the desired bound

$$\begin{aligned}
\mathop{\mathrm{E}}_{S,\boldsymbol{r}}[Q(S,\boldsymbol{r})] &\leq (1-\delta)\rho_\delta + \delta\cdot 2 \\
&\leq \sqrt{\frac{8}{m}\left(d\left(\ln\frac{m}{d}+1\right)+\ln\frac{m}{d}+\ln 2\right)}+\frac{2d}{m} \\
&\leq 5\sqrt{\frac{d+1}{m}\left(\ln\frac{m}{d}+1\right)}.
\end{aligned}$$

Consider a fixed $S = ((x_1, y_1), \ldots, (x_m, y_m))$.

We call any sequence $\boldsymbol{\ell} \in \{0,1\}^m$ a label sequence, and say that a label sequence $\boldsymbol{\ell}$ is valid if for some $h \in H$ we have $\ell_i = L_{0-1}(h, x_i, y_i)$.

Fix $\rho \geq 0$. Now $Q(S, \boldsymbol{r}) \geq \rho$ holds iff for some valid label sequence $\boldsymbol{\ell}$ we have

$$\left| \frac{2}{m} \sum_{i=1}^{m} r_i \ell_i \right| \geq \rho.$$

In this case we say that $\boldsymbol{\ell}$ covers $\boldsymbol{r}$.

Clearly there are $|H_{\restriction D}|$ different valid label sequences, where $D = \{x_1, \ldots, x_m\}$. By Sauer's lemma,

$$|H_{\restriction D}| \leq \left(\frac{em}{d}\right)^d.$$

Consider first a fixed label sequence $\boldsymbol{\ell}$, and let $I = \{\, i \in \{\, 1, \ldots, m \,\} \mid \ell_i = 1 \,\}$. Then

$$\Pr_{\boldsymbol{r}}\left(\frac{2}{m}\sum_{i=1}^{m} r_i \ell_i \geq \rho\right) = \Pr_{\boldsymbol{r}}\left(\sum_{i\in I}\frac{1+r_i}{2} \geq \frac{|I|}{2} + \frac{m\rho}{4}\right).$$

Similarly

$$\Pr_{\boldsymbol{r}}\left(\frac{2}{m}\sum_{i=1}^{m} r_i \ell_i \leq -\rho\right) = \Pr_{\boldsymbol{r}}\left(\sum_{i\in I}\frac{1+r_i}{2} \leq \frac{|I|}{2} - \frac{m\rho}{4}\right).$$

The random variable $\sum_{i\in I}(1+r_i)/2$ has distribution $\mathrm{Bin}(|I|, 1/2)$. By Hoeffding's inequality, the probability of drawing $\boldsymbol{r}$ that is covered by $\boldsymbol{\ell}$ is

$$\begin{aligned}
\Pr_{\boldsymbol{r}}\left(\left|\frac{2}{m}\sum_{i=1}^{m} r_i \ell_i\right| \geq \rho\right) &= \Pr_{\boldsymbol{r}}\left(\left|\frac{1}{|I|}\sum_{i\in I}\frac{1+r_i}{2} - \frac{1}{2}\right| \geq \frac{m\rho}{4|I|}\right) \\
&\leq 2\exp\left(-2|I|(m\rho/(4|I|))^2\right) \\
&\leq 2\exp\left(-m\rho^2/8\right).
\end{aligned}$$

Therefore, a single label sequence $\boldsymbol{\ell}$ covers at most a proportion $2\exp(-m\rho^2/8)$ of sequences $\boldsymbol{r} \in \{-1, 1\}^m$.

Since there are at most $(em/d)^d$ valid sequences, the proportion of all covered sequences $\boldsymbol{r}$ is at most

$$2\left(\frac{em}{d}\right)^d \exp(-m\rho^2/8).$$

By choosing $\rho = \rho_\delta$ this becomes δ.

We have shown

$$\Pr_{\boldsymbol{r}}[Q(S, \boldsymbol{r}) \geq \rho_\delta] \leq \delta$$

for any fixed S. This implies

$$\Pr_{S,\boldsymbol{r}}[Q(S, \boldsymbol{r}) \geq \rho_\delta] \leq \delta.$$

(As usual, we ignore some measurability assumptions.) $\square$

We now use the Rademacher theory to develop a margin-based generalisation bound for linear classifiers.

For $B > 0$, let $\mathcal{F}_B$ be the function class

$$\mathcal{F}_B = \{ (\boldsymbol{x}, y) \mapsto y\boldsymbol{w} \cdot \boldsymbol{x} \mid \|\boldsymbol{w}\|_2 \leq B \}.$$

Theorem 3.25: For any $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m))$ we have

$$\hat{R}_m(\mathcal{F}_B, S) \leq \frac{2B}{m} \sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_t\|_2^2}.$$

Notice that if $\|\boldsymbol{x}_t\|_2 \leq X$, the bound becomes $2BX/\sqrt{m}$. However, this does not yet give a loss bound, since

- functions $f \in \mathcal{F}_B$ are not bounded to $[0, 1]$, and
- $\mathcal{F}$ is not a loss class for any interesting L.

Proof: If $f(\boldsymbol{x}, y) = y\boldsymbol{w} \cdot \boldsymbol{x}$ and $\|\boldsymbol{w}\|_2 \leq B$, we have

$$\left|\sum_{i=1}^{m} r_i f(\boldsymbol{x}_i, y_i)\right| = \left|\sum_{i=1}^{m} r_i y_i \boldsymbol{w} \cdot \boldsymbol{x}_i\right| = \left|\boldsymbol{w} \cdot \left(\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right)\right| \leq B\left\|\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right\|_2$$

by Minkowski's inequality $|\boldsymbol{w} \cdot \boldsymbol{x}| \leq \|\boldsymbol{w}\|_2 \|\boldsymbol{x}\|_2$. Therefore

$$\begin{aligned}
\hat{R}_m(\mathcal{F}, S) &= \mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m}\left[\sup_{f \in \mathcal{F}_B}\left|\frac{2}{m}\sum_{i=1}^{m} r_i f(\boldsymbol{x}_i)\right|\right] \\
&\leq \frac{2B}{m}\mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m}\left[\left\|\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right\|_2\right] \\
&= \frac{2B}{m}\mathop{\mathsf{E}}_{\boldsymbol{r} \in \{-1,1\}^m}\left[\sqrt{\left(\sum_{i=1}^{m} r_i y_i \boldsymbol{x}_i\right) \cdot \left(\sum_{j=1}^{m} r_j y_j \boldsymbol{x}_j\right)}\right].
\end{aligned}$$

Since $\sqrt{\cdot}$ is concave, we can apply Jensen to get

$$\begin{aligned}
&\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sqrt{\left(\sum_{i=1}^m r_i y_i \boldsymbol{x}_i\right)\cdot\left(\sum_{j=1}^m r_j y_j \boldsymbol{x}_j\right)}\right] \\
&\qquad\leq \sqrt{\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\left(\sum_{i=1}^m r_i y_i \boldsymbol{x}_i\right)\cdot\left(\sum_{j=1}^m r_j y_j \boldsymbol{x}_j\right)\right]} \\
&\qquad= \sqrt{\sum_{i,j=1}^m \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}[r_i r_j y_i y_j \boldsymbol{x}_i\cdot\boldsymbol{x}_j]}.
\end{aligned}$$

Since $\mathsf{E}[r_i r_j] = \delta_{ij}$ and $y_i \in \{-1,1\}$, the claim follows. □

A function $f: \mathbf{R} \to \mathbf{R}$ has Lipschitz constant c if it satisfies

$$|f(x) - f(y)| \leq c\,|x - y|$$

for all $x, y \in \mathbf{R}$.

The class we are ultimately interested here is the discrete loss class for norm-bounded linear classifiers

$$T \circ \mathcal{F}_B = \{\, (\boldsymbol{x}, y) \mapsto T(f(\boldsymbol{x}, y)) \mid f \in \mathcal{F}_B \,\}$$

where $T(p) = 0$ if $p > 0$ and $T(p) = 1$ if $p \leq 0$.

For technical reasons we introduce for $\mu \geq 0$ the function A_μ with

$$A_\mu(p) = \begin{cases} 1 & \text{if } p \leq 0 \\ 1 - p/\mu & \text{if } 0 < p < \mu \\ 0 & \text{if } \mu \leq p. \end{cases}$$

Then A_μ has Lispschitz constant $1/\mu$, and $A_\mu(p) \geq T(p)$ for all p.

Theorem 3.26: If function G has Lipschitz constant c and $G(0) = 0$, then

$$\hat{R}_m(G \circ \mathcal{F}, S) \leq 2c\hat{R}_m(\mathcal{F}, S)$$

for all S. $\square$

(The proof of this theorem is technical and we will not go into it.)

In particular, if we consider $G = A_\mu - 1$ to get $G(0) = 0$, we see that

$$\begin{aligned} \hat{R}_m((A_\mu - 1) \circ \mathcal{F}_B, S) &\leq \frac{2}{\mu}\hat{R}_m(\mathcal{F}_B, S) \\ &\leq \frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_i\|_2^2}. \end{aligned}$$

Recall the hinge loss $L_\mu(\boldsymbol{w}, \boldsymbol{x}, y) = \max\{0, \mu - y\boldsymbol{w} \cdot \boldsymbol{x}\}$.

Theorem 3.27: Fix $\mu > 0$ and $B > 0$ and a probability measure P over $\mathbf{R}^n \times \{-1, 1\}$. With probability at least $1 - \delta$ over drawing $S = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)) \sim P^m$, we have

$$\begin{aligned} \Pr_{(x,y)\sim P}(\text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}) \neq y) &= \mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w} \cdot \boldsymbol{x})] \\ &\leq \frac{1}{\mu m}\sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \boldsymbol{x}_i, y_i) \\ &\quad + \frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m}\|\boldsymbol{x}_i\|_2^2} + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}} \end{aligned}$$

for all $\boldsymbol{w}$ with $\|\boldsymbol{w}\|_2 \leq B$.

Remarks:

- If $\boldsymbol{w}$ has margin μ, we have $L_\mu(\boldsymbol{w}, \boldsymbol{x}_i, y_i) = 0$ for all i, and the bound becomes

$$\frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m} \|\boldsymbol{x}_i\|_2^2} + 3\sqrt{\frac{1}{2m} \ln \frac{4}{\delta}}.$$

- The bound depends only on empirical quantities, with no assumptions about P. If we are "lucky" and get a sample that allows a large margin, we also get a good bound.

- The bound can be kernelised. Suppose we use kernel k associated with feature map $\boldsymbol{\psi}$. Then $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \boldsymbol{\psi}(x_i)$ for some α_i. The relevant quantities in the bound become

$$\sum_{i=1}^{m} \|\boldsymbol{\psi}(x_i)\|_2^2 = \sum_{i=1}^{m} k(x_i, x_i) \qquad \text{and} \qquad \|\boldsymbol{w}\|_2^2 = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j).$$

Proof: Since $T - 1 \leq A_\mu - 1$, we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] \leq \mathop{\mathsf{E}}_{(x,y)\sim P}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1].$$

From Theorem 3.19, we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] \leq \mathop{\mathsf{E}}_{(x,y)\sim S}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) - 1] + R_m((A_\mu - 1)\circ\mathcal{F}_B) + \sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

for all $\boldsymbol{w}$ with probability $1 - \delta/2$. From Theorem 3.23, we have

$$R_m((A_\mu - 1)\circ\mathcal{F}_B) \leq \hat{R}_m((A_\mu - 1)\circ\mathcal{F}_B, S) + 2\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

with probability $1 - \delta/2$.

Since $\mathsf{E}_{(x,y)\sim P}[-1] = \mathsf{E}_{(x,y)\sim S}[-1] = -1$, we get

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[T(y\boldsymbol{w}\cdot\boldsymbol{x})] \quad \leq \quad \mathop{\mathsf{E}}_{(x,y)\sim S}[A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x})] + \hat{R}_m((A_\mu - 1)\circ\mathcal{F}_B, S) + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

with probability $1-\delta$. We notice that

$$\mu A_\mu(y\boldsymbol{w}\cdot\boldsymbol{x}) \leq L_\mu(\boldsymbol{w},\boldsymbol{x},y)$$

and plug in the estimate for $\hat{R}_m((A_\mu - 1)\circ\mathcal{F}_B, S)$. □

Again, one should not expect this bound to give numerical estimates that are useful in practice. Getting practically useful bounds is still work in progress. Currently the so-called "PAC-Bayesian" approach seems promising, but we will not go into it here.

However, margin bounds like this are currently the main explanation for why kernel based methods, such as support vector machines, work so well.

Model selection

Assume we have a hierarchy of hypothesis classes $H_1 \subseteq \cdots \subseteq H_k$, and for each i a learning algorithm A_i that outputs hypotheses from H_i.

We have a sample S and run each algorithm A_i separately on it, resulting in hypotheses $h_i \in H_i$, $i = 1, \ldots, k$.

The question is, which h_i should we choose, *i.e.*, which hypothesis class (or model using statistical terminology) should we prefer.

One method (and usually the best in practice) is to use cross validation.

An alternative would be to use training set bounds. Indeed, this is one often quoted motivation for studying such bounds. Let us look into it in a little more detail.

Assume we have for each i a training set bound that gives

$$\mathrm{err}(h_i) \leq \widehat{\mathrm{err}}(h_i) + r_i(S, \delta)$$

with probability at least $1 - \delta$. For example, we could take the Rademacher bound (Theorem 3.19) or the VC bound (Theorem 3.18).

Let now $\varepsilon_i = r_i(S, \delta/k)$. Then with probability at least $1 - \delta$ we have

$$\mathrm{err}(h_i) \leq \widehat{\mathrm{err}}(h_i) + \varepsilon_i$$

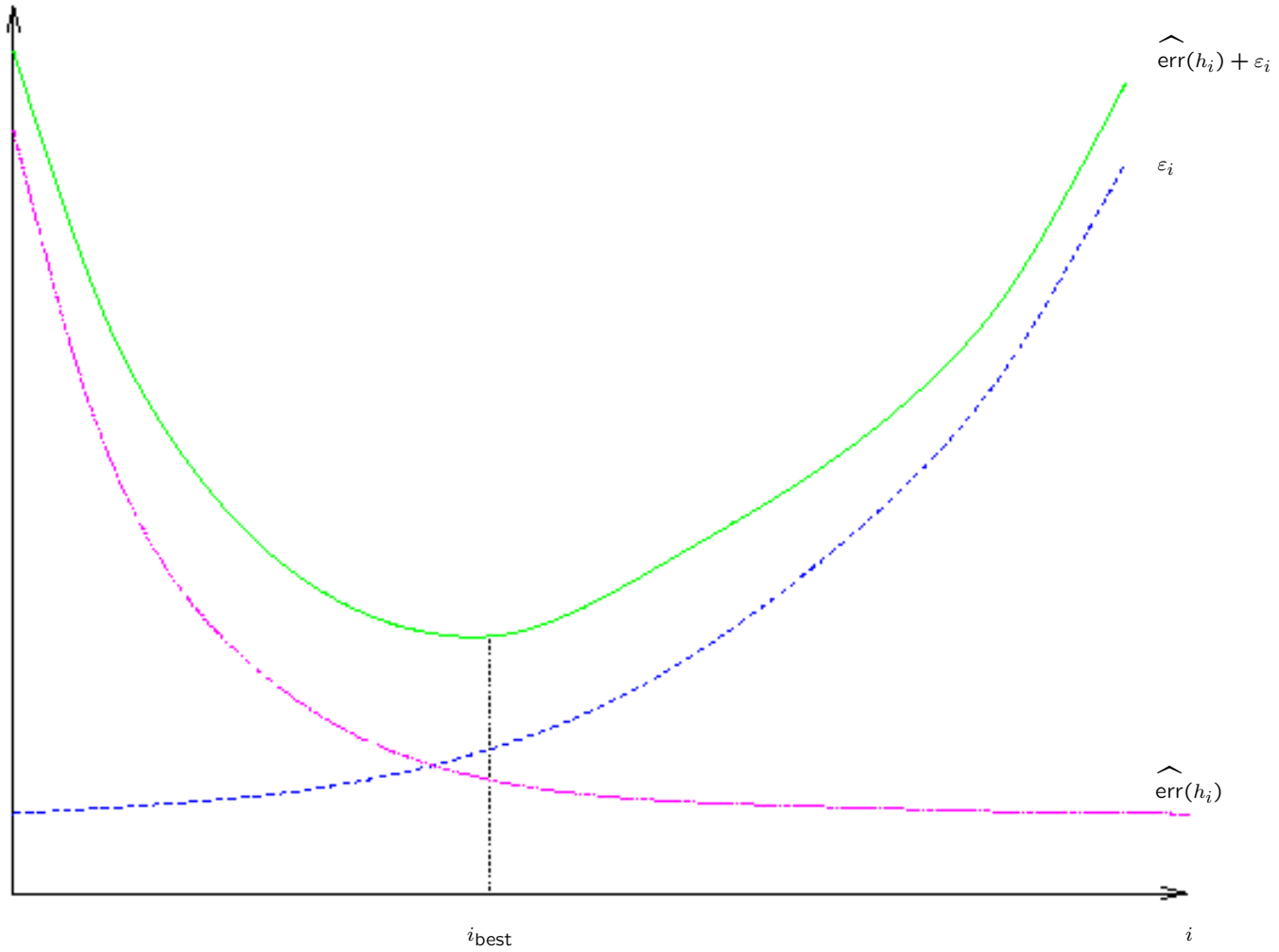
for *all* i (union bound). We propose choosing h_i such that

$$\widehat{\mathrm{err}}(h_i) + \varepsilon_i$$

is minimised.

In particular, if each h_i is obtained by empirical risk minimisation, this procedure is called structural risk minimisation (SRM).

Since $H_i \subseteq H_{i+1}$, we generally expect $\widehat{\mathrm{err}}(h_i) \geq \widehat{\mathrm{err}}(h_{i+1})$ and $\varepsilon_i \leq \varepsilon_{i+1}$.


$\widehat{\text{err}}(h_i) + \varepsilon_i$
ε_i
$\widehat{\text{err}}(h_i)$
i_{best}
i

We can also apply this with an infinite hierarchy $(H_i)_{i\in\mathbf{N}}$. We choose a sequence $(\delta_i)_{i\in\mathbf{N}}$ such that $\sum_i \delta_i = \delta$, for example $\delta_i = 6\delta/\pi^2 i^2$. Then we take $\epsilon_i = r_i(S, \delta_i)$.

Since ϵ_i goes to infinity as i increases, there is only some finite number of values i we need to consider.

In practice, the error bounds we can obtain with current methods are still so loose that this method is not really justified theoretically. Cross validation or using a test set give better results.

3.5 Online vs. statistical learning

Suppose we have a statistical (*i.e.*, batch) learning algorithm A that takes as input a sample S and returns a hypothesis $A(S)$. How can we apply it in an online setting?

In principle, we can at each step t compute $h_t = A(S_{t-1})$ where $S_{t-1} = ((x_1, y_1), \ldots, (x_{t-1}, y_{t-1}))$, and predict $\widehat{y}_t = h_t(x_t)$. If we have a statistical error bound for A, we get an online bound for the expected number of errors assuming the data are i.i.d.

In practice, this naive procedure can take too much computation time, or memory. Online applications such as signal processing can have very tight resource constraints.

The challenge in converting batch algorithms to an online setting is in creating an incremental version of the algorithm.

An incremental algorithm should be able to compute $h_{t+1} = A(S_t)$ directly from $h_t = A(S_{t-1})$ and (x_t, y_t), without access to (x_i, y_i) for $i < t$.

Developing incremental versions of popular batch learning algorithms is mostly an open research problem, to which we will not go here.

Instead, we consider the reverse problem: given an online algorithm (and a mistake bound), what is a good way of converting it into a batch algorithm (and an error bound).

The main motivation is that online algorithms are typically easy to implement, and efficient in terms of running time and memory. Purpose-built batch algorithms will in general have better accuracy, assuming one has the resources to use them.

