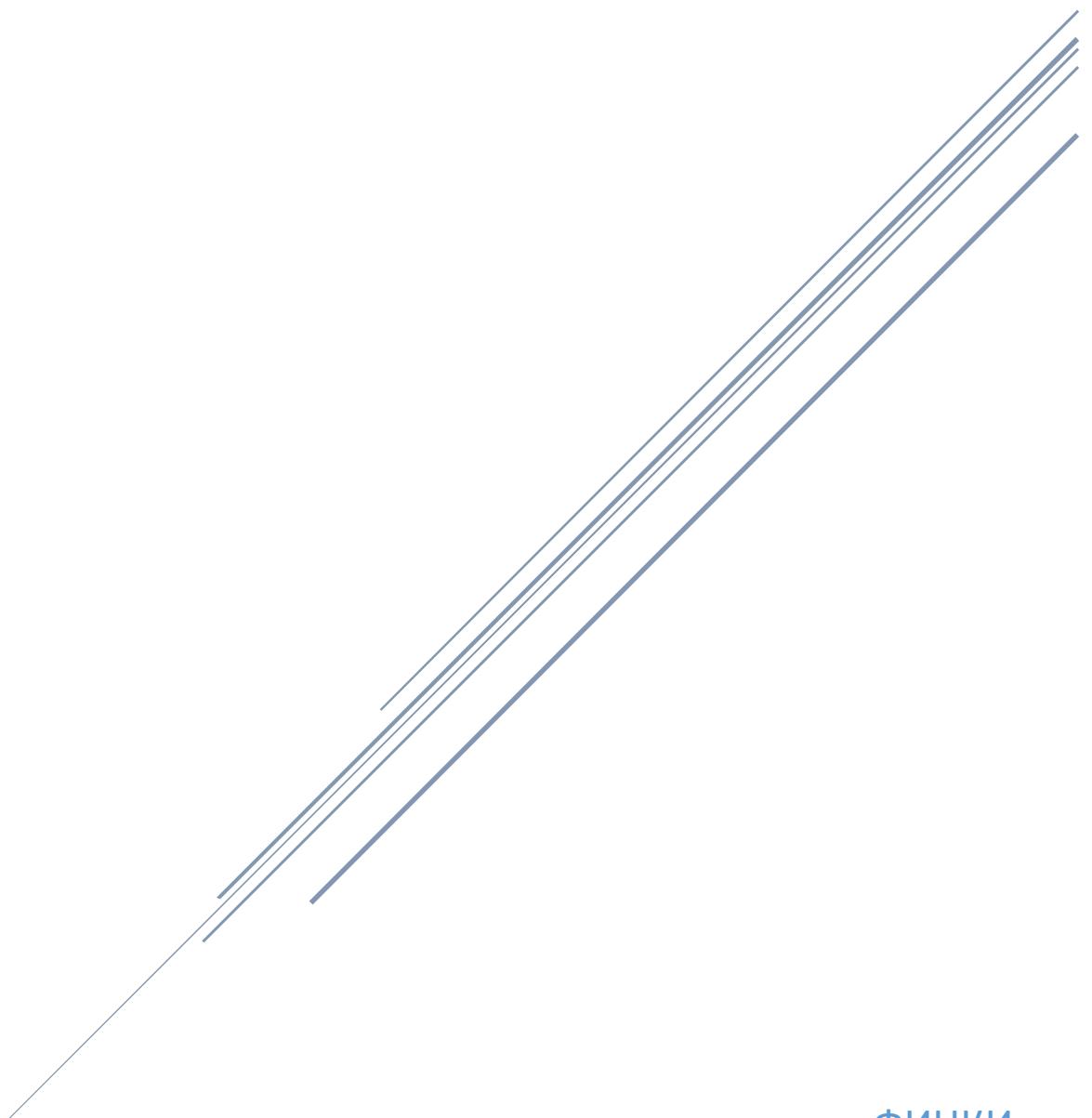


МАШИНСКО УЧЕЊЕ

Обработени предавања - Kevin P. Murphy - Machine Learning A Probabilistic Perspective



ФИНКИ
2019-2020

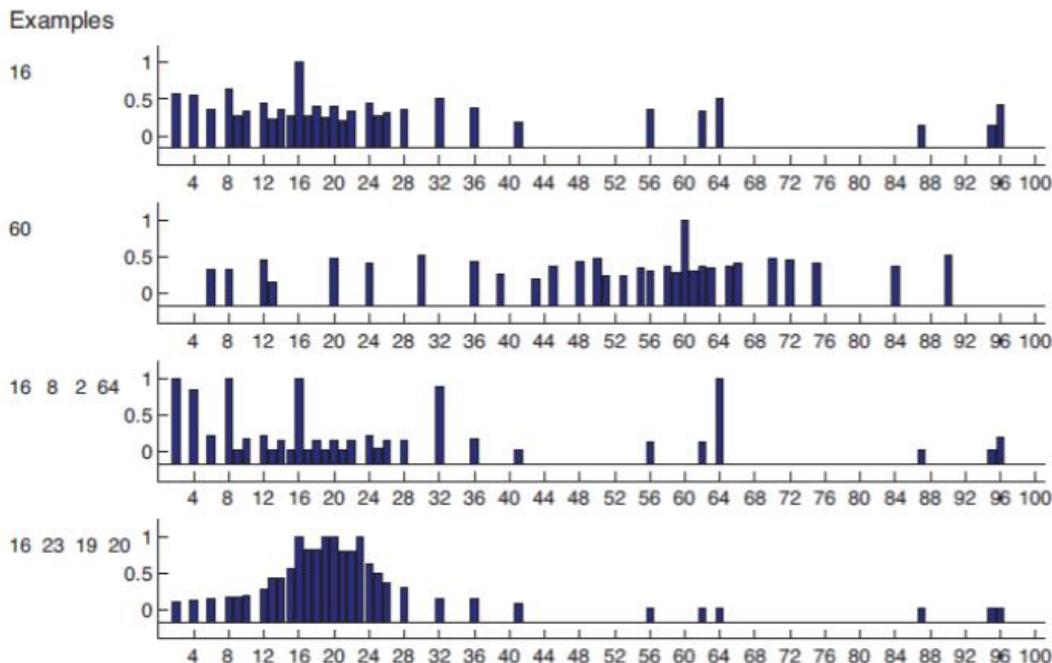
Generative Models for Discrete Data/ Генеративни модели за дискретни податоци

Целта на ова е дека користиме Бајесово правило за да класифицираме, значи пример ако треба да одредиме веројатноста на некој нов вектор да припаѓа на класа С. Во ова предавање ќе се фокусираме на дискретни податоци, целта е освен оваа предиктивна дистрибуција да одредиме кој се параметрите тета на моделот. Прво да разјасниме која е разликата помеѓу генеративни и дискриминативни модели. Во случајот двета се класификациони модели но работат на различен принцип. Значи генеративните модели воглавно целта е да дознае што е можно повеќе информации за податоците и во однос на нив да прави класификација. Тука ја моделираме веројатноста на класата и веројатноста на X , доколку ја знаеме класата, и со бајесово правило го користиме тоа за да добиеме која е веројатноста тој вектор X да припаѓа на дадената класа. Додека дискриминативните модели се фокусирани директно на оваа условна веројатност каде што целта не е да се дознае што повеќе информации од секоја класа туку кој се разликите помеѓу податоците во дадената класа.

Пример татко има две деца, Дете А и Дете Б. Детето А има специјален карактер каде што може да се учи во длабочина, суштински. Детето Б има специјален карактер каде што тој може да ги научи разликите помеѓу тоа што има видено. Еден ден таткото ги носи децата во зоолошка, зоолошката е многу мала и има два типа на животни, лав и слон. Откако излегле од зоолошка татко им покажал слика и ги прашал „Дали ова животно е лав или слон?“. Детето А, одеднаш нацрта слика од лавот и слонот на парче хартија базирано на тоа што видол во зоолошката, ги споредил двете според најдобро поклонување на атрибути на слонот и лавот што ги виде и атрибутите на животното на сликата, тоа дете одговорило дека е лав. Додека детето Б ги споредило само разликите па тоа што има најмалку разлики тоа е најверојатно животното, кое на крај заклучило дека е лав. Во Машинско Учење ние ова го нарекуваме Генеративен модел (дете А) и Дискриминативен модел (дете Б).

Во ова предавање ќе работиме генеративни модели. Следно е Бајесов концепт на учење, ова е слично како ние самите учиме. Значи како деца не' научиле што е куче, најверојатно не сме имале прилика каде луѓето од околу ни кажуваат ова не е куче, туку имаме само позитивни примери каде што добиваме информации за ентитети кои што се кучиња. За разлика од стандардна класификација каде што имаме примери од позитивната класа и од неативната класа, во случајов работиме само со позитивни примери од позитивната класа. Целта е да се најде некоја функција која што ќе ни има вредност 1 доколку X припаѓа на таа класа и вредност 0 доколку X не припаѓа на таа класа. За да го проучиме овој модел ќе проучиме една игра Number game, правилата се многу едноставни, имаме некој концепт каде што доаѓаат некои броеви од некоја класа и треба да најдеме кое е правилото како што се појавуваат броевите, односно на крај треба да се класифицираат дали за некоја нова вредност на X дали припаѓа на тоа правило или не. За поедноствност, ќе имаме броеви од 1 до 100, и сега првично ни доаѓа бројот 16 и треба да одлучиме кое е правилото од кое што може да дојде бројот 16. Има многу различни правила прво рандом