The most obvious method is the following.

1. Run the online algorithm repeatedly through S until the mistake frequency stops improving.

2. Output the final hypothesis.

We call this the naive online-to-batch conversion.

Unless the data are quite easy, one cannot expect convergence (*i.e.*, achieving zero mistake rate). This means the hypothesis will keep changing until the very end.

Therefore the method can be **very sensitive** to things like the ordering of the data and the exact time when we stop.

The Pocket Algorithm (or "longest survivor") is a traditional way of trying to increase the stability of the naive conversion.

Algorithm 3.28 (Pocket Algorithm): Here h is the current hypothesis of the online algorithm.

Initialise $h_{\text{best}} := h$, $s_{\text{best}} := 0$, $s := 0$.
Run the algorithm repeatedly through S.
At each time step
    if the hypothesis h did not make a mistake:
        $s := s + 1$
        if $s > s_{\text{best}}$ then
            $h_{\text{best}} := h$ (put h in "pocket")
            $s_{\text{best}} := s$
        Do not update h.
    if the hypothesis h made a mistake:
        Update h.
        $s := 0$.
When h_{best} does not change any more, output h_{best}.

The theory of the Pocket Algorithm is not well understood, but it is worth trying as an alternative to the naive method.

A more principled method is the randomised conversion:

1. Run the algorithm through $S = ((x_1, y_1), \ldots, (x_m, y_m))$ to obtain $m + 1$ hypotheses $h_1, \ldots, h_{m+1}$.

2. Output the randomised hypothesis that on input x

   (a) picks an index $i \in \{1, \ldots, m\}$ uniformly at random and

   (b) returns $h_i(x)$.

Having a randomised hypothesis may of course be undesirable.

Even if we do not mind randomness, storing all the m hypotheses can be a problem.

Notice that we exclude h_{t+1} from the hypothesis. From the point of view of the online algorithm (and any mistake bound we may have for it), the hypothesis h_{t+1} does not get tested and we have in principle no reason to assume it would be any good.

If the label space Y is convex, for example $Y = \mathbf{R}$ or $Y = [a, b]$ for some $a, b \in \mathbf{R}$, we may replace the randomised hypothesis by the averaged hypothesis $\bar{h}$ where

$$\bar{h}(x) = \frac{1}{m} \sum_{i=1}^{m} h_i(x).$$

If the loss function L is also convex, this does not increase the loss:

$$L\left(\frac{1}{m} \sum_{i=1}^{m} h_i(x)\right) \leq \frac{1}{m} \sum_{i=1}^{m} L(h_i(x))$$

where the left-hand side has the loss of the averaged hypothesis and the right-hand side the expected loss of the randomised hypothesis.

For linear online classifiers, in practice it is usually best to take the average weight vector:

$$h(\boldsymbol{x}) = \text{sign}\left(\left(\frac{1}{m}\sum_{i=1}^{m}\boldsymbol{w}_i\right)\cdot\boldsymbol{x}\right).$$

Notice that this is quite different from taking the averaged classifier

$$\frac{1}{m}\sum_{i=1}^{m}\text{sign}\left(\boldsymbol{w}_i\cdot\boldsymbol{x}\right).$$

Since the discrete loss is **not** convex, the theoretical justification is a bit unclear. We shall soon return to the issue of proving loss bounds for randomised and averaged conversions.

The final method we consider is called penalised empirical risk minimisation. Assume $0 \leq L(x) \leq B$ for all x and pick a confidence parameter $0 < \delta \leq 1$.

1. Run the algorithm through S to obtain $m+1$ hypotheses $h_1, \ldots, h_{m+1}$.

2. For $t = 1, \ldots, m$, calculate the risk estimate

$$\widehat{\varepsilon}_t = \frac{1}{m-t+1} \sum_{i=t}^{m} L(y_i, h_t(x_i))$$

   and the penalty term

$$p_t = B\sqrt{\frac{2}{m-t+1} \ln \frac{4m}{\delta}}.$$

3. Output h_t for which $\widehat{\varepsilon}_t + p_t$ is minimised.

The estimate $\widehat{\varepsilon}_t$ is based on data not used for learning h_t. Assuming i.i.d. data, $\widehat{\varepsilon}_t$ is therefore an unbiased estimate of err(h_t).

Unfortunately the estimates $\widehat{\varepsilon}_t$ get very unreliable towards the end of the sequence, since the amount of test data gets small. This is reflected in the penalties p_t getting very large at the end, so the choice favours the early hypotheses.

The analysis giving the penalty term is a little slack, and consequently the method is not that good in practice. However, one may expect this to improve, as the whole method is still quite new.

We will now get into theoretical analysis. The penalised ERM is close in spirit to SRM. However, since some independence assumptions are no longer valid, we need slightly more general tools from probability theory.

Martingales

Our analysis of statistical learning has been based on noticing that with high probability, the a sum of independent random variables is close to its expectation.

Similar estimates can be obtained when the random variable are not independent by considering martingales.

Martingales are a simple and powerful tool and have many applications in analysing randomised algorithms (and of course outside computer science, too).

Historically "martingale" refers to a betting strategy where a player doubles his bet every time he looses. The idea is to guarantee ending up with a net gain, but the (noticeable) risk is that bancruptcy occurs first. We are interested in more conservative "betting strategies".

Warning: The formal definitions make the concept look a lot more complicated than it really is. Try to concentrate on the intuitive ideas.

As a basic example, assume that at time t, the player bets b_t euros. Then a coin is flipped, with probability $1/2$ of coming up heads. If the outcome is heads, the player gains b_t euros; otherwise he loses b_t euros.

Let X_t be the player's wealth (in euros) after the tth coin flip, and $Y_t = X_t - X_{t-1}$. Notice that $\mathsf{E}[Y] = 0$ and $\mathsf{E}[X_t] = X_0$ (the initial wealth).

Consider first the special case $b_t = 1$ for all t. Using the Hoeffding bound for binomial distribution, it is easy to see that for any $0 < \varepsilon \leq 1/2$ we have

$$\Pr\left(|X_t - \mathsf{E}[X_t]| \geq \varepsilon t\right) \leq 2\exp\left(-t\varepsilon^2/2\right).$$

Suppose now that we allow b_t to depend on X_{t-1}. Then the Y_t are not independent any more, so we need more powerful tools.

Consider two discrete random variables X and Y with $\Pr(X = x, Y = y) = p(x, y)$. The conditional expectation of X given $Y = y$ is a constant given by

$$\mathsf{E}[X \mid Y = y] = \sum_x x \Pr(X = x \mid Y = y) = \frac{\sum_x xp(x, y)}{\sum_x p(x, y)}.$$

The conditional expectation of X given Y is the random variable $\mathsf{E}[X \mid Y]$ where

$$\Pr(\mathsf{E}[X \mid Y] = r) = \sum_{\mathsf{E}[X|Y=y]=r} \Pr(Y = y).$$

This can be generalised to continuous random variables by replacing the probabilities with density functions in the obvious manner.

Example 3.29: Consider the probability space where the sample space is all Finnish tax payers. Let Y be the annual income of a random taxpayer (in euros), and X the tax rate of a random taxpayer.

Then $\mathsf{E}[X \mid Y = 30000]$ is the average tax rate of those with income exactly 30 000 euros.

Further, write $r_y = \mathsf{E}[X \mid Y = y]$ for all possible income levels y, and assume all the values r_y are distinct. Then $\mathsf{E}[X \mid Y]$ is the random variable that gets the value r_y with probability $\Pr(Y = y)$. □

We can of course condition over several r.v.'s simultaneously.

Example 3.30: Throw a 6-sided die 10 times, and for $i = 1, \ldots, 6$ let X_i be the number of times we get result i. Then

$$\begin{aligned} \mathsf{E}[X_1 \mid X_2 = 3] &= \frac{10-3}{5} = \frac{7}{5} \\ \mathsf{E}[X_1 \mid X_2 = 3, X_3 = 4] &= \frac{10-3-4}{4} = \frac{3}{4} \end{aligned}$$

and

$$\begin{aligned} \mathsf{E}[X_1 \mid X_2] &= \frac{10-X_2}{5} \\ \mathsf{E}[X_1 \mid X_2, X_3] &= \frac{10-X_2-X_3}{4}. \end{aligned}$$

□

The following basic equalities follow directly from the definitions.

$$\begin{aligned} \mathsf{E}[\mathsf{E}[X \mid Y]] &= \mathsf{E}[X] \\ \mathsf{E}[Y \cdot \mathsf{E}[X \mid Y]] &= \mathsf{E}[XY]. \end{aligned}$$

More generally, for any Z we have

$$\begin{aligned} \mathsf{E}[\mathsf{E}[X \mid Y, Z] \mid Z] &= \mathsf{E}[X \mid Z] \\ \mathsf{E}[Y \cdot \mathsf{E}[X \mid Y, Z] \mid Z] &= \mathsf{E}[XY \mid Z]. \end{aligned}$$

Example 3.31: Continuing with Finnish tax payers, the average tax rate for a tax payer with random income is just the total average tax rate, *i.e.*,

$$\mathsf{E}[\mathsf{E}[X \mid Y]] = \mathsf{E}[X].$$

The average tax in euros is

$$\mathsf{E}[XY] = \mathsf{E}[Y\,\mathsf{E}[X \mid Y]].$$

□

Definition 3.32: A sequence of random variables $X_0, X_1, \ldots$ is a martingale if for all t

$$\mathsf{E}[X_t \mid X_0, \ldots, X_{t-1}] = X_{t-1}.$$

A sequence of random variables $Y_1, Y_2, \ldots$ is a martingale difference sequence if for all t

$$\mathsf{E}[Y_t \mid Y_1, \ldots, Y_{t-1}] = 0.$$

□

If (X_t) is a martingale then $(X_t - X_{t-1})$ is a martingale difference sequence, and vice versa.

We also see that $\mathsf{E}[X_t] = X_0$ for all t.

Martingale differences define fair bets in the sense that whatever has happened in the past, the expected gain is always 0. However, the past may affect other properties of the bet.

Theorem 3.33 (Azuma's Inequality): Let (X_t) be a martingale sequence such that always

$$|X_t - X_{t-1}| \leq c_t$$

for some constants c_t. Then for any t and any $\alpha > 0$ we have

$$\Pr(|X_t - X_0| \geq \alpha) \leq 2 \exp\left(-\frac{\alpha^2}{2\sum_{i=1}^{t} c_i^2}\right).$$

□

Example 3.34: Consider the basic coin flipping game, with a restriction that $0 \leq b_t \leq 1$ but no other restrictions on b_t. For example, the player might set a target wealth C and let $b_t = 0$ if $X_{t-1} \geq C$. By taking $c_i = 1$ and $\alpha = t\varepsilon$, we get

$$\Pr(|X_t - X_0| \geq \varepsilon t) \leq 2 \exp\left(-\frac{t\varepsilon^2}{2}\right).$$

□

Our definition is not yet quite general enough.

Consider a variant of the betting game where instead of flipping a coin, we throw a 6-sided die, and the player wins if the result is odd (1, 3 or 5). If the die is symmetrical, the result is of course the same as for flipping a symmetrical coin. However, our definition does not capture all possibilities of unfairness in the game.

Suppose for example that we have a very strange die, such that after "2" it always produces a "1", and after "4" it always produces a "6". The game still satisfies our requirement

$$E[X_t \mid X_0, \ldots, X_{t-1}] = X_{t-1}.$$

However, if in addition to X_{t-1} the player knows the actual outcomes of the die, he can make an expected positive profit ($b_t = 1$ after "2" but $b_t = 0$ after "4").

Thus, for fairness we need

$$E[X_t \mid Z_0, \ldots, Z_{t-1}] = X_{t-1}$$

for any possible "side information" (Z_i).

Intuitively, we want to define (X_i) as martingale if

$$E[X_t \mid Z_0, \ldots, Z_{t-1}] = X_{t-1}$$

holds for all (Z_t) where the value of Z_t depends only on things that happen "before" X_{t+1} is chosen.

To make this more precise, recall that a probability space actually consists of a triple $(\Omega, \mathcal{F}, P)$, where $\mathcal{F} \subseteq \mathcal{P}(\Omega)$ is a σ-algebra and the probability measure P is a mapping from $\mathcal{F}$ to $[0, 1]$. The sets in $\mathcal{F}$ are called measurable, and the probability $P(A)$ is defined iff $A \in \mathcal{F}$. Further, a random variable is a measurable function from Ω to $\mathbf{R}$, *i.e.*, a function f such that $\{\, \omega \in \Omega \mid f(\omega) \leq c \,\}$ is measurable for all $c \in \mathbf{R}$.

Usually we prefer to take $\mathcal{F}$ large enough that we do not need to worry about measurability. For finite Ω we take $\mathcal{F} = \mathcal{P}(\Omega)$, and for $\Omega = \mathbf{R}^n$ we take $\mathcal{F}$ to consist of all Borel sets.

However, now it turns out to be convenient to artificially restrict what probabilities are well-defined.

Given a probability space $(\Omega, \mathcal{F}, P)$, we say that a sequence $\mathcal{F}_0, \ldots, \mathcal{F}_n$ is a filter if

1. each $\mathcal{F}_t$ is a σ-algebra,
2. $\mathcal{F}_0 = \{\emptyset, \Omega\}$,
3. $\mathcal{F}_{t-1} \subseteq \mathcal{F}_t$ for $1 \leq t \leq n$ and
4. $\mathcal{F}_n = \mathcal{F}$.

A function $f: \Omega \to \mathbf{R}$ is $\mathcal{F}_t$-measurable if $\{\omega \in \Omega \mid f(\omega) \leq c\} \in \mathcal{F}_t$ for all $c \in \mathbf{R}$.

Example 3.35: To describe a sequence of n throws of a 6-sided die, let $B = \{1, \ldots, 6\}$, $\Omega = B^n$ and $\mathcal{F} = \mathcal{P}(\Omega)$. Define a filter by

$$\mathcal{F}_t = \left\{ A \times B^{n-t} \mid A \subseteq B^t \right\}, \quad i = 1, \ldots, n.$$

In other words, $E \subseteq \Omega$ is in $\mathcal{F}_t$ if there is a set $A \subseteq B^t$ such that

$$(z_1, \ldots, z_n) \in E \Leftrightarrow (z_1, \ldots, z_t) \in A.$$

Then a function $f\colon \Omega \to \mathbf{R}$ is $\mathcal{F}_i$-measurable if

$$f(z_1, \ldots, z_n) = f(z_1, \ldots, z_t, z'_{t+1}, \ldots, z'_n)$$

for all $z_i, z'_i \in B$.

In particular, let Z_t be the outcome of the tth die throw (*i.e.*, $Z_t((z_1, \ldots, z_n)) = z_t$). Then Z_t is $\mathcal{F}_i$-measurable for $i \geq t$, but not for $i < t$. Similarly, any function that can be defined in terms of $Z_1, \ldots, Z_t$ is $\mathcal{F}_i$-measurable for $i \geq t$. □

Definition 3.36: Let $(\mathcal{F}_t)$ be a filter and (X_t) be a sequence of random variables such that X_t is $\mathcal{F}_t$ measurable for all t. The sequence (X_i) is a martingale with respect to $(\mathcal{F}_i)$, if for all random variable sequences (Z_i) such that Z_t is $\mathcal{F}_t$-measurable and X_i is a function of $Z_0, \ldots, Z_i$ we have

$$\mathsf{E}[X_t \mid Z_0, \ldots, Z_{t-1}] = X_{t-1}.$$

A sequence (X_t) is a martingale if there is some $(\mathcal{F}_t)$ such that (X_t) is a martingale with respect to $(\mathcal{F}_t)$. If (X_t) is a martingale, then $(X_t - X_{t-1})$ is a martingale difference sequence. □

Azuma's Inequality holds for this more general definition, too.

We recover our earlier definition by considering the smallest σ-algebras $\mathcal{F}_t$ such that X_t is $\mathcal{F}_t$-measurable. However, often it is useful to consider larger σ-algebras (*i.e.*, a finer partitioning of Ω).

Example 3.37: Let again $B = \{1, \ldots, 6\}$, $\Omega = B^n$ and

$$\mathcal{F}_t = \left\{ A \times B^{n-t} \mid A \subseteq B^t \right\}, \quad i = 1, \ldots, n.$$

Let $Z_t \in B$ be the result of the tth throw, and let $Q_t = 1$ if Z_t is odd and $Q_t = -1$ if Z_t is even.

Given random variables $(B_1, \ldots, B_n)$, define $Y_t = B_t Q_t$. Then $X_t = X_0 + \sum_{i=1}^{t-1} Y_i$ is the wealth of the player placing bets B_t. The bets B_t are random variables, since we allow them to depend on the observed outcomes.

Consider first the uniform probability measure $P(\{\omega\}) = 1/6^n$ for all $\omega \in \Omega$, and assume that B_t is $\mathcal{F}_{t-1}$-measurable. Let $\mathcal{R}$ be any collection of $\mathcal{F}_{t-1}$-measurable random variables. Since Q_t is independent of $\mathcal{R}$, we see that Y_t is a martingale difference sequence with respect to $(\mathcal{F}_t)$:

$$\begin{aligned} \mathsf{E}[Y_t \mid \mathcal{R}] &= \mathsf{E}[B_t Q_t \mid \mathcal{R}] \\ &= \mathsf{E}[B_t \, \mathsf{E}[Q_t \mid B_t, \mathcal{R}] \mid \mathcal{R}] \\ &= \mathsf{E}[B_t \mid \mathcal{R}] \, \mathsf{E}[Q_t] \\ &= 0. \end{aligned}$$

On the other hand, $\mathsf{E}[Y_t \mid Z_t, B_t] = B_t Z_t \neq 0$. (*i.e.*, the random variable $\mathsf{E}[Y_t \mid Z_t, B_t]$ is not identically zero). This does not matter, since Z_t is not $\mathcal{F}_{t-1}$-measurable.

Consider now the strange die for which $Z_{t+1} = 1$ if $Z_t = 2$, and $Z_{t+1} = 6$ if $Z_t = 4$, but otherwise the probabilities are uniform. Let $B_t = 1$ if $Z_{t-1} = 2$, and $B_t = 0$ otherwise. Now (Y_t) is not a martingale difference sequence with respect to $(\mathcal{F}_t)$, since $\mathsf{E}[Y \mid Z_{t-1}]$ is not identically zero.

(We could define a coarser σ-algebra $\mathcal{F}'_t \subset \mathcal{F}_t$ by requiring that for any measurable set A and any $\boldsymbol{z}, \boldsymbol{z}'$ such that $z_i \bmod 2 = z'_i \bmod 2$, we have $(z_1, \ldots, z_n) \in A$ iff $(z'_1, \ldots, z'_n) \in A$. Then we would have $\mathsf{E}[Y_t \mid \mathcal{R}] = 0$ for any $\mathcal{F}'_{t-1}$-measurable $\mathcal{R}$, but the (B_t) we defined above would not be $\mathcal{F}'_{t-1}$-measurable. In other words, we can redefine our notion of martingale to take into account only whether we win or lose at each time, but then our betting strategies should not use any side information, either.) □

We are now ready for a probabilistic analysis of online learning algorithms.

Let P be a probability measure over $X \times Y$, and let
$S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times Y)^m$ be a sample drawn according to P^m.

Consider an online learning algorithm A that on sample S produces the sequence of hypotheses $h_1, \ldots, h_{m+1}$. Notice that h_t is a random element depending on $x_1, y_1, \ldots, x_{t-1}, y_{t-1}$.

Assume L is a bounded loss function with $L(y, y') \in [0, B]$ for all y, y'. Define

$$M = \frac{1}{m} \sum_{i=1}^{m} L(y_i, h_i(x_i))$$

and

$$\widehat{\varepsilon}_t = \frac{1}{m - t + 1} \sum_{i=t}^{m} L(y_i, h_t(x_i)).$$

Write $\varepsilon_t = \mathsf{E}_{(x,y)\sim P}[L(y, h_t(x)]$. Then $\widehat{\varepsilon}_t$ is an unbiased estimate of ε_t.

Consider first the randomised hypothesis $\tilde{h}$ produced by the randomised conversion: $\tilde{h}(x) = h_t(x)$ with probability $1/m$, for $t = 1, \ldots, m$. Then $\frac{1}{m}\sum_{t=1}^{m} \varepsilon_t$ is the expected loss of $\tilde{h}$.

Theorem 3.38: With probability at least $1 - \delta$ we have

$$\frac{1}{m}\sum_{t=1}^{m} \varepsilon_t \leq M + B\sqrt{\frac{2}{m}\ln\frac{2}{\delta}}.$$

Proof: Define

$$V_t = \varepsilon_t - L(y_t, h_t(x_t)).$$

Then

$$\frac{1}{m}\sum_{t=1}^{m} \varepsilon_t = M + \frac{1}{m}\sum_{t=1}^{m} V_t.$$

We bound $\sum_t V_t$ by using Azuma's Inequality.

The hypothesis h_t depends only on $x_1, y_1, \ldots, x_{t-1}, y_{t-1}$. Regardless of what these values are, and what h_t is, we have $\varepsilon_t = \mathsf{E}_{(x,y)\sim P}[L(y, h_t(x))]$. Therefore

$$\mathsf{E}[V_t \mid x_1, y_1, \ldots, x_{t-1}, y_{t-1}] = 0$$

and (V_t) is a martingale difference sequence. (Notice that (x_i, y_i), $i < t$, contain all the information that affects h_t, so conditioning with respect to some additional random variables would not change anything.)

Since $0 \le L(y, y') \le B$, we have $|V_t| \le B$.

By applying Azuma's Inequality with

$$\alpha = mB\sqrt{\frac{2}{m}\ln\frac{2}{\delta}}$$

we see that $\left|\frac{1}{m}\sum_{t=1}^{m} V_t\right| \le \alpha/m$ with probability at least $1 - \delta$. □

Assume now that Y is a convex set, and let $\bar{h}$ be the averaged hypothesis

$$\bar{h}(x) = \frac{1}{m}\sum_{i=1}^{m} h_i(x).$$

Corollary 3.39: If L is convex, then with probability at least $1-\delta$ we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[L(y,\bar{h}(x)] \leq M + B\sqrt{\frac{2}{m}\ln\frac{2}{\delta}}.$$

Proof: By Jensen's Inequality we have

$$\mathop{\mathsf{E}}_{(x,y)\sim P}\left[L\left(y,\frac{1}{m}\sum_{i=1}^{m} h_i(x)\right)\right] \leq \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathsf{E}}_{(x,y)\sim P}\left[L\left(y,h_i(x)\right)\right].$$

We then apply the previous Theorem. □

Example 3.40: With a slight modification of the proof of Theorem 2.42, we can show that the Marginalised Perceptron satisfies

$$\sum_{t=1}^{m} L_\rho(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t) \leq \sum_{t=1}^{m} L_\mu(\boldsymbol{u}, \boldsymbol{x}_t, y_t) + \frac{X^2}{4(\mu - \rho)}$$

where we assume $\|\boldsymbol{u}\|_2 = 1$ and $\|\boldsymbol{x}_t\|_2 \leq X$ for all t and $0 \leq \rho < \mu$. Consider for simplicity the separable case $L_\mu(\boldsymbol{u}, \boldsymbol{x}_t, y_t) = 0$, and choose $\rho = \mu/2$. We get

$$\sum_{t=1}^{m} L_{\mu/2}(\boldsymbol{w}_t, \boldsymbol{x}_t, y_t) \leq \frac{X^2}{2\mu}.$$

Since clearly $\mu \leq X$ and and $L_\rho(\boldsymbol{w}, \boldsymbol{x}, y) \leq \rho + \|\boldsymbol{w}\|_2 \|\boldsymbol{x}\|_2$, we can apply the previous Corollary with $B = 3X/2$ to see that

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[L_{\mu/2}(\overline{h}, x, y)] \leq \frac{X^2}{2m\mu} + \frac{3X}{2}\sqrt{\frac{2}{m}\ln\frac{2}{\delta}}$$

holds with probability $1 - \delta$.

Now, since $L_{0-1}(\boldsymbol{w}, \boldsymbol{x}, y) \leq L_\rho(\boldsymbol{w}, \boldsymbol{x}, y)/\rho$, we have with probability $1 - \delta$

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[L_{0-1}(\bar{h}, x, y)] \leq \frac{X^2}{m\mu^2} + \frac{3X}{\mu}\sqrt{\frac{2}{m}\ln\frac{2}{\delta}}.$$

Under similar assumption, empirical risk minimisation gives (via Theorem 3.27) the bound

$$\mathop{\mathsf{E}}_{(x,y)\sim P}[L_{0-1}(\hat{h}, \boldsymbol{x}, y)] \leq \frac{4X}{\mu\sqrt{m}} + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

which is (not surprisingly) somewhat better. □

Notice that in practice we should not use any theoretical bound for M since it is a quantity whose exact value we can simply observe.

If L is not convex, and we are not happy with a randomised hypothesis, we apply penalised risk minimisation. Let

$$p_t = B\sqrt{\frac{2}{m-t+1}\ln\frac{4m}{\delta}}$$

and $\widehat{h} = h_\tau$ where

$$\tau = \arg\min_t(\widehat{\varepsilon}_t + p_t).$$

Theorem 3.41: With probability at least $1-\delta$ we have

$$\varepsilon_\tau = \mathop{\mathsf{E}}_{(x,y)\sim P}[L(y,\widehat{h}(x))] \leq M + 6B\sqrt{\frac{1}{m}\ln\frac{4m}{\delta}}.$$

Proof: We first show that with probability at least $1 - \delta/2$,

$$\varepsilon_\tau \leq \min_t (\varepsilon_t + 2p_t).$$

Write $T = \arg\min_t(\varepsilon_t + 2p_t)$. By the Hoeffding bound (Corollary 3.22), for any given t we have

$$\Pr(|\varepsilon_t - \widehat{\varepsilon}_t| > p_t) \leq \frac{\delta}{2n}.$$

Thus, with probability at least $1 - \delta/2$ we have in particular

$$\widehat{\varepsilon}_\tau \geq \varepsilon_\tau - p_\tau \qquad \text{and} \qquad \widehat{\varepsilon}_T \leq \varepsilon_T + p_T.$$

By the choice of τ, we have

$$\widehat{\varepsilon}_\tau + p_\tau \leq \widehat{\varepsilon}_T + p_T$$

which gives the desired

$$\varepsilon_\tau \leq \varepsilon_T + 2p_T.$$

We conclude the proof by showing that, with probability at least $1 - \delta/2$, we have

$$\min_t(\varepsilon_t + 2p_t) \leq M + 6B\sqrt{\frac{1}{m}\ln\frac{4m}{\delta}}.$$

Assume

$$\frac{1}{m}\sum_{t=1}^{m}\varepsilon_t \leq M + B\sqrt{\frac{2}{m}\ln\frac{4}{\delta}}$$

which by Theorem 3.38 happens with probability at least $1 - \delta/2$. Then

$$\begin{aligned}
\min_t(\varepsilon_t + 2p_t) &\leq \frac{1}{m}\sum_{t=1}^{m}(\varepsilon_t + 2p_t) \\
&\leq M + B\sqrt{\frac{2}{m}\ln\frac{4}{\delta}} + \frac{2B}{m}\sum_{t=1}^{m}\sqrt{\frac{2}{m-t+1}\ln\frac{4m}{\delta}} \\
&\leq M + B\sqrt{\frac{2}{m}\ln\frac{4}{\delta}} + 4B\sqrt{\frac{1}{m}\ln\frac{4m}{\delta}}
\end{aligned}$$

where we used $\sum_{i=1}^{m} 1/\sqrt{i} \leq 2\sqrt{m}$. □

Statistical learning: summary

Test set bounds are the most practical way of getting accurate error estimates. Cross validation is typically recommended for model selection.

Training error bounds are getting much better than they used to be, but are still short of giving practically useful numerical values.

Vapnik-Chervonenkis dimension is the basic combinatorial parameter that can be used for capacity control of the hypothesis class.

Bounds based on Rademacher complexities are better in that they take into account the actual distribution of the data. (Currently there are attempts to also take into account the algorithm being used; uniform convergence is unnecessarily strong.)

Online loss bounds can be directly converted to statistical bounds.

Key tools: concentration for sums of independent random variables and for martingales.

4. Support vector machines

We have already seen the main ingredients of a popular learning technique called Support Vector Machine (SVM): kernels and margins. Now we see how they are put together in a statistical learning setting:

- What are the requirements for a valid kernel?
- How do we formulate a mathematical optimisation problem in terms of margins?
- How to deal with noise?

We restrict ourselves on binary classification, but the ideas can be applied to multi-class tasks, regression, dimensionality reduction, outlier detection etc.

4.1 Some ideological comments

As we have remarked, a kernel can be seen as a computational trick to implement a feature map. Let $\boldsymbol{\psi}\colon X \to \mathbf{R}^r$ be a feature map and k the corresponding kernel: $\boldsymbol{\psi}(x_1) \cdot \boldsymbol{\psi}(x_2) = k(x_1, x_2)$.

Suppose we want to run the Perceptron with feature map $\boldsymbol{\psi}$. Then we replace the examples x_i by $\boldsymbol{\psi}(x_i)$, and obtain a weight vector $\boldsymbol{w} \in \mathbf{R}^k$ with

$$\boldsymbol{w} = \sum_{i \in I} y_i \boldsymbol{\psi}(x_i)$$

for some $I \subseteq \{1, \ldots, m\}$. To classify a new instance $z \in X$, we need to compute

$$\boldsymbol{w} \cdot \boldsymbol{\psi}(z) = \sum_{i \in I} y_i k(z, x_i),$$

which can be done without storing an explicit representation of $\boldsymbol{w}$.

We interpret $k(z, x_i)$ as a measure for how much the class of x_i should affect the classification of z. In other words, the kernel is a measure of similarity.

The basic kernel paradigm for machine learning is thus the following:

1. Pick a learning algorithm that can be kernelised. This means that $h(\boldsymbol{z})$ can be expressed in terms of y_i, $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ and $\boldsymbol{x}_i \cdot \boldsymbol{z}$, $1 \leq i, j \leq m$.

2. Pick a suitable kernel k, where $k(x, z)$ seems like a good measure of similarity between x and z, in terms of the application domain.

3. Run the algorithm with $\boldsymbol{x}_i \cdot \boldsymbol{z}$ and $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ replaced by $k(x_i, z)$ and $k(x_i, x_j)$.

This modularises the problem: we "only" need to

- find general-purpose learning algorithms that kernelise, and then
- for any given application domain, find a suitable kernel.

Example 4.1: Let us kernelise the Marginalised Perceptron (p. 114). Write $f_t(z) = \boldsymbol{w}_t \cdot \boldsymbol{\psi}(z)$. The goal is to show that we can compute $f_t(z)$ without explicitly creating $\boldsymbol{w}_t$ or $\boldsymbol{\psi}(z)$.

Let us use $f(\cdot)$ to denote the function $x \mapsto f(x)$, with the obvious interpretation of $f(\cdot) + g(\cdot)$ etc.

We will maintain a kernel expansion

$$f_t(\cdot) = \sum_{i=1}^{t-1} \alpha_i k(\cdot, x_i).$$

In other words, we store (x_i) and (α_i) so we can compute $f_t(z) = \sum_{i=1}^{t-1} \alpha_i k(z, x_i)$.

The basic update $\boldsymbol{w}'_{t+1} = \boldsymbol{w}_t + \eta\tilde{\sigma}_t y_t \boldsymbol{x}_t$ becomes

$$f_{t+1}(\cdot) := f_t + \eta\tilde{\sigma}_t y_t k(\cdot, x_t),$$

in other words $\alpha_t := \eta\tilde{\sigma}_t y_t$.

For normalising, notice that

$$\|\boldsymbol{w}_t\|_2^2 = \left(\sum_{i=1}^{t} \alpha_i \boldsymbol{x}_i\right) \cdot \left(\sum_{i=1}^{t} \alpha_i \boldsymbol{x}_i\right) = \sum_{i,j=1}^{t} \alpha_i \alpha_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j.$$

To normalise the kernel expansion, we compute

$$Z = \sqrt{\sum_{i,j=1}^{t} \alpha_i \alpha_j k(x_i, x_j)}$$

and if $Z > 1$ we set

$$f_{t+1}(\cdot) := f_{t+1}(\cdot)/Z,$$

in other words $\alpha_i := \alpha_i/Z$ for $i = 1, \ldots, t$. □

Of course, there really is **no such thing** as a (good) general-purpose learning algorithm. In terms of statistical learning theory, the choices are

- very rich hypothesis class: can eventually learn (almost) anything but need lots of data.
- very restricted hypothesis class: need less data, but total failure if the class does not include any good hypotheses.

In terms of kernels, the trade-off is basically

- high-dimensional feature space: can learn anything, given *enough data*
- low-dimensional feature space: needs less data, but must have the right features (or kernel).

By choosing a low-dimensional feature space, the user in some sense encodes his prior beliefs about the problem.

Of course, besides the feature space, the actual features also matter.

Suppose $X = \mathbf{R}^n$, and let $(a_1, \ldots, a_n) \in \mathbf{R}^n$ be such that $a_i > 0$ for all i. Take $\mathbf{R}^n$ as the feature space, too, and let $\boldsymbol{\psi}(\boldsymbol{x}) = (a_1 x_1, \ldots, a_n x_n)$. This feature map emphasises the components x_i that correspond to a large a_i. The unit ball is mapped into an ellipsoid with half-axes $a_1, \ldots, a_n$. If only few of the a_i are large, and most are comparatively small, then the effective dimension of the feature space is small. In other words, the feature vectors tend to lie in a very flat ellipsoid.

This particular feature map is of course not very helpful, unless we has some idea of which components are relevant. This idea is more important in in analysing some kernels with a very high-dimensional feature space.

Let $U \in \mathbf{R}^{n \times n}$ be an orthonormal matrix: $U^\mathsf{T} U = \mathbf{I}$ where $^\mathsf{T}$ denotes transpose and $\mathbf{I}$ is the $n \times n$ identity matrix. In other words, the columns $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ are orthonormal: $\boldsymbol{u}_i \cdot \boldsymbol{u}_j = \delta_{ij}$. The linear mapping $\boldsymbol{x} \mapsto U\boldsymbol{x}$ is a rotation of $\mathbf{R}^n$ that maps the ith unit vector into $\boldsymbol{u}_i$.

For any $\boldsymbol{x}, \boldsymbol{z} \in \mathbf{R}^n$ we have

$$(U\boldsymbol{x}) \cdot (U\boldsymbol{z}) = (\boldsymbol{x}^\mathsf{T} U^\mathsf{T})(U\boldsymbol{z}) = \boldsymbol{x}^\mathsf{T}(U^\mathsf{T} U)\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{z}.$$

In other words, rotations leave dot products unchanged. This corresponds to the geometrical interpretation of dot products in terms of angles.

Any kernelisable algorithm relies entirely on dot products, and therefore is rotation invariant: replacing each data point $\boldsymbol{x}$ with $U\boldsymbol{x}$, where U is any fixed orthonormal matrix, leaves the predictions unchanged.

The Perceptron family of algorithms are rotation invariant, and kernelisable.

Winnow is **not** rotation invariant. This is a feature, not a bug. The algorithm has been designed so that other things being equal, it finds the target $(1, 0, \ldots, 0)$ easier to learn than $(1/\sqrt{n}, \ldots, 1/\sqrt{n})$.

If we interpret a rotation as a change of coordinate system, then Winnow indicates a preference for targets that are sparse in the original coordinate system. Since sparseness is not a rotation invariant concept, no kernelisable algorithm can have such a preference.

4.2 Properties of kernels

An inner product space over $\mathbf{R}$ a pair $(A, \langle \cdot, \cdot \rangle)$ where $\langle \cdot, \cdot \rangle : X \times X \to \mathbf{R}$ is a function such that

1. $\langle x, y \rangle = \langle y, x \rangle$ for all $x, y \in A$,

2. $\langle \alpha x + \beta y, z \rangle = \alpha \langle x, z \rangle + \beta \langle y, z \rangle$ for all $x, y, z \in A$, $\alpha, \beta \in \mathbf{R}$ and

3. $\langle x, x \rangle \geq 0$ for all $x \in A$.

The inner product is strict if $\langle x, x \rangle = 0$ holds only for $x = \mathbf{0}$. A strict dot product can be used to define a norm $\|x\| = \sqrt{\langle x, x \rangle}$.

The basic example of an inner product space over $\mathbf{R}$ is $(R^n, \cdot)$ where $\cdot$ is the usual dot product.

In our proof of the Perceptron Convergence Theorem etc. we can replace the assumption $x_i \in \mathbf{R}^n$ by $x_i \in A$ for any inner product space over the reals.

Therefore, all the results hold for the kernelised versions (with $\|\boldsymbol{x}\|_2^2$ replaced by $k(x, x)$ etc.) assuming there is some inner product space $(A, \langle \cdot, \cdot \rangle)$ and some feature map $\boldsymbol{\psi}\colon X \to A$.

To guarantee that certain optimisation problems will have a solution, we require the stronger condition.

An infinite sequence $(x_1, x_2, \ldots)$ in a norm space is a Cauchy sequence if

$$\lim_{n\to\infty} \sup_{m>n} \|x_m - x_n\| = 0.$$

The space A is complete, if for any Cauchy sequence (x_n) there is a point $x \in A$ such that $\lim_{n\to\infty} x_n = x$.

Example 4.2: Consider the set $\mathbf{Q}$ of rational numbers, with the norm defined by the absolute value.

Define $f(n) = \max\{\, m \mid m^2 < 2n^2 \,\}$. Then the sequence $(f(n)/n)$ is Cauchy, but does not have a limit in $\mathbf{Q}$. Similarly, the set $\{\, q \in \mathbf{Q} \mid q^2 < 2 \,\}$ does not have a supremum in $\mathbf{Q}$.

If we view $\mathbf{Q}$ as a subset of $\mathbf{R}$, then both the limit and the supremum exist: namely the irrational number $\sqrt{2}$.

These are examples of the fact that $\mathbf{R}$ is complete, while $\mathbf{Q}$ is not.

In fact, one method of defining $\mathbf{R}$ is to first define $\mathbf{Q}$ by algebraic constructions and then let $\mathbf{R}$ be the completion of $\mathbf{Q}$. Formally, let R' be the set of all Cauchy sequences in $\mathbf{Q}$. Define an equivalence relation $\sim$ in R' by $(x_n) \sim (y_n)$ iff $\lim_n(x_n - y_n) = 0$. Then $\mathbf{R}$ is the set of equivalence classes of $\sim$. □

If an inner product space over $\mathbf{R}$ is complete, it is a Hilbert space over $\mathbf{R}$.

Example 4.3: $(\mathbf{R}^n, \cdot)$ is a Hilbert space over $\mathbf{R}$. □

Example 4.4: Let A be the set of sequences $(x_1, x_2, \ldots)$ such that $\sum_{i=1}^{\infty} x_i^2 < \infty$, and define

$$\langle (x_i), (y_i) \rangle = \sum_{i=1}^{\infty} x_i y_i.$$

It is easy to show that $(A, \langle \cdot, \cdot \rangle)$ is a Hilbert space over $\mathbf{R}$. We denote it by $\mathbf{R}^\infty$. □

In any inner product space, and thus in any Hilbert space, we can easily show that the Cauchy-Schwarz inequality holds:

$$\langle x, y \rangle^2 \leq \|x\|^2 \|y\|^2 .$$

A metric space A is separable if there is a countable subset $B \subseteq A$ such that any $x \in A$ is a limit of some sequence (x_i) where $y_i \in B$. Such a set B is dense in A.

Since $\mathbf{Q}$ is dense in $\mathbf{R}$, we see that $\mathbf{R}^n$ and $\mathbf{R}^\infty$ are separable. It can be shown that any separable Hilbert space over $\mathbf{R}$ is isomorphic either to $\mathbf{R}^n$ for some n or to $\mathbf{R}^\infty$. Intuitively, we can consider a Hilbert space as a generalisation of $\mathbf{R}^n$ where we allow a countably infinite number of basis vectors $\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots)$, $i = 1, 2, \ldots$, where $x_{ij} = \delta_{ij}$. Besides this intuition, we will not make much use of separability.

All our Hilbert spaces are over $\mathbf{R}$, so we will stop mentioning that explicitly. However, it should be noted that when people talk about "the" Hilbert space, that means the counterpart of $\mathbf{R}^\infty$ with complex coefficients.

Given m data points $\boldsymbol{x}_i \in \mathbf{R}^n$, the $m \times m$ matrix G with $G_{ij} = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$ is called the Gram matrix. More generally, given $(x_i) \in X^m$ and a kernel k over X, we also use the term Gram matrix for the matrix G where $G_{ij} = k(x_i, x_j)$. This latter matrix is also sometimes called the kernel matrix.

The Gram matrix is the central data structure in the algorithms we will soon encounter. Since we are interested in kernelised algorithms, the Gram matrix (together with the correct classes (y_i)) encodes all the information about the data that our algorithm is going to use.

From a practical point of view, it makes a huge difference in computation time whether the Gram matrix fits into main memory. Otherwise we have to store it on disk, or re-compute the kernel values whenever they are needed (which can be quite a few times over some optimisation algorithm).

A matrix $A \in \mathbf{R}^{m \times m}$ is positive semi-definite if for all $\boldsymbol{z} \in \mathbf{R}^m$ we have $\boldsymbol{z}^\mathsf{T} A \boldsymbol{z} \geq 0$. The Gram matrix is always positive semi-definite:

$$\begin{aligned}
\boldsymbol{z}^\mathsf{T} G \boldsymbol{z} &= \sum_{i,j=1}^{m} z_i z_j k(x_i, x_j) \\
&= \sum_{i,j=1}^{m} z_i z_j \left\langle \boldsymbol{\psi}(x_i), \boldsymbol{\psi}(x_j) \right\rangle \\
&= \left\langle \sum_{i=1}^{m} z_i \boldsymbol{\psi}(x_i), \sum_{j=1}^{m} z_j \boldsymbol{\psi}(x_j) \right\rangle \\
&= \left\| \sum_{i=1}^{m} z_i \boldsymbol{\psi}(x_i) \right\|^2 \\
&\geq 0.
\end{aligned}$$

Let $A \in \mathbf{R}^{n \times n}$. If $A\boldsymbol{x} = \lambda \boldsymbol{x}$ for some $\boldsymbol{x} \in \mathbf{R}^n$ and $\lambda \in \mathbf{R}$, we say that $\boldsymbol{x}$ is an eigenvector of A, and λ is the corresponding eigenvalue. A scalar multiple of an eigenvector is also an eigenvector with the same eigenvalue. The eigenvalues of a matrix are the same as the roots of its characteristic polynomial $\det(A - \lambda \mathbf{I})$.

Assume now that A is symmetrical. Then A has exactly n eigenvalues, when each value is counted as many times as it appears as a root of the characterictic polynomial. We usually write the eigenvalues as

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

The eigenvectors of a symmetrical matrix corresponding to different eigenvalues are orthogonal. It is easy to see that we can compose a basis $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$ such that

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$$

and the basis is orthonormal (*i.e.*, $\boldsymbol{v}_i \cdot \boldsymbol{v}_j = \delta_{ij}$).

Given the orthonormal basis $(\boldsymbol{v}_i)$, let V be the matrix with $\boldsymbol{v}_i$ as its ith column. Then $VV^\mathsf{T} = V^\mathsf{T}V = \mathbf{I}$, and we have the eigen-decomposition

$$A = V\Lambda V^\mathsf{T}$$

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. A symmetrical matrix is positive semidefinite if and only if all its eigenvalues are non-negative.

Define a matrix norm by

$$\|A\| = \max\{\, \|A\boldsymbol{x}\|_2 \mid \|\boldsymbol{x}\|_2 = 1 \,\}.$$

(There are various other matrix norms, too.) Then

$$\|A\| = \max_{1 \le i \le n} |\lambda_i|.$$

The determinant of A is given by

$$\det(A) = \lambda_1 \cdot \cdots \cdot \lambda_n.$$

The trace $\mathrm{tr}(A) = \sum_{i=1}^n a_{ii}$ is given by

$$\mathrm{tr}(A) = \lambda_1 + \cdots + \lambda_n.$$

We say that $k: X \times X \to \mathbf{R}$ is a valid kernel if there is a Hilbert space $\mathcal{H}$ and a mapping $\boldsymbol{\psi}: X \to \mathcal{H}$ such that $\langle \boldsymbol{\psi}(x), \boldsymbol{\psi}(y) \rangle = k(x, y)$ for $x, y \in X$.

A function $k: X \times X \to \mathbf{R}$ is a finitely positive semi-definite if it is symmetrical ($k(x, y) = k(y, x)$) and has the property that for any m and $(x_1, \ldots, x_m) \in X^m$, the Gram matrix $(k(x_i, x_j))$ is positive semi-definite.

Theorem 4.5: Consider $k: X \times X \to \mathbf{R}$. Then k is a valid kernel, if and only if it is finitely positive semi-definite.

Proof: The "only-if" part follows from previous remarks. We prove the "if" part by constructing a suitable $\mathcal{H}$ and $\boldsymbol{\psi}$. The Hilbert space $\mathcal{H}$ will consist of certain functions $X \to \mathbf{R}$, with the usual scalar multiplication and addition:

$$(\alpha f + \beta g)(x) = \alpha f(x) + \beta g(x) \qquad \text{for } \alpha, \beta \in \mathbf{R},\ f, g \in \mathcal{H},\ x \in X.$$

First, let

$$\mathcal{H}' = \left\{ \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \mid n \in \mathbf{N}, \alpha_i \in \mathbf{R}, x_i \in X \right\}.$$

We next construct an inner product for $\mathcal{H}'$. Some additions will be needed to obtain $\mathcal{H}$ that is also complete.

Given $f, g \in \mathcal{H}'$ with

$$f = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \qquad \text{and} \qquad g = \sum_{j=1}^{m} \beta_j k(\cdot, x_j),$$

define

$$\langle f, g \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x_j).$$

This inner product is well-defined even though the expansions for f and g are not unique. For example, assume we also have

$$g = \sum_{j=1}^{\ell} \beta'_j k(\cdot, x'_j).$$

Then

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i \beta_j k(x_i, x_j) = \sum_{i=1}^{n} \alpha_i g(x_i) = \sum_{i=1}^{n}\sum_{j=1}^{\ell} \alpha_i \beta'_j k(x_i, x'_j).$$

By finite positive semi-definiteness, we have

$$\langle f, f \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \geq 0.$$

It is now easy to verify the rest of the properties required for $\mathcal{H}'$ to be an inner product space.

At this stage we can notice the important reproducing property. Given $x \in X$, consider $g = k(\cdot, x)$. For $f = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i)$ we get

$$\langle f, k(\cdot, x) \rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x) = f(x).$$

We are later going to define $\psi(x) = k(\cdot, x)$, so this can be written as

$$\langle f, \psi(x) \rangle = f(x).$$

In other words, the functions $f \colon X \to \mathbf{R}$ in $\mathcal{H}'$ correspond to linear functions in the feature space. Because of this property, the space $\mathcal{H}'$ is called the Reproducing Kernel Hilbert Space (RHKS) for kernel k. (Technically, the RHKS is the space $\mathcal{H}$ we next construct that is also complete.)

We now see that the inner product is strict: if $\|f\| = 0$, then for all x we have by Cauchy-Schwarz

$$f(x)^2 = \langle f, k(\cdot, x) \rangle^2 \leq \|f\|^2 \, k(x, x) = 0.$$

Consider now a Cauchy sequence $(f_1, f_2, \ldots)$ and fix a point $x \in X$.

Using the reproducing property, and then Cauchy-Schwarz, we get

$$|f_n(x) - f_m(x)|^2 = \langle f_n - f_m, k(\cdot, x)\rangle^2 \leq \|f_n - f_m\|^2 \, k(x, x).$$

Since the sequence (f_n) is Cauchy, so is the sequence $(f_n(x))$ for any $x \in X$. We can therefore define a function $\bar{f}$ with $\bar{f}(x) = \lim_{n\to\infty} f_n(x)$.

Now $\mathcal{H}$ consist of $\mathcal{H}'$ extended by $\bar{f}$ for all Cauchy sequences (f_n), with the operations generalised to limits in the obvious manner ($\langle \bar{f}, g\rangle = \lim_{n\to\infty} \langle f_n, g\rangle$ etc.). It is easy to verify that $\mathcal{H}$ is indeed a Hilbert space.

Finally, we define $\boldsymbol{\psi}(x) = k(\cdot, x)$, which gives

$$\langle \boldsymbol{\psi}(x), \boldsymbol{\psi}(y)\rangle = \langle k(\cdot, x), k(\cdot, y)\rangle = k(x, y)$$

as desired. □

We give (without proof) an alternative theorem that shows the existence of the RKHS and gives a little more intuition into its elements.

First, given a compact (*i.e.*, closed and bounded) set $X \subseteq \mathbf{R}^n$, let

$$L_2(X) = \left\{ f \colon X \to \mathbf{R} \mid \int_X f(x)^2 \, dx < \infty \right\}$$

with the inner product

$$\langle f, g \rangle = \int_X f(x) g(x) \, dx.$$

Theorem 4.6 (Mercer): Assume $k \colon X \times X \to \mathbf{R}$ is symmetrical and continuous and satisfies

$$\int_{X \times X} k(x, y) f(x) f(y) \, dx \, dy \geq 0$$

for all $f \in L_2(X)$. Let $T \colon L_2(X) \to L_2(X)$ be the operator

$$Tf = \int_X k(\cdot, x) f(x) \, dx.$$

Then there is a sequence of functions $\psi_i \in L_2(X)$ and scalars $\lambda_i \in \mathbf{R}$, $i = 1, 2, \ldots$, such that

1. $\langle \psi_i, \psi_j \rangle = \delta_{ij}$,
2. $T\psi_i = \lambda_i \psi_i$,
3. $\lambda_i \geq 0$ for all i and $\sum_i \lambda_i < \infty$ and
4. for $x, y \in X$ we have $k(x, y) = \sum_{i=1}^{\infty} \lambda_i \psi_i(x) \psi_i(y)$. $\square$

The intuitive interpretation is that the eigenvectors ψ_i form an orthonormal basis for the Hilbert space. In the inner product, the component corresponding to ψ_i is weighted by the eigenvalue λ_i.

Notice however that not all kernels are defined on a compact set X.

Theorem 4.7: Let k_1 and k_2 be valid kernels on $X \times X$. Then the following functions K are valid kernels:

1. $k(x, y) = k_1(x, y) + k_2(x, y)$,

2. $k(x, y) = ak_1(x, y)$ for any $a > 0$ and

3. $k(x, y) = k_1(x, y)k_2(x, y)$.

Proof: By Theorem 4.5, the proof can be reduced to considering positive semi-definite matrices. Parts (1) and (2) are left as an exercise.

For Part (3), let K' and K'' be symmetrical positive semi-definite $m \times m$ matrices. We need to show that K given by $K_{ij} = K'_{ij}K''_{ij}$ is positive semi-definite.

Given matrices $A \in \mathbf{R}^{n \times m}$ and $B \in \mathbf{R}^{k \times \ell}$, let their tensor product $A \otimes B$ be the $(nk) \times (m\ell)$ matrix obtained by replacing in A each entry a_{ij} by the matrix $a_{ij}B$.

Both K' and K'' have n non-negative eigenvalues. The eigenvalues of $K' \otimes K''$ are the n^2 pairwise products of these. Therefore, $K' \otimes K''$ is positive semi-definite.

The matrix K corresponding to k is obtained from $K' \otimes K''$ by picking n rows and columns $(i-1)n + i$, $i = 1, \ldots, n$. If K had a negative eigenvalue, we would obtain a negative eigenvalue for $K' \otimes K''$ by padding the eigenvector of K with zeroes.

Therefore, K cannot have negative eigenvalues. $\square$

Corollary 4.8: Let k_1 be a kernel. Then the following k are valid kernels:

1. $k(x, y) = p(k_1(x, y))$ where p is a polynomial with positive coefficients,
2. $k(x, y) = \exp(k_1(x, y))$ and
3. $k(x, y) = \exp(-\|x - y\|^2 / (2\sigma^2))$ where $\|z\|^2 = \langle z, z \rangle$.

Proof: Left as an exercise. □

4.3 Optimising margins

Assume now that we have chosen a kernel k on X, and thus implicitly a feature map $\psi\colon X \to H$ for some Hilbert space $(H, \langle \cdot, \cdot \rangle)$.

Suppose a learning algorithm has returned a classifier $f\colon X \to \{-1, 1\}$ where $f(x) = \operatorname{sign}(\langle \boldsymbol{w}, \psi(x) \rangle)$ for some $\boldsymbol{w} \in H$ with $\|\boldsymbol{w}\| \leq B$. From Theorem 3.27, we have the error bound

$$\frac{1}{\mu m} \sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \psi(x_i), y_i) + \frac{4B}{\mu m} \sqrt{\sum_{i=1}^{m} \|\psi(x_i)\|^2} + 3\sqrt{\frac{1}{2m} \ln \frac{4}{\delta}}.$$

To minimise this bound, we would like to have

1. μ as large as possible, and
2. $\sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \psi(x_i), y_i)$ as small as possible.

These subgoals are, of course, in conflict (recall $L_\mu(\boldsymbol{w}, \boldsymbol{x}, y) = \max\{0, \mu - y\boldsymbol{w} \cdot \boldsymbol{x}\}$).

We consider three approaches to this conflict. The simplest one is

hard-margin SVM: we choose largest μ such that some $\boldsymbol{w} \in H$ separates the data with margin μ, *i.e.*, $L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i) = 0$ for all i.

The hard margin case is a good introduction to the methodology, but in practice it is very **sensitive to noise** and outliers. Even noisy data can be made linearly separable by using a sufficiently "rich" kernel, but this is still overfitting. In terms of our bound, the norms $\|\boldsymbol{w}\|$ and $\|\boldsymbol{\psi}(x_i)\|$ would become very large, or the margin very small.

Thus, in practice we usually prefer some kind of soft margin SVM, of which we present

1-norm soft margin SVM: minimise $-\mu + C \sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i)$ for some trade-off parameter $C > 0$ and

ν-SVM: same as 1-norm soft margin SVM, except that the trade-off is parameterised by using $0 \leq \nu \leq 1$.

The parameters C and ν need to be tuned empirically, but the ν formalism has the advantage that ν turns out to have an intuitive meaning.

Hard margin SVM

We consider functions $f\colon X \to \mathbf{R}$ of the form

$$f(x) = \langle \boldsymbol{w}, \boldsymbol{\psi}(x) \rangle - b$$

for some $\boldsymbol{w} \in H$ and $b \in \mathbf{R}$. Notice that we now explicitly include the bias b. As usual, we get a classifier as $g(x) = \mathrm{sign}(f(x))$.

Recall that

$$\frac{|f(x)|}{\|\boldsymbol{w}\|}$$

is the geometrical distance between the point $\boldsymbol{\psi}(x)$ and the hyperplane $f(x) = 0$ (in Hilbert space H). Further, $yf(x) > 0$ means that $g(x) = y$. Given a margin parameter μ, we define slack variables ξ_i, $i = 1, \ldots, m$, by

$$\xi_i = \max\{\, 0, \mu - y_i f(x_i) \,\}.$$

Thus, if $\|\boldsymbol{w}\| = 1$, the slack variable ξ_i measures the failure of (x_i, y_i) to be at least a distance μ on the correct side of the decision boundary.

In the hard margin SVM, we set the hard constraint that all the slack variables must be zero. The task is to achieve this with the geometric margin as large as possible. For notational convenience, we formalise this as a minimisation problem.

Optimisation 4.9: Variables $\boldsymbol{w} \in H$, $b \in \mathbf{R}$, $\mu \in \mathbf{R}$

$$\begin{array}{ll} \textbf{minimise} & -\mu \\ \textbf{subject to} & \mu \geq 0 \\ & y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) - \mu \geq 0 \text{ for } i = 1, \ldots, m \\ & \|\boldsymbol{w}\|^2 \leq 1. \end{array}$$

A variable assignment is feasible if it satisfies all the constraints. In our case, if $(\boldsymbol{w}, b, \mu)$ is feasible, then $(r\boldsymbol{w}, rb, r\mu)$ is feasible for all $0 < r \leq 1/\|\boldsymbol{w}\|$. This means that

1. If feasible solutions exist, the optimum occurs for $\|\boldsymbol{w}\| = 1$.

2. We would get the same solutions, up to rescaling, by minimising $\|\boldsymbol{w}\|^2$ subject to constraints $y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) \geq 1$. This formalisation is a quadratic optimisation problem.

We now recall some basic facts from optimisation theory. Consider the following problem where f, p_i and q_i are functions $\mathbf{R}^n \to \mathbf{R}$.

Optimisation 4.10: Variables $\boldsymbol{x} \in \mathbf{R}^n$

$$\begin{array}{ll} \textbf{minimise} & f(\boldsymbol{x}) \\ \textbf{subject to} & p_i(\boldsymbol{x}) \leq 0 \text{ for } i = 1, \ldots, m \\ & q_j(\boldsymbol{x}) = 0 \text{ for } j = 1, \ldots, l. \end{array}$$

The problem is convex if f is convex, p_i is convex for $i = 1, \ldots, m$, and q_j is affine for $j = 1, \ldots, l$ (*i.e.*, $q_j(\boldsymbol{x}) = \boldsymbol{u}_j \cdot \boldsymbol{x} + b_j$ for some $\boldsymbol{u}_j \in \mathbf{R}^n$, $b_j \in \mathbf{R}$).

Notice that Optimisation 4.9 is convex (with some trivial rewriting of the constraints).

The feasible set is the set of $\boldsymbol{x}$ such that the constraints $p_i(\boldsymbol{x}) \leq 0$ and $q_j(\boldsymbol{x}) = 0$ are satisfied. For a convex problem, the feasible set is convex.

For illustration, assume we have just one inequality constraint:

minimise $f(\boldsymbol{x})$
over $\boldsymbol{x} \in \mathbf{R}^n$
subject to $p(\boldsymbol{x}) \leq 0$.

Consider $f(\boldsymbol{x})$ as the potential (say, elevation) at point $\boldsymbol{x}$, so the negative gradient $-\nabla f(\boldsymbol{x})$ is the force acting on a particle at point $\boldsymbol{x}$.

Further, along the surface $p(\boldsymbol{x}) = 0$ there is a fence. For a particle at point $\boldsymbol{x}$ with $p(\boldsymbol{x}) = 0$, the fence exerts a contraint force that is just sufficient to keep the particle on the right side. The constraint force is perpendicular to the fence and points invards, so it is $-\lambda \nabla p(\boldsymbol{x})$ for some $\lambda > 0$.

Since we assume an infinitely strong fence, we have no prior knowledge of how large λ might be. However, if the particle does not touch the fence, the constraint force is zero: if $p(\boldsymbol{x}) \neq 0$, then $\lambda = 0$, which we write as $\lambda p(\boldsymbol{x}) = 0$.

The solution to the optimisation problem is the point $\boldsymbol{x}^*$ at which the particle is at balance, *i.e.*, the net force acting on it is zero. Summarising the conditions, we get

$$\begin{aligned} \nabla f(\boldsymbol{x}^*) + \lambda \nabla p(\boldsymbol{x}^*) &= 0 \\ \lambda &\geq 0 \\ \lambda p(\boldsymbol{x}^*) &= 0. \end{aligned}$$

If there are several inequality constraints, then each fence $p_i(\boldsymbol{x})$ can exert its own constraint force $-\lambda_i \nabla p_i(\boldsymbol{x})$, again subject to $\lambda_i \geq 0$ and $\lambda_i p_i(\boldsymbol{x}) = 0$.

For equality constraints, the particle is prevented from leaving the surface $q_j(\boldsymbol{x}) = 0$. The constraint force is still perpendicular to the surface, but can be to either direction, so it is given by $\beta_j \nabla q_j(\boldsymbol{x})$ where $\beta_j \in \mathbf{R}$.

By combining all the above, we get the Karush-Kuhn-Tucker (KKT) conditions for optimisation problem 4.10:

$$\begin{aligned}
\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i \nabla p_i(\boldsymbol{x}^*) + \sum_{j=1}^{l} \beta_j \nabla q_j(\boldsymbol{x}^*) &= 0 \\
p_i(\boldsymbol{x}^*) &\leq 0 \qquad \text{for } i = 1, \ldots, m \\
q_j(\boldsymbol{x}^*) &= 0 \qquad \text{for } j = 1, \ldots, l \\
\lambda_i &\geq 0 \qquad \text{for } i = 1, \ldots, m \\
\lambda_i p_i(\boldsymbol{x}^*) &= 0 \qquad \text{for } i = 1, \ldots, m.
\end{aligned}$$

If the problem is convex (and satisfies some regularity conditions), there are no local minima, and the KKT conditions can be proved to be necessary and sufficient for $\boldsymbol{x}^*$ to be the solution of Optimisation 4.10.

If the problem is not convex, the situation is more complicated, but often the KKT conditions are still necessary.

A further important notion is Lagrange duality (or simply duality).

Given Optimisation 4.10, we define the Lagrangian $L\colon \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R}^l \to \mathbf{R}$ by

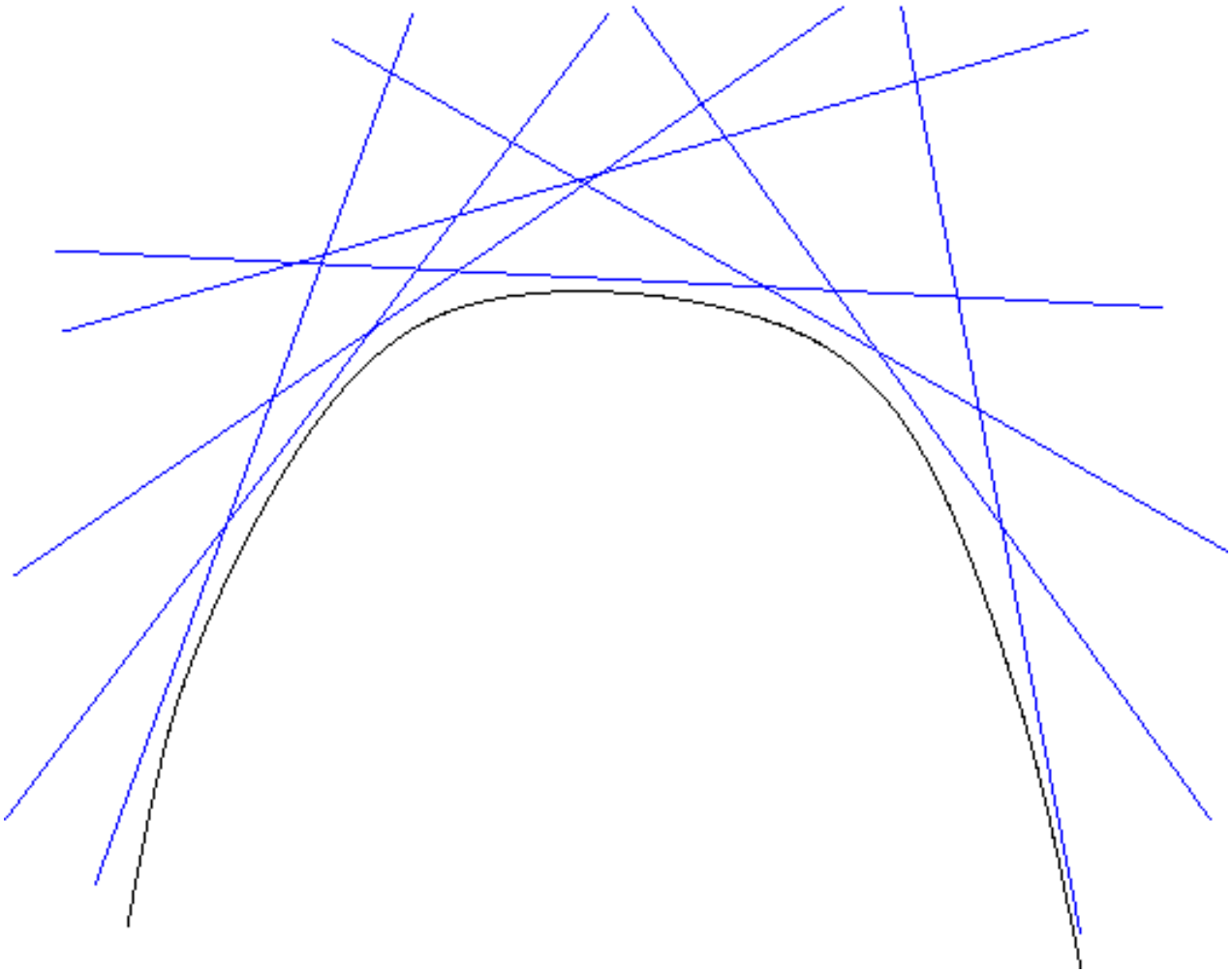
$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_j q_j(\boldsymbol{x}).$$

(Compare with KKT conditions.) The new variable λ_i is called the Lagrange multiplier for the constraint $p_i(\boldsymbol{x}) \leq 0$, and similarly for β_j.

The dual function is $g\colon \mathbf{R}^m \times \mathbf{R}^l \to \mathbf{R} \cup \{-\infty\}$ where

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) = \inf_{\boldsymbol{x} \in \mathbf{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}).$$

Since g is a pointwise infimum of affine functions, it is concave even if the original (primal) problem is not convex.



The infimum of affine functions is concave.

Let $\boldsymbol{x}$ be feasible (*i.e.*, satisfy the constraints), $\lambda_i \geq 0$ for all i, and $\beta_j \in \mathbf{R}$ arbitrary. Since $p_i(\boldsymbol{x}) \leq 0$, we have

$$\sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_i q_i(\boldsymbol{x}) \leq 0.$$

Therefore

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_i q_i(\boldsymbol{x}) \leq f(\boldsymbol{x})$$

which implies

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) = \inf_{\boldsymbol{x}' \in \mathbf{R}^n} L(\boldsymbol{x}', \boldsymbol{\lambda}, \boldsymbol{\beta}) \leq f(\boldsymbol{x}).$$

We can now define the dual problem of the primal problem 4.10 as the following.

Optimisation 4.11: Variables: $\boldsymbol{\lambda} \in \mathbf{R}^m$, $\boldsymbol{\beta} \in \mathbf{R}^l$

$$\begin{array}{ll} \textbf{maximise} & g(\boldsymbol{\lambda}, \boldsymbol{\beta}) \\ \textbf{subject to} & \lambda_i \geq 0 \text{ for } i = 1, \ldots, m. \end{array}$$

We call $(\boldsymbol{\lambda}, \boldsymbol{\beta}) \in \mathbf{R}^m \times \mathbf{R}^l$ dual feasible if $\lambda_i \geq 0$ for all i.

Let $\boldsymbol{x}^*$ be a solution to the primal 4.10 and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ to the dual 4.11. (They need not be unique.)

Given a feasible $\boldsymbol{x}$ and dual feasible $(\boldsymbol{\lambda}, \boldsymbol{\beta})$, the remark on previous page gives

$$g(\boldsymbol{\lambda}, \boldsymbol{\beta}) \leq g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \leq f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}).$$

Many optimisation algorithms produce a sequence of dual-primal feasible $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$, from which we thus get immediately an estimate of how far we are from the optimal values.

Given primal-dual feasible $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$, the quantity $f(\boldsymbol{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\beta})$ is called the duality gap.

The quantity $f(\boldsymbol{x}^*) - g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ is the optimal duality gap. The fact that it is always non-negative, as we just saw, is called weak duality. If it is actually zero, *i.e.*, $f(\boldsymbol{x}^*) = g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$, we say that strong duality holds.

Convexity with some additional conditions is a sufficient (but not necessary) condition for strong duality. Requiring the existence of "strictly feasible" solutions is one way of setting the extra conditions.

Theorem 4.12 (Slater): If Optimisation 4.10 is convex and there is some $\boldsymbol{x} \in \mathbf{R}^n$ such that $p_i(\boldsymbol{x}) < 0$ for $1 \leq i \leq m$ and $q_j(\boldsymbol{x}) = 0$ for $1 \leq j \leq l$, then strong duality holds.

Proof can be found in optimisation text books. □

Assume that $\boldsymbol{x}^*$ is primal and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ dual optimal, and strong duality holds. Then

$$\begin{aligned} f(\boldsymbol{x}^*) &= g(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \\ &= \inf_{\boldsymbol{x} \in \mathbf{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*) \\ &\leq f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* p_i(\boldsymbol{x}^*) + \sum_{j=i}^{l} \beta_j^* q_j(\boldsymbol{x}^*) \\ &\leq f(\boldsymbol{x}^*) \end{aligned}$$

where the last step follows from primal and dual feasibility. Thus the inequalities hold as equalities, and in particular $\sum_{i=1}^{m} \lambda_i^* p_i(\boldsymbol{x}^*) = 0$. Since each term in the sum in non-negative, we must have

$$\lambda_i^* p_i(\boldsymbol{x}^*) = 0 \qquad \text{for } 1 \leq i \leq m.$$

We already gave an intuitive motivation for this condition known as complementary slackness. It means that at the optimum, the Lagrange coefficient can be non-zero only for constraints that are active.

Another important observation is that $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \mathbf{R}^n} L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$.

Recall that the KKT conditions for $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$ are

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i \nabla p_i(\boldsymbol{x}) + \sum_{j=1}^{l} \beta_j \nabla q_j(\boldsymbol{x}) = 0 \tag{1}$$

$$\begin{aligned} p_i(\boldsymbol{x}) &\leq 0 \quad \text{for } i = 1, \ldots, m && (2)\\ q_j(\boldsymbol{x}) &= 0 \quad \text{for } j = 1, \ldots, l && (3)\\ \lambda_i &\geq 0 \quad \text{for } i = 1, \ldots, m && (4)\\ \lambda_i p_i(\boldsymbol{x}) &= 0 \quad \text{for } i = 1, \ldots, m. && (5) \end{aligned}$$

Theorem 4.13 (Karush-Kuhn-Tucker): Let $\boldsymbol{x}^*$ be primal and $(\boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ dual optimal, and assume that strong duality holds. Assume futher that f and p_i are differentiable. Then $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ satisfies the KKT conditions. Further, if the primal problem is convex, then any $(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\beta})$ that satisfies the KKT conditions is optimal.

Proof: Let $(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ be as assumed. Conditions (2)–(4) are the feasibility constraints, and (5) is the complementary slackness we just verified.

Finally, since $\boldsymbol{x} = \boldsymbol{x}^*$ minimises $L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$, the gradient of $L(\boldsymbol{x}, \boldsymbol{\lambda}^*, \boldsymbol{\beta}^*)$ w.r.t. $\boldsymbol{x}$ must be zero at $\boldsymbol{x} = \boldsymbol{x}^*$.

Assume now that the problem is convex and $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}})$ satisfy the KKT conditions. In particular, this implies feasibility. Since $\tilde{\lambda}_i \geq 0$, we see that $L(\boldsymbol{x}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}})$ is convex in $\boldsymbol{x}$ and therefore attains its minimum when the gradient is zero. Using (1), this happens for $\boldsymbol{x} = \tilde{\boldsymbol{x}}$. We conclude

$$\begin{aligned} g(\tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}}) &= L(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{\lambda}}, \tilde{\boldsymbol{\beta}}) \\ &= f(\tilde{\boldsymbol{x}}) + \sum_{i=1}^{m} \tilde{\lambda}_i p_i(\tilde{\boldsymbol{x}}) + \sum_{j=1}^{l} \tilde{\beta}_j q_j(\tilde{\boldsymbol{x}}) \\ &= f(\tilde{\boldsymbol{x}}) \end{aligned}$$

where the last step uses (3) and (5). Hence, we have zero duality gap, which implies optimality. □

We are now ready to tackle the hard margin SVM optimisation 4.9. Clearly the problem is convex, and if a positive margin is possible then Slater's condition is satisfied.

It should be noticed that we are (conceptually) optimising over $\boldsymbol{w} \in H$ for some Hilbert space, and not $\boldsymbol{w} \in \mathbf{R}^n$. However, it is easy to see that all the required properties still hold.

First, we set up the Lagrangian

$$L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda) = -\mu - \sum_{i=1}^{m} \alpha_i (y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) - \mu) + \lambda(\|\boldsymbol{w}\|^2 - 1)$$

where the constraints for the Lagrange coefficients are $\alpha_i \geq 0$ and $\lambda \geq 0$.

We have omitted the constraint $\mu \geq 0$ since we can just check afterwards whether it holds. If not, then the problem is not separable.

We get the dual by optimising away the primal variables. The derivatives are

$$\begin{aligned}
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial \boldsymbol{w}} &= -\sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i) + 2\lambda \boldsymbol{w} \\
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial b} &= \sum_{i=1}^{m} \alpha_i y_i \\
\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)}{\partial \mu} &= -1 + \sum_{i=1}^{m} \alpha_i .
\end{aligned}$$

Setting them all to zero gives us

$$\begin{aligned}
\boldsymbol{w} &= \frac{1}{2\lambda} \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i) \\
\sum_{i=1}^{n} \alpha_i y_i &= 0 \\
\sum_{i=1}^{n} \alpha_i &= 1.
\end{aligned}$$

We now simplify the dual

$$g(\boldsymbol{\alpha}, \lambda) = \inf_{\boldsymbol{w}, b, \mu} L(\boldsymbol{w}, b, \mu, \boldsymbol{\alpha}, \lambda)$$

assuming $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $\sum_{i=1}^{m} \alpha_i = 1$. (If these conditions do not hold, then $g(\boldsymbol{\alpha}, \lambda) = -\infty$.)

Plugging also the condition for $\boldsymbol{w}$ into the Lagrangian gives us

$$\begin{aligned} g(\boldsymbol{\alpha}, \lambda) &= -\sum_{i=1}^{m} \alpha_i y_i \langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle + \lambda \|\boldsymbol{w}\|^2 - \lambda \\ &= \left(-\frac{1}{2\lambda} + \frac{1}{4\lambda} \right) \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) - \lambda \\ &= \frac{1}{4\lambda} W(\boldsymbol{\alpha}) - \lambda \end{aligned}$$

where

$$W(\boldsymbol{\alpha}) = -\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = -\|2\lambda \boldsymbol{w}\|^2 .$$

Maximising w.r.t. lambda gives

$$\max_{\lambda \geq 0} g(\boldsymbol{\alpha}, \lambda) = -(-W(\boldsymbol{\alpha}))^{1/2}$$

with the maximum attained at $\lambda = \frac{1}{2}(-W(\boldsymbol{\alpha}))^{1/2}$. Hence, the dual solution $(\boldsymbol{\alpha}^*, \lambda^*)$ is obtained as

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} \left(- \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \right) = \arg\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}),$$

where the optimisation is subject to $\sum_{i=1}^{m} \alpha_i y_i = 0$ and $\sum_{i=1}^{m} \alpha_i = 1$, and

$$\lambda^* = \frac{1}{2} \left(\sum_{i,j=1}^{m} \alpha_i^* \alpha_j^* y_i y_j k(x_i, x_j) \right)^{1/2}.$$

We still need to recover the primal variables. Since $-\mu$ is the objective function, strong duality implies

$$\mu^* = -g(\boldsymbol{\alpha}^*, \lambda^*) = (-W(\boldsymbol{\alpha}^*))^{1/2} = 2\lambda^*.$$

We already noticed

$$\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i).$$

To solve b^*, we notice that by complementary slackness we have $y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle - b^*) = \mu^*$ for any i such that $\alpha_i^* \neq 0$.

For simplicity, we choose as the final output $(\boldsymbol{w}, b)$ of the algorithm a scaled version $(2\lambda^* \boldsymbol{w}^*, 2\lambda^* b^*)$ of the solution to the original optimisation.

We summarise the preceding observations.

Algorithm 4.14 (Hard Margin SVM): Input: sample $((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$

1. Obtain $\boldsymbol{\alpha}^* \in \mathbf{R}^m$ as maximiser of

$$W(\boldsymbol{\alpha}) = \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

   subject to $\sum_{i=1}^m \alpha_i = 1$, $\sum_{i=1}^m \alpha_i y_i = 0$ and $\alpha_i \geq 0$ for all i.

2. Let $\mu^* = (-W(\boldsymbol{\alpha}^*))^{1/2}$. Choose any i such that $\alpha_i^* \neq 0$ and set

$$b = \sum_{j=1}^{m} \alpha_j^* y_j k(x_i, x_j) - y_i (\mu^*)^2.$$

3. Output the classifier $\operatorname{sign}(f(\cdot))$ where

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i^* y_i k(\cdot, x_i) - b.$$

The support vectors are those feature vectors $\boldsymbol{\psi}(x_i)$ for which $\alpha_i^* > 0$. By the KKT conditions, the support vectors satisfy

$$y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle - b^*) = \mu^*.$$

In other words, the optimal classifier has margin exactly μ^* on the support vectors.

(To understand the name of the method, visualise a strut of lenght μ^* from each support vector perpendicularly to the optimal separating hyperplane.)

Typically only a small proportion of the examples end up as support vectors. This is important from a computational point of view. Notice that to represent the hypothesis

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i^* y_i k(\cdot, x_i)$$

we need to store only those x_i, y_i and α_i^* for which $\alpha_i^* > 0$.

It would also be possible to derive generalisation error bounds based on having a small number of support vectors. We shall instead adapt the margin-based bound of Theorem 3.27.

If we omit the (ultimately redundant) constraint $\mu \geq 0$, then for any $\hat{\boldsymbol{w}}$ with $\|\hat{\boldsymbol{w}}\| = 1$ there is $\hat{b} \in \mathbf{R}$ and $\hat{\mu} \in \mathbf{R}$ such that $(\hat{\boldsymbol{w}}, \hat{b}, \hat{\mu})$ is feasible for the primal. The largest possible such $\hat{\mu}$ is given by

$$\hat{\mu} = \frac{1}{2}\left(\min\{\,\langle\,\hat{\boldsymbol{w}}, \boldsymbol{\psi}(x_i)\rangle \mid y_i = 1\,\} - \max\{\,\langle\,\hat{\boldsymbol{w}}, \boldsymbol{\psi}(x_i)\rangle \mid y_i = -1\,\}\right).$$

Given $\hat{\boldsymbol{\alpha}}$ that is feasible for the dual, let

$$\hat{\boldsymbol{w}} = \frac{1}{-W(\hat{\boldsymbol{\alpha}})^{1/2}} \sum_{i=1}^{m} \hat{\alpha}_i y_i \boldsymbol{\psi}(x_i)$$

as earlier. Then duality theory gives a bound

$$-(-W(\hat{\boldsymbol{\alpha}}))^{1/2} \leq -\mu^* \leq -\hat{\mu},$$

and if equality holds then we are at the optimum.

We are now ready to provide a bound for the generalisation error. We apply the margin bound of Theorem 3.27, but some adjustments are needed.

First, we have to take the bias term b into account. Given $B > 0$ and $D > 0$, define a function class

$$\mathcal{F}_{B,D} = \{ (x, y) \mapsto y(\langle \boldsymbol{w}, \boldsymbol{\psi}(x) \rangle - b) \mid \|\boldsymbol{w}\| \leq B, |b| \leq D \}.$$

Lemma 4.15: For any $S = ((x_1, y_1), \ldots, (x_m, y_m))$ we have

$$\widehat{R}(\mathcal{F}_{B,D}, S) \leq \frac{2B}{m^{1/2}} \sqrt{\frac{1}{m} \sum_{i=1}^{m} k(x_i, x_i)} + \frac{2D}{m^{1/2}}.$$

Remark: Case $D = 0$ is just the kernelised version of Theorem 3.25.

Proof: By applying the triangle inequality and the fact that $\sup_z(p(z)+q(z)) \leq \sup_z p(z) + \sup_z q(z)$ we get

$$\begin{aligned}
\hat{R}(\mathcal{F}_{B,D}, S) &= \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sup_{\|\boldsymbol{w}\|\leq B, |b|\leq D}\left|\frac{2}{m}\sum_{i=1}^{m} r_i y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b)\right|\right] \\
&\leq \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sup_{\|\boldsymbol{w}\|\leq B}\left|\frac{2}{m}\sum_{i=1}^{m} r_i y_i \langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle\right|\right] \\
&\quad + \mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sup_{|b|\leq D}\left|\frac{2}{m}\sum_{i=1}^{m} r_i y_i b\right|\right].
\end{aligned}$$

For the first term, Theorem 3.25 gives directly

$$\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sup_{\|\boldsymbol{w}\|\leq B}\left|\frac{2}{m}\sum_{i=1}^{m} r_i y_i \langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle\right|\right] \leq \frac{2B}{m^{1/2}}\sqrt{\frac{1}{m}\sum_{i=1}^{m} k(x_i, x_i)}.$$

For the second term, notice that

$$\begin{aligned}
\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sup_{|b|\le D}\left|\frac{2}{m}\sum_{i=1}^{m} r_i y_i b\right|\right] &= \frac{2D}{m}\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\left|\sum_{i=1}^{m} r_i\right|\right] \\
&= \frac{2D}{m}\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sqrt{\sum_{i,j=1}^{m} r_i r_j}\right] \\
&\le \frac{2D}{m}\sqrt{\mathop{\mathsf{E}}_{\boldsymbol{r}\in\{-1,1\}^m}\left[\sum_{i,j=1}^{m} r_i r_j\right]} \\
&= \frac{2D}{m}\sqrt{\sum_{i,j=1}^{m}\delta_{ij}}.
\end{aligned}$$

□

Corollary 4.16: Fix $\mu > 0$, $B > 0$ and $D > 0$, and a probability measure P over $X \times \{-1, 1\}$. With probability at least $1 - \delta$ over drawing $S = ((x_1, y_1), \ldots, (x_m, y_m)) \sim P^m$, we have

$$\Pr_{(x,y)\sim P}(\operatorname{sign}(\langle \boldsymbol{w}, \boldsymbol{\psi}(x)\rangle - b) \neq y)$$

$$\leq \frac{1}{\mu m}\sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i) + \frac{4B}{\mu m}\sqrt{\sum_{i=1}^{m} k(x_i, x_i)} + \frac{4D}{\mu m^{1/2}} + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}$$

for all $\boldsymbol{w}$ and b with $\|\boldsymbol{w}\| \leq B$ and $|b| \leq D$.

Proof: Simply replace $\mathcal{F}_B$ with $\mathcal{F}_{B,D}$ in the proof of Theorem 3.27. □

Notice that this holds for any given μ, B and D, which need to be fixed independently of the sample S.

Theorem 4.17: Fix $0 \leq b_1 < \cdots < b_r$ and $0 < \mu_1 < \cdots < \mu_s$ for some $r, s \in \mathbf{N}$. Run the hard margin SVM to obtain $\boldsymbol{\alpha}^*$, b, μ^* and f. Let $\widehat{\mu} \in \{\mu_1, \ldots, \mu_s\}$ be largest such that $\widehat{\mu} \leq \mu^*$, and $\widehat{b} \in \{b_1, \ldots, b_r\}$ smallest such that $|b/\mu^*| \leq \widehat{b}$. Then with probability at least $1 - \delta$ over the draw of S we have

$$\Pr_{(x,y)\sim P}(f(x) \neq y) \leq \frac{4}{\widehat{\mu} m}\sqrt{\sum_{i=1}^{m} k(x_i, x_i)} + \frac{4\widehat{b}}{\widehat{\mu} m^{1/2}} + 3\sqrt{\frac{1}{2m}\ln\frac{4rs}{\delta}}.$$

Remark: Clearly μ^* is upper bounded by $\max_i |\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle| \leq \max_i k(x_i, x_i)^{1/2}$ (since $\|\boldsymbol{w}^*\| = 1$). The same bound holds for the unscaled bias term $b^* = b/\mu^*$ (see page 270).

Proof: Let $(\boldsymbol{w}^*, b^*)$ be the solution to the original optimisation problem. Thus in particular $b = \mu^* b^*$.

We apply Corollary 4.16 simultaneously to all $\mu \in \{\mu_1, \ldots, \mu_s\}$ and $D \in \{b_1, \ldots, b_r\}$, with $B = 1$. Since we have replaced δ with $\delta/(rs)$, the union bound implies that the bound holds for all such choices.

In particular, since $\widehat{\mu} \leq \mu^*$, we have $L_{\widehat{\mu}}(\boldsymbol{w}^*, \boldsymbol{\psi}(x_i), y_i) = 0$ for all i. Also $b^* = b/\mu^* \leq \widehat{b}$. Therefore, the classifier $\langle \boldsymbol{w}^*, \boldsymbol{\psi}(\cdot) \rangle - b^*$ satisfies the bound we claim.

The scaled classifier $f(\cdot) = \langle \mu^* \boldsymbol{w}^*, \boldsymbol{\psi}(\cdot) \rangle - \mu^* b^*$ of course has the same loss. $\square$

Soft margin SVM

Instead of having a hard constraint that the hinge losses $L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i)$ must be zero, we take them as a part of the function to be minimised.

Since the non-differentiable "hinge" would be a problem in the optimisation, we use the standard trick of introducing free variables ξ_i, the so-called slack variables, and constraining them suitably.

Optimisation 4.18: Variables $\boldsymbol{w} \in H$, $b \in \mathbf{R}$, $\boldsymbol{\xi} \in \mathbf{R}^m$, $\mu \in \mathbf{R}$

$$\begin{array}{ll} \textbf{minimise} & -\mu + C \sum_{i=1}^m \xi_i \\ \textbf{subject to} & \|\boldsymbol{w}\|^2 - 1 \leq 0 \\ & -\xi_i \leq 0 \text{ for } i = 1, \ldots, m \\ & \mu - \xi_i - y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) \leq 0 \text{ for } i = 1, \ldots, m \end{array}$$

The parameter $C > 0$ is an arbitrary positive constant. In practice a suitable value is determined by cross-validation.

Notice that after minimising w.r.t. ξ_i, the constraints imply $\xi_i = \min\{\, 0, \mu - y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle - b) \,\} = L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i)$.

The Lagrangian now becomes

$$L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -\mu + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b) - \mu + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i + \lambda(\|\boldsymbol{w}\|^2 - 1)$$

where $\alpha_i \geq 0$, $\beta_i \geq 0$ and $\lambda \geq 0$. Differentiating w.r.t. the primal variables, we get

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \boldsymbol{w}} = -\sum_{i=1}^{m}\alpha_i y_i \boldsymbol{\psi}(x_i) + 2\lambda \boldsymbol{w} \tag{6}$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial b} = \sum_{i=1}^{m}\alpha_i y_i \tag{7}$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \mu} = -1 + \sum_{i=1}^{m}\alpha_i \tag{8}$$

$$\frac{\partial L(\boldsymbol{w}, b, \mu, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda)}{\partial \xi_i} = C - \alpha_i - \beta_i. \tag{9}$$

Consider the dual

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda).$$

If one of the derivatives (2)–(4) is non-zero, then $g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -\infty$. Assume this is not the case.

Since L is convex in $\boldsymbol{w}$, we obtain the minimum by making (1) zero. This happens for

$$\boldsymbol{w} = \frac{1}{2\lambda} \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\psi}(x_i).$$

Further taking (2)–(4) to be zero gives us

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = \frac{1}{4\lambda} W(\boldsymbol{\alpha}) - \lambda$$

where

$$W(\boldsymbol{\alpha}) = -\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j) = -\left\| 2\lambda \boldsymbol{w} \right\|^2$$

like in the hard margin case.

As in the hard margin case, we maximise w.r.t. λ to obtain

$$\max_{\lambda \geq 0} g(\boldsymbol{\alpha}, \boldsymbol{\beta}, \lambda) = -(-W(\boldsymbol{\alpha}))^{1/2}$$

with the optimum at $\lambda = \frac{1}{2}(-W(\boldsymbol{\alpha}))^{1/2}$. The solution to the dual is given by

$$\boldsymbol{\alpha}^* = \arg\max_{\boldsymbol{\alpha}} -(-W(\boldsymbol{\alpha}))^{1/2} = \arg\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}).$$

Given the dual solution $(\boldsymbol{\alpha}^*, \lambda^*)$, we already know that

$$\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i) = \frac{1}{(-W(\boldsymbol{\alpha}^*))^{1/2}} \sum_{i=1}^{m} \alpha_i^* y_i \boldsymbol{\psi}(x_i).$$

The remaining question is, how to get b^* and μ^*.

The constraint $\beta_i \geq 0$ and the KKT condition $C - \alpha_i - \beta_i = 0$ together imply the box constraint

$$0 \leq \alpha_i \leq C.$$

Intuitively, increasing α_i increases the margin of $\boldsymbol{w}$ on the example $(\boldsymbol{\psi}(x_i), y_i)$. In the hard margin case, we increase α_i as much as needed to push the slack variable to zero. In the soft margin case, we stop increasing α_i at some stage and allow the slack variable to get positive.

More formally, the complementary slackness conditions are

$$\begin{aligned} \alpha_i(y_i(\langle \boldsymbol{w}, \boldsymbol{\psi}(x_i)\rangle - b) - \mu + \xi_i) &= 0 \\ (C - \alpha_i)\xi_i &= 0. \end{aligned}$$

If $\xi_i > 0$, then $\alpha_i = C$. If $\alpha_i = 0$, then $\xi_i = 0$.

Assume first that there are some i and j such that $y_i = -1$, $y_j = 1$ and $0 < \alpha_i^*, \alpha_j^* < C$. Then $\xi_i = \xi_j = 0$, and

$$y_i(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle - b^*) - \mu^* = 0 = y_j(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle - b^*) - \mu^*$$

so

$$\begin{aligned} b^* &= \frac{1}{2}(\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle + \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle) \\ \mu^* &= \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle - b^*. \end{aligned}$$

Further,

$$\begin{aligned} \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i)\rangle &= \min\{\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k)\rangle \mid \alpha_k > 0, y_k = -1\} \\ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j)\rangle &= \max\{\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k)\rangle \mid \alpha_k > 0, y_k = 1\}. \end{aligned}$$

We can use these latter conditions to pick i and j, and then calculate b^* and μ^*, even in the special case that i and j such that $0 < \alpha_i^*, \alpha_j^* < C$ does not exist.

We summarise the preceding observations.

Algorithm 4.19 (1-norm Soft Margin SVM): Parameter: $C > 0$
Input: sample $((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$

1. Obtain $\boldsymbol{\alpha}^* \in \mathbf{R}^m$ as maximiser of $W(\boldsymbol{\alpha}) = \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(x_i, x_j)$ subject to $\sum_{i=1}^m \alpha_i = 1$, $\sum_{i=1}^m \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$ for $i = 1 \leq i \leq m$.

2. Let $\lambda^* = \frac{1}{2}(-W(\boldsymbol{\alpha}^*))^{1/2}$ and $\boldsymbol{w}^* = \frac{1}{2\lambda^*} \sum_{i=1}^m \alpha_i^* y_i \boldsymbol{\psi}(x_i)$

3. Choose i and j such that
$\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle = \min \{ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k) \rangle \mid \alpha_k > 0, y_k = -1 \}$ and
$\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle = \max \{ \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_k) \rangle \mid \alpha_k > 0, y_k = 1 \}$.

4. Let $b^* = \frac{1}{2} (\langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_i) \rangle + \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle)$ and $\mu^* = \langle \boldsymbol{w}^*, \boldsymbol{\psi}(x_j) \rangle - b^*$.

5. Output the classifier $\operatorname{sign}(f(\cdot))$ where

$$f(\cdot) = \frac{1}{2\lambda^*} \sum_{i=1}^m \alpha_i^* y_i k(\cdot, x_i) - b^*.$$

Theorem 4.20: Fix $D > 0$, $\mu > 0$ and $C > 0$. Run the 1-norm soft margin SVM with parameter C to obtain $\boldsymbol{\alpha}^*$, b^*, μ^* and f. If $|b| \leq D$, then with probability at least $1 - \delta$ over the draw of S we have

$$\Pr_{(x,y)\sim P}(f(x) \neq y) \leq \frac{1}{\mu m}\sum_{i=1}^{m} L_\mu(\boldsymbol{w}, \boldsymbol{\psi}(x_i), y_i) + \frac{4}{\mu m}\sqrt{\sum_{i=1}^{m} k(x_i, x_i)} + \frac{4D}{\mu m^{1/2}} + 3\sqrt{\frac{1}{2m}\ln\frac{4}{\delta}}.$$

Proof: Like for hard SVM, except now the slacks may be non-zero. □

Applying the bound gets a bit complicated, since μ needs to be fixed in advance so we **cannot** pick $\mu = \mu^*$. A grid system like in the hard margin case can be applied, but we omit the details.

The ν SVM is just the soft margin SVM with $C = 1/(\nu m)$ for some $0 < \nu \leq 1$. This parameterisation has an intuitive interpretation.

Theorem 4.21: Run the ν SVM on sample S to obtain $\boldsymbol{w}^*$, b^* and μ^*. Then at most νm examples in S have margin less than μ^*, and at most $(1-\nu)m$ examples in S have margin larger than μ^*.

Proof: Recall that $\sum_{i=1}^{m} \alpha_i = 1$.

If $\xi_i > 0$, then $\alpha_i = 1/(\nu m)$. Since $\alpha_i \geq 0$, this can happen at most νm times.

Similarly, since $\alpha_i \leq 1/(\nu m)$, we need at least νm examples with $\alpha_i > 0$. They all have margin at most μ^*. $\square$

Intutively, we can pick ν to correpond to the "noise rate" in the data. We allow a fraction ν of the data to fail to have a large margin. (In practice, cross-validation is still needed.)

5. Boosting

Boosting is a generic aggregation method that tries to create a good hypothesis by combining mediocre ones. The main topics of the section are

- bias-variance tradeoff as a motivation for aggregating,
- boosting in PAC learning theory,
- practical boosting algorithms, in particular AdaBoost and
- understanding boosting in terms of margins and optimisation.

As with SVMs, we focus on binary classification. The technique can be generalised to multiclass problems and regression, but the way to do this is not always obvious.

5.1 Bias-variance tradeoff

For simplicity, we start with regression. Assume there is a distribution P over $X \times Y$ with $Y \subseteq \mathbf{R}$, and we wish to find $h\colon X \to \mathbf{R}$ with small squared loss

$$L(h) = \mathop{\mathsf{E}}_{(x,y)\sim P}[(y - h(x))^2].$$

We consider our learning algorithm as generating h according to some probability measure D over the space of all functions $X \to \mathbf{R}$. We include into D the randomness due to the drawing of the sample and any internal randomisations of the algorithm. However, we consider the sample size as a fixed parameter of the algorithm. Let $\bar{h}$ be the "average hypothesis":

$$\bar{h}(x) = \mathop{\mathsf{E}}_{h\sim D}[h(x)] \quad \text{for all } x \in X.$$

We wish to estimate the average loss

$$\mathop{\mathsf{E}}_{h\sim D}[L(h)] = \mathop{\mathsf{E}}_{h\sim D}\mathop{\mathsf{E}}_{(x,y)\sim P}[(y - h(x))^2].$$

We have

$$\begin{aligned}
\mathop{\mathsf{E}}_{h\sim D}\mathop{\mathsf{E}}_{(x,y)\sim P}[(y-h(x))^2] &= \mathop{\mathsf{E}}_{h\sim D}\mathop{\mathsf{E}}_{(x,y)\sim P}[y^2-2yh(x)+h(x)^2] \\
&= \mathop{\mathsf{E}}_{(x,y)\sim P}[y^2-2y\bar{h}(x)+\bar{h}(x)^2-\bar{h}(x)^2+\mathop{\mathsf{E}}_{h\sim D}[h(x)^2]] \\
&= L(\bar{h})+\mathop{\mathsf{E}}_{h\sim D}[\mathop{\mathsf{E}}_{(x,y)\sim P}[h(x)^2-\bar{h}(x)^2]] \\
&= L(\bar{h})+\mathop{\mathsf{E}}_{h\sim D}[\mathop{\mathsf{E}}_{(x,y)\sim P}[(\bar{h}(x)-h(x))^2]]
\end{aligned}$$

where the last step uses $\mathsf{E}_{h\sim D}[2\bar{h}(x)h(x)] = 2\bar{h}(x)^2$.

We call $L(\bar{h}) = \mathsf{E}_{(x,y)\sim P}[(y-\bar{h}(x))^2]$ the bias term and $\mathsf{E}_{h\sim D}[\mathsf{E}_{(x,y)\sim P}[(\bar{h}(x)-h(x))^2]]$ the variance term.

The bias term measures how well our algorithm would do if we could eliminate all randomness. Remember that this includes randomness due to finite sample size.

It is easy to see that for minimising the bias, the best choice for the learning algorithm would be $\bar{h}(x) = y^*(x)$ for all x where

$$y^*(x) = \mathop{\mathsf{E}}_{(X,Y)\sim P}[Y \mid X = x].$$

This would still leave a nonzero term due to the variance of y.

To minimise the bias term, the best one could do in practice would be empirical risk minimisation with a large hypothesis class. However, in practice one does **not** want to minimise just the variance term.

Although we have averaged away from the bias term the effect of an individual sample, it often still depends on the sample size. For example, the SVM (when properly generalised to regression) finds $\boldsymbol{w}$ with small 2-norm, which biases the predictions towards zero. Increasing sample size allows the hypothesis to get further away from zero.

The variance term $\mathsf{E}_{h\sim D}[\mathsf{E}_{(x,y)\sim P}[(\bar{h}(x) - h(x))^2]]$ measures the extra error on top of $L(\bar{h})$ due to the variability in h. In particular, this includes the effects of finite sample size.

A zero variance is trivially obtained by using a hypothesis class with exactly one hypothesis. Naturally, this would give a very large bias. More realistically, we can aim for small variance by using a small hypothesis class.

Even more interestingly, variance can be reduced by aggregating. Consider drawing k hypotheses $h_1, \ldots, h_k$ independently from D, and let $\widehat{h} = \frac{1}{k}\sum_{i=1}^{k} h_i$. Then the basic result about the variance of an average of independent random variables implies

$$\mathop{\mathsf{E}}_{h_1,\ldots,h_k\sim D}[\mathop{\mathsf{E}}_{(x,y)\sim P}[(\bar{h}(x) - \widehat{h}(x))^2]] = \frac{1}{k}\mathop{\mathsf{E}}_{h\sim D}[\mathop{\mathsf{E}}_{(x,y)\sim P}[(\bar{h}(x) - h(x))^2]]$$

so we cut the variance term by a factor $1/k$.

In practice getting k independent hypotheses would require splitting the sample into k parts. This would typically change $\bar{h}$ and increase the bias term, so the above reasoning is not quite valid as such. Nevertheless, it motivates the aggregating method known as bagging.

Algorithm 5.1 (Bagging):

given: sample $S \in (X \times \mathbf{R})^m$
learning algorithm A
for $i := 1$ **to** k **do**
Obtain sample $S_i \in (X \times \mathbf{R})^l$ by drawing k examples from S with replacement.
$h_i := A(S_i)$
return $\widehat{h} = \frac{1}{k} \sum_{i=1}^{k} h_i$.

Bagging can be used also for binary classification, returning the majority vote hypothesis

$$\bar{h}(x) = \text{sign} \left(\sum_{i=1}^{k} h_i(x) \right).$$

Bagging is a very useful practical technique.

The bias-variance tradeoff is a phrase used informally to discuss the conflicting subgoals of minimising bias and minimising variance. We use it also for classifier learning, although formally the 0-1 loss does not decompose like square loss.

For example, we think of Bagging as primarily a variance reduction method. Hence, we expect it to be most efficient when the hypothesis class is rich (decision trees, neural networks), meaning originally a low bias but large variance.

We already met the bias-variance tradeoff in discussing model selection. In the figure on page 186, we interpret the $\widehat{\mathrm{err}}(h_i)$ curve as representing bias and ε_i as (an upper bound for) variance.

The bias-variance tradeoff is made explicit by algorithms that minimise a regularised loss

$$\frac{1}{m}\sum_{i=1}^{m} L(y_i, h(x_i)) + \lambda C(h)$$

where C is some complexity measure and $\lambda > 0$ a tradeoff parameter. Large λ puts more weight on the variance term. The soft-margin SVM is an example of this approach; there $C(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$.

5.2 Strong and weak learnability

For this subsection, we consider the classical (unrealistic) PAC model. It is a theoretical motivation that leads to a practical algorithm.

We have a known concept class C. A probability measure P is consistent with C, if a target concept $f^* \in C$ exists such that $\Pr_{(x,y)\sim P}(y = f^*(x)) = 1$. We use $D(x) = \sum_y P(x, y)$ to denote the marginal distribution over X. Then

$$\mathrm{err}(h) = \Pr_{x\sim D}(h(x) \neq f^*(x)).$$

Let $0 < \varepsilon, \delta \leq 1$. We say that an algorithm A learns C with accuracy ε and confidence δ using sample size m, if the hypothesis h of A satisfies

$$\Pr_{S\sim P^m}(\mathrm{err}(h) > \varepsilon) \leq \delta$$

for all P that are consistent with C.

A concept class C is strongly PAC learnable if there is an algorithm A and a polynomial $p(\cdot,\cdot)$ such that for all $0 < \varepsilon, \delta \leq 1$ the algorithm A learns C with accuracy ε and confidence δ using sample size $\lceil p(1/\varepsilon, 1/\delta) \rceil$.

A concept class is weakly PAC learnable if there is some algorithm A and some values $0 < \delta \leq 1$, $0 < \varepsilon < 1/2$ and $m > 0$ such that A learns C with accuracy ε and confidence δ using sample size m.

Notice that $\varepsilon = 1/2$ is trivially achieved by random guessing. In weak learning, we only require that we have some fixed, nonzero edge $\gamma = 1 - 2\varepsilon > 0$ over random guessing.

When we say just "PAC learnable" we mean "strongly PAC learnable".

The original boosting algorithm was proposed as a solution to the big theoretical question about whether weak learnability implies strong learnability. Quite surprisingly, the answer turned out to be **yes**!

Originally the notion of PAC learnability also included polynomial computation time, but we do not consider that here. With this definition, we already know from Chapter 3 that strong PAC learnability is equivalent with having finite VC dimension. The question is how to characterise weak learnability.

Consider some fixed $0 < \delta_0 < 1$. Assume some A learns with some accuracy ε and confidence δ_0 using sample size m.

It is easy to see that for any $0 < \delta < 1$, we can achieve accuracy 2ε with confidence δ using sample size $mf(1/\varepsilon, \ln(1/\delta))$ where f is polynomial. We just run A with $O(\ln(1/\delta))$ independent samples and use an independent test set to pick the best result.

Hence, we could as well fix δ also in the definition of strong learnability. From now on we mainly ignore the confidence parameter.

Like we mentioned, weak learnability actually implies strong learnability. The crucial thing turns out to be the quantification for all P in the definitions. Given a weak learning algorithm A and a sample S, we can "boost" A as follows.

1. Split S into three parts S_1, S_2, S_3 (not necessarily same size). Run A on S_1 to obtain hypothesis h_1.

2. Create sample S_2' by removing from S_2 all examples (x, y) such that $h_1(x) = y$. Run A on S_2' to obtain h_2.

3. Create sample S_3' by removing from S_3 all examples (x, y) on which $h_1(x) = h_2(x)$. Run A on S_3' to obtain h_3.

4. Return the majority vote $\widehat{h} = \text{sign}(h_1 + h_2 + h_3)$.

The key is the filtering we do to samples S_2 and S_3, and the assumption that A also works for the filtered distributions.

A carefully engineered recursive application of this procedure can be shown to boost a weak learner into a strong one. However, the algorithm will not be robust enough to be practically very useful.

5.3 AdaBoost

For AdaBoost, we need to consider learning algorithms A takes as input a sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$ together with a weight vector $\boldsymbol{d} \in [0, 1]^m$ such that $\sum_{i=1}^n d_i = 1$. As usual, A returns a hypothesis $h = A(S, \boldsymbol{d})$ which is a mapping $h \colon X \to \{-1, 1\}$.

Given a standard algorithm A that inputs only an unweighted sample, there are (at least) two ways to get A' that works with weights:

1. For any A, we can run A on sample S' where S' is obtained by sampling from S according to distribution $\boldsymbol{d}$ (with replacement).

2. If A is based on minimising a loss $\sum_{i=1}^n L(h, x_i, y_i)$, we can change the loss to $\sum_{i=1}^n d_i L(h, x_i, y_i)$.

It is not clear which of these gives the best result, but often method 2 is easier to implement.

The idea of AdaBoost is to call A repeatedly, always concentrating the weight on examples that currently are not classified correctly. In this context, we call A the weak learner.

If A is a weak PAC learning algorithm, **then** applying AdaBoost to it will result in a strong PAC learning algorithm. Otherwise (*i.e.*, in practice) we can still apply AdaBoost and quite often get an improvement in accuracy.

The error of the hypothesis h with respect to the weighted sample is defined as

$$\widehat{\mathrm{err}}(h, S, \boldsymbol{d}) = \sum_{i=1}^{n} d_i I(y_i \neq h(x_i))$$

where $I(\phi) = 1$ if ϕ is true and $I(\phi) = 0$ otherwise.

Often it is more convenient to measure the quality of h by its edge

$$\hat{\gamma}(h, S, \boldsymbol{d}) = 1 - 2\widehat{\mathrm{err}}(h, S, \boldsymbol{d})$$

which is in $[0, 1]$ assuming $\widehat{\mathrm{err}} \leq 1/2$. Then $\widehat{\mathrm{err}} = \frac{1}{2} - \frac{1}{2}\hat{\gamma}$.

Algorithm 5.2 (AdaBoost): Input sample $S \in (X \times \{-1, 1\}^m)$, stopping time T.

1. Initialize $d_i^{(1)} = 1/m$ for $i = 1, \ldots, m$.

2. Repeat for $t = 1, \ldots, T$:

   (a) Let $h_t = A(S, \boldsymbol{d}^{(t)})$ and $\varepsilon_t = \widehat{\mathrm{err}}(h_t, S, \boldsymbol{d}^{(t)})$. If $\varepsilon_t \geq 1/2$, set $T = t - 1$ and break from the loop. If $\varepsilon_t = 0$, return h_t.

   (b) Update

$$d_i^{(t+1)} = \frac{1}{Z_t} d_i^{(t)} \exp\left(-\alpha_t y_i h_t(x_i)\right)$$

   where Z_t is a normalising factor such that $\sum_{i=1}^m d_i^{(t+1)} = 1$, and

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}.$$

3. Output H where

$$H(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

We call each h_t a weak hypothesis. Since small ε_t gives a large α_t, the final hypothesis gives more weight to the good weak hypotheses.

Since $\alpha_t > 0$, the weight update removes weight from those examples that h_t classifies correctly and gives it to those that h_t classifies incorrectly.

To motivate the precise formulas, define f_t by

$$f_t(x) = \sum_{j=1}^{t} \alpha_j h_j(x).$$

It turns out that $\alpha = \alpha_t$ minimises

$$G_t(\alpha) = \sum_{i=1}^{m} \exp\left(-\alpha y_i h_t(x_i) - y_i f_{t-1}(x_i)\right).$$

Therefore, AdaBoost can be seen as trying to find f that minimises $\sum_{i=1}^{m} \exp(-y_i f(x_i))$, which gives an exponential penalty for margin $y_i f(x_i)$ being negative.

The weight update is related to a convex optimisation that has G_t as its dual. More of this later.

Although AdaBoost minimises a margin-based loss function, it does not necessarily find $\boldsymbol{\alpha}$ that would maximise the normalised margin

$$\min_i \frac{\sum_t y_i \alpha_t h_t(x_i)}{\sum_t |\alpha_t|}.$$

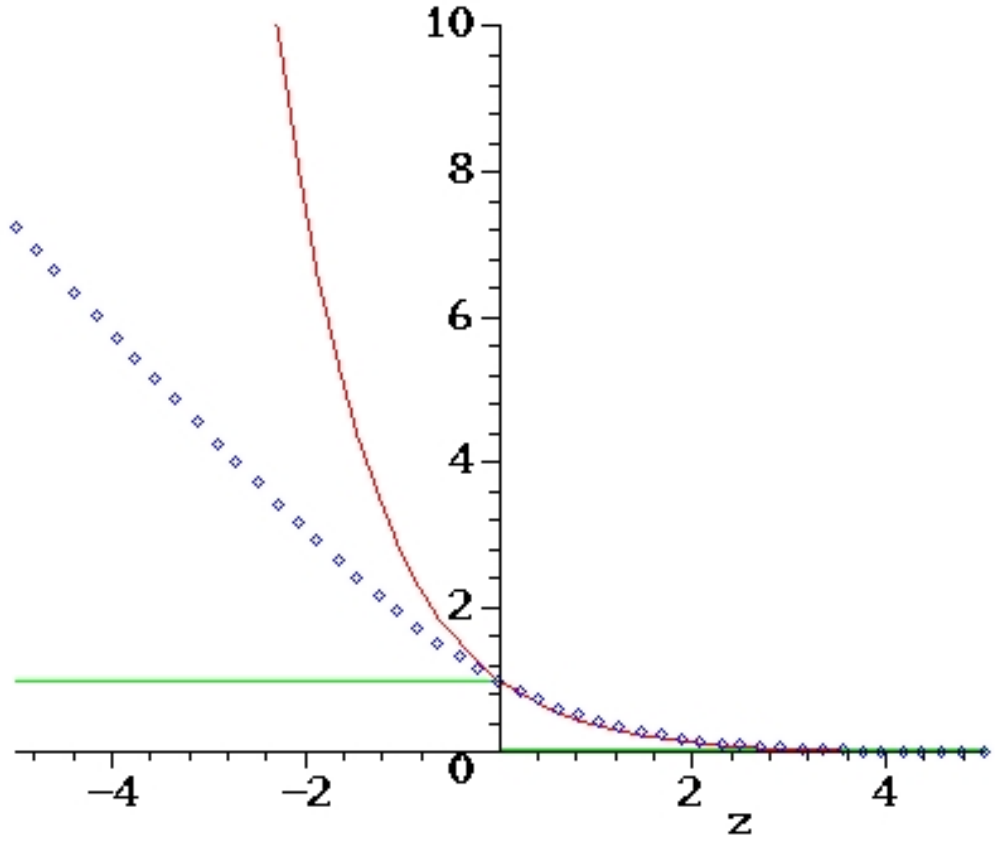
There are other algorithms that do minimise the margin.

However, for large enough T, AdaBoost does concentrate the weight on examples on which it has difficulty getting a positive margin. In practice, this means outliers and noisy examples. With very high noise rates, this is a problem. A more robust approach is to use a less extreme loss function, such as

$$G_t^{(\log)}(\alpha) = \sum_{i=1}^{m} \log_2 \left(1 + \exp(-\alpha y_i h_t(x_i) - y_i f_{t-1}(x_i)\right)$$

that penalises negative margins only linearly. The loss function $G^{\log}$ leads to algorithm known as LogitBoost.

For moderate noise rates, the basic AdaBoost works quite well. In fact, its failure to overfit even for very large T caused initially a lot of bafflement.



AdaBoost (solid curve) and LogitBoost (dots) cost functions in terms of the margin $z = yf(x)$. Both are upper bounds for $I(z < 0)$.

Linear classifiers, and many others, actually produce a real value $h(x)$, where $\operatorname{sign}(h(x))$ is the class prediction and $|h(x)|$ can be interpreted as a confidence.

For non-binary hypotheses, the error $\widehat{\mathrm{err}}$ is not meaningful. We re-define the edge as

$$\hat{\gamma}(h, S, \boldsymbol{d}) = \sum_{i=1}^{m} d_i y_i h(x_i)$$

which for $h(x) \in \{ -1, 1 \}$ coincides with the old definition.

The formula used for α_t in Algorithm 5.2 gives the minimiser of G_t in the special case $h_t(x_i) \in \{ -1, 1 \}$ for all i. In the general case, the optimisation needs to be solved numerically (or some heuristic used).

The final hypothesis should also be confidence-rated, and suitably scaled.

We summarise the changes in the following.

Algorithm 5.3 (Confidence-rated AdaBoost): like Algorithm 5.2, except that

- We assume $h_t(x) \in \mathbf{R}$.
- We break if the edge is nonpositive, *i.e.*, $\hat{\gamma}(h_t, S, \boldsymbol{d}^{(t)}) \leq 0$.
- We use

$$\alpha_t = \arg\min_{\alpha} \sum_{i=1}^{m} \exp\left(-\alpha y_i h_t(x_i) - y_i f_{t-1}(x_i)\right).$$

- The final hypothesis is

$$F(x) = \frac{f_T(x)}{\sum_{t=1}^{T} \alpha_t} = \frac{\sum_{t=1}^{T} \alpha_t h_t(x)}{\sum_{t=1}^{T} \alpha_i}.$$

For theoretical considerations, suppose A is a weak PAC learning algorithm for some concept class C. More specifically, assume A has error $\varepsilon < 1/2$ and confidence δ using sample size m. Write $\gamma = 1 - 2\varepsilon$.

Consider the following algorithm for learning from weighted samples $(S, \boldsymbol{d})$:

1. Create a sample S' by drawing m times (with replacement) from S using distribution $\boldsymbol{d}$. Let $h = A(S')$.

2. If $\widehat{\gamma}(h, S, \boldsymbol{d}) \geq \gamma$, return h. Otherwise fail.

Assume that the sample S originally came from a distribution consistent with C. Since $\widehat{\mathrm{err}}(h, S, \boldsymbol{d})$ is the true error rate of h for the distribution from which S' was formed, we have $\widehat{\gamma}(h, S, \boldsymbol{d}) \geq \gamma$ with probability at least $1 - \delta$.

We can decrease the failure probability to any $\delta' > 0$ by repeating $O(\ln(1/\delta'))$ times.

Therefore, if we assume weak learnability, we can assume the existence of an algorithm with a guaranteed edge at least γ for some constant $\gamma > 0$.

The basic AdaBoost (Algorithm 5.2) has the following bound for the fraction of examples that fail to have margin μ.

Theorem 5.4: Write $\gamma_t = \widehat{\gamma}(h_t, S, \boldsymbol{d}^{(t)})$. For any $\mu \geq 0$, AdaBoost satisfies

$$\frac{1}{m}\sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) \leq \prod_{t=1}^{T} (1-\gamma_t)^{(1-\mu)/2}(1+\gamma_t)^{(1+\mu)/2}.$$

Proof: If $y_i F(x_i) \leq \mu$, then $y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) - \mu \sum_{t=1}^{T} \alpha_i \leq 0$. Therefore

$$I(y_i F(x_i) \leq \mu) \leq \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) + \mu \sum_{t=1}^{T} \alpha_i\right).$$

We will first show

$$\frac{1}{m}\sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) \leq \exp\left(\mu \sum_{t=1}^{T} \alpha_t\right) \prod_{t=1}^{T} Z_t.$$

By induction we have

$$d_i^{(T+1)} = \frac{1}{m}\left(\prod_{t=1}^{T} Z_t\right)^{-1} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right).$$

We get

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) &\leq \frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) + \mu \sum_{t=1}^{T} \alpha_i\right) \\
&= \exp\left(\mu \sum_{t=1}^{T} \alpha_t\right)\left(\prod_{t=1}^{T} Z_t\right) \sum_{i=1}^{m} d_i^{(T+1)} \\
&= \exp\left(\mu \sum_{t=1}^{T} \alpha_t\right)\left(\prod_{t=1}^{T} Z_t\right)
\end{aligned}$$

because $\sum_{i=1}^{m} d_i^{(T+1)} = 1$. We have not used any assumptions on h_t or α_t yet, so this intermediate bounds holds also for the confidence-rated version with arbitrary choice of α_t.

Assume now $h_t(x) \in \{-1, 1\}$. For all t we have

$$\begin{aligned} Z_t &= \sum_{i=1}^{m} d_i^{(t)} \exp(-y_i \alpha_t h_t(x_i)) \\ &= \exp(-\alpha_t) \sum_{y_i h_t(x_i)=1} d_i^{(t)} + \exp(\alpha_t) \sum_{y_i h_t(x_i)=-1} d_i^{(t)} \\ &= \exp(-\alpha_t) \left(\frac{1}{2} + \frac{1}{2}\gamma_t\right) + \exp(\alpha_t) \left(\frac{1}{2} - \frac{1}{2}\gamma_t\right). \end{aligned}$$

For $\alpha_t = (1/2)\ln((1+\gamma_t)/(1-\gamma_t))$, we now have $Z_t = (1+\gamma_t)^{1/2}(1-\gamma_t)^{1/2}$. Since further

$$\exp\left(\mu \sum_{t=1}^{T} \alpha_t\right) = \prod_{t=1}^{T} \left(\frac{1+\gamma_t}{1-\gamma_t}\right)^{\mu/2},$$

we get

$$\frac{1}{m} \sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) \leq \prod_{t=1}^{T} (1+\gamma_t)^{(1+\mu)/2} (1-\gamma_t)^{(1-\mu)/2}.$$

□

Assume now that $\gamma_t \geq \gamma > 0$ for all t. Then

$$\frac{1}{m}\sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) \leq \left((1+\gamma)^{1+\mu}(1-\gamma)^{1-\mu}\right)^{T/2}.$$

It can be shown that for $\mu \leq \gamma/2$ we have

$$(1+\gamma)^{1+\mu}(1-\gamma)^{1-\mu} < 1,$$

so the fraction of margin errors goes to zero at an exponential rate as T increases.

Thus, eventually AdaBoost achieves a margin at least $\gamma/2$. We later see that the optimum margin would actually be at least γ. Indeed often AdaBoost fails to achieve a margin better than half the optimum.

The name AdaBoost comes from the fact that previous boosting algorithms needed to be tuned assuming a known lower bound γ for the edge. This was of course very impractical.

Although we still find it convenient to write theoretical bounds for AdaBoost assuming a fixed γ, the algorithm itself adapts to the observed edges γ_t.

Considering $\mu = 0$ and assuming $\gamma_t \geq \gamma > 0$, we see that

$$\widehat{\mathrm{err}}(H, S) = \frac{1}{m} \sum_{i=1}^{m} I(y_i F(x_i) \leq 0) \leq (1 - \gamma^2)^{T/2} \leq \exp\left(-\frac{\gamma^2 T}{2}\right).$$

In other words, $\widehat{\mathrm{err}}(H, S) \leq \varepsilon$ holds for $T \geq T(\varepsilon)$ where

$$T(\varepsilon) = \left\lceil \frac{1}{2\gamma^2} \ln \frac{1}{\varepsilon} \right\rceil.$$

Since the smallest nonzero error is $1/m$, we see in particular that $\widehat{\mathrm{err}}(H, S) = 0$ for

$$T > \frac{1}{2\gamma^2} \ln m.$$

Hence, assuming weak learnability, we can get zero training error in $O(\log m)$ iterations.

What about the generalisation error?

We sketch the original proof for the PAC learning property of AdaBoost.

Given a hypothesis class H and a parameter $T > 0$, let $C_T(H)$ be the class of hypotheses of the form $\text{sign}(\alpha_1 h_1 + \ldots + \alpha_T h_T)$ where $\alpha_i \in \mathbf{R}$ and $h_i \in H$. Thus, running AdaBoost for T iterations with weak hypothesis class H gives a final hypothesis from class $C_T(H)$.

Theorem 5.5: If $\text{VCdim}(H) = d < \infty$, then

$$\text{VCdim}(C_T(H)) \leq 2(d+1)(T+1)\log_2(\text{e}(T+1)) = O(dT \log T).$$

We omit the proof.

We combine this with the VC bound for PAC learning:

Theorem 5.6: If $\text{VCdim}(H) = d < \infty$ and for any S consistent with H the algorithm A produces a hypothesis $A(S) \in H$ such that $\widehat{\text{err}}(A(S), S) = 0$, then A is a strong PAC learning algorithm for H using sample size

$$O\left(\frac{1}{\varepsilon}\left(d \log \frac{1}{\varepsilon} + \log \frac{1}{\delta}\right)\right).$$

This is basically the result from Chapter 3, but somewhat sharper due to the assumption that S is consistent with H. Again, we omit the proof.

Combining the previous remarks, we get the following.

Corollary 5.7: If A is a weak PAC learning algorithm for H, then AdaBoost is a strong PAC learning algorithm for H.

Proof sketch: If H is weakly learnable, it has a finite VC dimension d.

If A is a weak learning algorithm, we can assume a constant edge γ. Therefore, AdaBoost achieves $\widehat{\mathrm{err}}(H, S) = 0$ within $T = O(\log m)$ iterations.

Hence, AdaBoost produces a consistent hypothesis from a hypothesis class $C_T(H)$ with VC dimension

$$\tilde{d} = O(d \log m \log \log m).$$

By Theorem 5.6, it is sufficient to have

$$m \geq \frac{c}{\varepsilon}\left(\tilde{d} \log \frac{1}{\varepsilon} + \log \frac{1}{\delta}\right)$$

for some $c > 0$. This clearly holds for a suitable $m = \mathrm{poly}(1/\varepsilon, 1/\delta)$. □

How many round should we boost (*i.e.*, what is a good T)? Write

$$F_t = \frac{\sum_{i=1}^{t} \alpha_t h_t}{\sum_{i=1}^{t} \alpha_t}.$$

In practice, the weak learner may eventually fail to get a positive edge, and we have to stop. However, this can take quite long, and may not happen at all for sufficiently strong weak learners.

Given the previous bound, structural risk minimisation theory would suggest stopping boosting as soon as the empirical error $\widehat{\text{err}}(F_t, S)$ reaches zero, or even sooner.

Empirically, it has been noticed that when noise level in the data is moderate, the true error $\text{err}(F_t, P)$ keeps decreasing when t is increased even after the point where empirical error reaches zero.

Theorem 5.4 suggests that after reaching zero empirical error, *i.e.*, zero margin on all examples, AdaBoost keeps going for a larger positive margin.

We now show that large margin should indeed help generalisation performance.

Theorem 5.8: Let $\mathcal{F}$ be a class of functions $X \times Y \to \mathbf{R}$, and P a probability measure on $X \times Y$. For $S \sim P^m$, with probability at least $1 - \delta$ we have

$$\Pr_{(x,y)\sim P}(f(x,y) \le 0) \quad \le \quad \frac{1}{m}\sum_{i=1}^{m} I(f(x_i,y_i) \le \mu) + \frac{8R_m(\mathcal{F})}{\mu} + \sqrt{\frac{\ln \log_2(2/\mu)}{m}} + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}$$

for all $f \in \mathcal{F}$ and $0 < \mu < 1$.

We omit the proof.

The bound holds at the same time for all μ, unlike Theorem 3.27. This causes the extra log log term.

There is a trade-off in the choice of μ. Large μ causes more margin errors (first term) but increases the denominator in the second term.

Theorem 5.9: Let P be a probability measure over $X \times \{-1, 1\}$. Let A be a learning algorithm with hypothesis class H, where each $h \in H$ is a function $X \to \mathbf{R}$. With probability at least $1 - \delta$, the hypothesis F produced by AdaBoost satisfies for all $0 < \mu < 1$ the bound

$$\Pr_{(x,y)\sim P}(yF(x) \leq 0) \leq \frac{1}{m}\sum_{i=1}^{m} I(y_i F(x_i) \leq \mu) + \frac{8R_m(H)}{\mu} + \sqrt{\frac{\ln \log_2(2/\mu)}{m}} + \sqrt{\frac{1}{2m}\ln\frac{2}{\delta}}.$$

Proof: Notice that $F \in \text{conv}(H)$ where

$$\text{conv}(H) = \left\{ \sum_t \alpha_t h_t \mid \alpha_t \geq 0, \sum_t \alpha_t = 1 \right\}.$$

We know (Exercise 7.1) that $R_m(\text{conv}(H)) = R_m(H)$.

We apply Theorem 5.8 with $\mathcal{F}$ consisting of functions $(x, y) \mapsto yF(x)$ where $F \in \text{conv}(H)$. It is easy to see that $R_m(\mathcal{F}) = R_m(\text{conv}(H))$. □

5.4 Boosting as convex optimisation

Fix a hypothesis class H, and suppose we have an "optimal" weak learner that always returns a hypothesis $h \in H$ with maximal edge. We write an optimisation problem to determine the smallest edge γ_* that could occur.

For simplicity, assume H is finite, with $H = \{\tilde{h}_1, \ldots, \tilde{h}_n\}$. (We use $\tilde{h}_i$ to differentiate from the actual weak hypotheses h_t chosen by AdaBoost.) Then γ_* is the solution to

Optimisation 5.10: Variables $\gamma \in \mathbf{R}$, $\boldsymbol{d} \in \mathbf{R}^m$

$$\begin{array}{ll} \textbf{minimise} & \gamma \\ \textbf{subject to} & \sum_{i=1}^m d_i y_i \tilde{h}_j(x_i) \leq \gamma \text{ for } j = 1, \ldots, n \\ & d_i \geq 0,\ \sum_{i=1}^m d_i = 1 \end{array}$$

The optimal $\boldsymbol{d}$ gives the worst-case distribution.

This is a convex (actually, linear) problem. We next write the dual.

The Lagrangian is

$$L(\gamma, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{\lambda}, \mu) = \gamma + \sum_{j=1}^{n} w_j \left(\sum_{i=1}^{m} d_i y_i \tilde{h}_j(x_i) - \gamma \right) - \sum_{i=1}^{m} \beta_i d_i + \mu \left(1 - \sum_{i=1}^{m} d_i \right)$$

where $w_i, \beta_i \geq 0$. Differentiate:

$$\begin{aligned} \frac{\partial L}{\partial \gamma} &= 1 - \sum_{j=1}^{n} w_j \\ \frac{\partial L}{\partial d_i} &= \sum_{j=1}^{n} w_j y_i \tilde{h}_j(x_i) - \beta_i - \mu. \end{aligned}$$

Setting $\partial L / \partial \gamma = 0$ gives $\sum_{j=1}^{n} w_i = 1$.

Setting $\partial L / \partial d_i = 0$ and observing $\beta_i \geq 0$ gives $\sum_{j=1}^{n} w_j y_i \tilde{h}_j(x_i) \geq \mu$.

Plugging all this back into L and simplifying gives

$$\min_{\gamma,\boldsymbol{d}} L(\gamma, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{\lambda}, \mu) = \mu.$$

Hence, the dual problem is

Optimisation 5.11: Variables $\mu \in \mathbf{R}$, $\boldsymbol{w} \in \mathbf{R}^n$

maximise	μ
subject to	$y_i \left(\sum_{j=1}^n w_j \tilde{h}_j(x_i) \right) \geq \mu$ for $i = 1, \ldots, m$
	$w_j \geq 0$, $\sum_{j=1}^n w_j = 1$.

Notice that $y_i(\sum_{j=1}^n w_j \tilde{h}_j(x_i))$ is the margin of the hypothesis $\sum_j w_j \tilde{h}_j$ on (x_i, y_i).

Thus the optimal value μ_* is the best margin we can obtain using a convex combination of base hypotheses from H.

Because of strong duality, we have $\gamma_* = \mu_*$.

We can write this as a minimax equation:

$$\min_{\boldsymbol{d}} \max_{\tilde{h} \in H} \sum_{i=1}^{m} d_i y_i \tilde{h}(x_i) = \max_{\boldsymbol{w}} \min_{1 \le i \le m} y_i \sum_{j=1}^{n} w_j \tilde{h}_j(x_i)$$

where $d_i, w_i \geq 0$ and $\sum_i d_i = \sum_j w_j = 1$.

In other words, the minimum edge to which we can force the weak learner is the same as the maximum margin we can achieve.

This can be directly generalised for countably infinite H, and with some regularity conditions for uncountable H.

If $H = \{ \tilde{h}_1, \ldots, \tilde{h}_n \}$ is finite, we can interpret it as a feature map $\boldsymbol{\psi}: X \to \mathbf{R}^n$ where

$$\boldsymbol{\psi}(x) = (\tilde{h}_1(x), \ldots, \tilde{h}_n(x)).$$

Also a infinite H has a similar interpretation.

For simplicity, assume H is closed w.r.t. negation (if $h \in H$ then $-h \in H$). This means we do not need to worry about negative w_i. Denoting the feature space by F, we can now write the dual problem as

Optimisation 5.12: Variables $\mu \in \mathbf{R}$, $\boldsymbol{w} \in F$

maximise	μ
subject to	$y_i \langle \boldsymbol{w}, \boldsymbol{\psi}(x_i) \rangle \geq \mu$ for $i = 1, \ldots, m$
	$\|\boldsymbol{w}\|_1 = 1.$

This is similar to hard margin SVM, except that we constrain the 1-norm and not 2-norm. (Also the bias term b is missing from this formulation.)

Recall the geometric dual norm interpretation of margin:

Theorem 5.13: Let $1/p + 1/q = 1$. Let $B = \{ \boldsymbol{x} \in \mathbf{R}^n \mid \boldsymbol{w} \cdot \boldsymbol{x} = 0 \}$ be a hyperplane. Then for any $\boldsymbol{x} \in \mathbf{R}^n$ we have

$$\frac{|\langle \boldsymbol{w}, \boldsymbol{x} \rangle|}{\|\boldsymbol{w}\|_q} = \|\boldsymbol{x} - B\|_p$$

where $\|\boldsymbol{x} - B\|_p = \min \{ \|\boldsymbol{x} - \boldsymbol{z}\|_p \mid \boldsymbol{z} \in B \}$.

Hence, boosting is maximising an ∞-norm distance from the separating hyperplane, whereas SVM maximises a 2-norm distance.

Algorithmically there is of course the major difference that the feature map we defined for boosting does not usually have a nice kernel. (But it is quite possible to run boosting with $H = \{ k(\cdot, x_i) \mid i = 1, \ldots, m \}$ for some popular kernel k.)

AdaBoost and similar procedures can be interpreted as gradient based methods for maximising a margin-based loss function.

Suppose we wish to minimise some additive loss of type

$$G(f, S) = \sum_{i=1}^{m} g(y_i f(x_i)).$$

For example, we have claimed that AdaBoost is based on $g(z) = e^{-z}$, and mentioned LogitBoost based on $g(z) = \log_2(1 + e^{-z})$.

We restrict our solutions to form $f(x) = \sum_{j=1}^{n} w_j \tilde{h}_j(x)$ where for notational convenience we assume a finite hypothesis class $H = \{\tilde{h}_1, \ldots, \tilde{h}_n\}$. We interpret w_j as the jth coordinate of f. Consider now the derivative

$$\begin{aligned} \left.\frac{\partial L(f + \alpha\tilde{h}_j, S)}{\partial \alpha}\right|_{\alpha=0} &= \left.\frac{\partial}{\partial \alpha} \sum_{i=1}^{m} g(y_i(f(x_i) + \alpha\tilde{h}_j(x_i)))\right|_{\alpha=0} \\ &= \sum_{i=1}^{m} g'(y_i f(x_i)) y_i \tilde{h}_j(x_i). \end{aligned}$$

If we choose $d_i = -g'(y_i f(x_i))$ (properly normalised), this is exactly $-\hat{\gamma}(\tilde{h}_j, S, \boldsymbol{d})$.

In other words, maximising the edge w.r.t. this $\boldsymbol{d}$ corresponds to finding the steepest descent direction for G.

We get the following algorithm for minimising G.

Algorithm 5.14 (Generic margin optimisation):

1. Initialise $f_0 = 0$.
2. Repeat for $t = 1, 2, \ldots$:
   (a) $d_i^{(t)} = -g'(y_i f_{t-1}(x_i))$ for $i = 1, \ldots, m$.
   (b) $h_t := A(S, \boldsymbol{d}^{(t)})$
   (c) $\alpha_t = \arg\min_\alpha G(f_{t-1} + \alpha h_t, S)$
   (d) $f_t := f_{t-1} + \alpha_t h_t$

For simplicity, we ignored the normalisation of $\boldsymbol{d}$.

In the special case that A always finds the weak hypothesis with maximum edge, the algorithm performs steepest descent in the direction of the best single coordinate.

If A does not maximise the edge, we still hope that the distribution $\boldsymbol{d}$ points it to roughly the right direction.

For AdaBoost, $g(z) = e^{-z}$ so $d_i \propto \exp(-y_i f(x_i))$.

For LogitBoost, $g(z) = \log_2(1 + e^{-z})$ so $d_i \propto 1/(1 + \exp(y_i f(x_i)))$. This is more moderate than AdaBoost for very negative margins.

Instead of heuristics, it is also possible to maximise the margin (Optimisation 5.11) directly using Linear Programming.

Since maximising the (hard) margin may not be a good idea in noisy problems, we introduce slack variables like with SVM.

Optimisation 5.15: Variables $\mu \in \mathbf{R}$, $\boldsymbol{w} \in \mathbf{R}^n$, $\boldsymbol{\xi} \in \mathbf{R}^m$

$$\begin{array}{ll} \textbf{maximise} & \mu - C \sum_{i=1}^{m} \xi_i \\ \textbf{subject to} & y_i \left(\sum_{j=1}^{n} w_j \tilde{h}_j(x_i) \right) \geq \mu - \xi_i \text{ for } i = 1, \ldots, m \\ & \xi_i \geq 0,\ w_j \geq 0,\ \sum_{j=1}^{n} w_j = 1. \end{array}$$

The problem is very similar to the 1-norm soft margin SVM. In particular, we could choose $C = 1/(\nu m)$ to get a more intuitive parameterisation.

Solving Optimisation 5.15 directly in terms of the primal variables is of course not feasible since n can be huge.

The dual of Optimisation 5.15 is the following.

Optimisation 5.16: Variables $\gamma \in \mathbf{R}$, $\boldsymbol{d} \in \mathbf{R}^m$

minimise	γ
subject to	$\sum_{i=1}^{m} d_i y_i \tilde{h}_j(x_i) \leq \gamma$ for $j = 1, \ldots, n$
	$0 \leq d_i \leq C$, $\sum_{i=1}^{m} d_i = 1$.

The difference to Optimisation 5.10 is the box constraint $0 \leq d_i \leq C$ like with SVMs.

Notice that a weak learner A can be considered an oracle that given tentative $\boldsymbol{d}$ and γ tries to find $\tilde{h} \in H$ such that the constraint $\sum_{i=1}^{m} d_i y_i \tilde{h}(x_i) \leq \gamma$ is violated.

Optimisation 5.16, and consequently 5.15, can be solved iteratively by a method known as column generation in linear programming.

For $\tilde{h} \in H$, call the constraint $\sum_{i=1}^{m} d_i y_i \tilde{h}(x_i) \leq \gamma$ the constraint for $\tilde{h}$.

1. Choose some small initial $H' \subset H$.

2. Solve Optimisation 5.16 but ignore the constraints for $\tilde{h} \notin H'$.

3. Use the weak learner A to find $\tilde{h} \in H - H'$ such that the current solution violates the constraint for $\tilde{h}$. If no such $\tilde{h}$ was found, go to Step 5.

4. Take $H' := H' \cup \{\tilde{h}\}$ and go to Step 2.

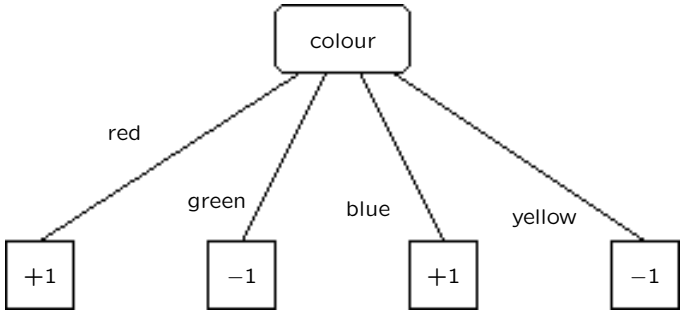
5. Recover the primal solution $\boldsymbol{w}$ from the Lagrange coefficients for the found dual solution.

This works well in practice, even if the weak learner does not always find the "best" $\tilde{h}$.

5.5 Other issues

The choice of weak learner is of course important, but not much is known to guide it.

Most popular weak learners are general decision tree algorithms (such as C4.5) and algorithms that use decision stumps, *i.e.*, one-level decision trees. The advantage of stumps is that an optimal weak hypothesis can easily be found, and such simple hypotheses are well suited for boosting.



Decision stump: if attribute **colour** has value *red* or *blue*, then class is +1; if *colour* has value *green* or *yellow*, then class is +1.

Also other learning algorithms can be used, such as neural networks, but with rich hypothesis classes one must beware overfitting.

Some issues we have not addressed:

Statistical:

What are the minimum requirements for the weak learner?
When does such a weak learner exist?
Is the procedure consistent, *i.e.*, as $m \to \infty$ does the hypothesis approach optimal with probability one?
Can we get better error bounds?

Algorithmic:

What kind of weak learner is sufficient for the procedures to converge?
What other optimisation procedures exist?

Practical:

What are good weak learners?
How to deal with multiclass, regression etc.?
How to deal with large amounts of noise?

Some of these issues are more or less understood, many remain open research problems.

6. Final remarks

This course has not been intended as a balanced view of the whole field of machine learning.

We have selected a group of topics that could be fitted together into a (somewhat) coherent whole. The selection has been biased by the personal preferences of the lecturer.

Our emphasis has been theoretical: what are the underlying principles, and assumptions, that allow extrapolating from past experience into the future.

We have viewed machine learning as a fairly well-defined, narrow technical problem. The kind of general learning methods that would be needed for true artificial intelligence are way beyond current state of the art.

We have studies only classification problems. Among other topics, we have ignored

- regression, multiclass problems
- relational learning, learning in structured domains
- unsupervised learning: clustering, dimensionality reduction, novelty detection
- reinforcement learning, active learning
- theoretical models such as query learning and learning in the limit

Within classification, we have omitted such established techniques as decision trees.

We have largely ignored algorithmic and implementation issues.

On a more general level, we have ignored the context in which (machine) learning takes place: where does the data come from, how is it preprocessed, how are the results used, what do the users actually need?

For a succesful application, these questions can be **much more important** than which algorithm to use. Consultation with domain specialists is needed.

What we *have* seen is a collection of theoretically-motivated yet practically useful algorithms:

- online linear classifiers,
- support vector machines and
- boosting.

SVMs and boosting are part of practitioners' standard tool kit. Online algorithms are not yet so established.

The End