избран 16 од 1 - 100, па броеви 2 на некој степен, парни броеви итн итн. И сега кое од овие правила најдобро ги опишува податоците. Само со еден примерок баш не можеме тоа да го заклучиме но како што има повеќе примероци, толку повеќе информации добиваме за податоците и толку подобро може да го истренираме класификаторот односно целиот модел. Заклучивме дека со зголемување на бројот на податоците стануваме посигурни и имаме помалаче неизвесност во новите податоци. Исто така треба да се најде за нов број што би внесиле доколку имаме условно D, а D се податоците кои што се до сега стигнати да кажеме дали припаѓа на тоа правило или не припаѓа на тоа правило. Сега доколку на 16 ги додадеме броевите 8, 2, 24, множеството хипотези станува малце помало односно веќе повеојатно е дека има некое правило каде што се гледа 2 на некој степен отколку за разлика од некои други хипотези кои што претходно ние ги претпоставивме, и тоа што инститтивно помисливме сите на тоа е пример за индукција ☺. За различно податочно множество како ова добиваме различни индукции т.е претпоставки, $D = \{16, 23, 19, 20\}$, пример тука немаме некој индукција како претходно туку претпоставуваме дека е некоја дистрибуција што е околу овие броеви, пример има поголема веројатност да се појави бројот 18,17 во однос на бројот 90.



Ова се податоци од експеримент кој што го направиле, каде што луѓето сами погодувале кое е правилото кое ја опишува играта. За 16 можеме да видиме дека има броеви близку 16, за бројот 60 нема некој слични правила како за 16 и затоа вовлавно податоците се расфрлани и околу 60, броеви што завршуваат со 0, парни итн итн. А доколку имаме повеќе информации имаме повеќе некои пикови на дистрибуцијата односно можеме да

видиме на цртежот дека најверојатно правилото е некој експонент на 2 или на 4 во случајов 2 бидејќи го имаме бројот 2 во дата сетот и заради тоа ги земаме 4, 8, 12 итн итн, доколку гледаме во последниот график не можеме да извлечеме некое правило како погоре туку гледаме дека е некаква дистрибуција околу каде податоците се околу 16 – 20. Сега се поставува прашањето како ова да го симулираме во машина односно да направиме некој модел, користеќи машинско учење и кој би можел да претпоставува која е дистрибуцијата како што луѓето на експериментот претпоставиле. Целта е, имаме множество на хипотези и треба да најдеме која е веројатноста за секоја хипотеза, па доколку имаме множество хипотези, можеме да пресметаме за нов вектор X која е веројатноста да припаѓа на таа хипотеза или не. Е сега за тоа да го моделираме Бајесово правило каде што ќе имаме Likelihood, Prior и Posterior веројатност за секоја хипотеза и после тоа ќе го искористиме за предикција.

Likelihood

Кога ги видовме 16, 8, 4 природно ни дојде дека се работи за правило кое е некој степен на 2 и бевме поуверени дека е тоа, отколку правило на парни броеви. Тука главниот концепт е дека доколку хипотезата е стварно парни броеви, но која е веројатноста да се појават специфично броеви кои се степен на 2, таа прилично мала и затоа сакаме да избегнеме вакви сомнителни случајности (**suspicious coincidences**). За ова да го направиме имаме за секоја хипотеза влечење униформно и за да добиеме likelihood на тој вектор да се појави како нов податок ја користиме следната формула:

$$p(D|h) = \left[\frac{1}{\text{size}(h)} \right]^N = \left[\frac{1}{|h|} \right]^N$$

Значи доколку влечеме униформно од таа дистрибуција веројатноста да се појави некој податок од таа дистрибуција е реципрочно во однос на големината на дистрибуцијата, пример доколку направиме претпоставка овие се добиени од парни броеви, веројатноста да е да е точно 16 извлечен е една 1/50, доколку направиме претпоставка дека е некој број што е степен на 2 имаме само 6 броја помеѓу 1 и 100 кои се степен на 2 па веројатноста е 1/6. Потоа таа веројатност ја степенуваме со бројот на податоци на дата сетот. Заклучивме дека со likelihood може да моделираме т.е да претпоставиме и да избегнеме suspicious coincidence.

Prior

Концептот сите броеви на степен 2 освен 32, според likelihood претходно дефиниран е поверијатен отколку сите броеви на степен два. Поради тоа што ова правило содржи само 5 членови значи истото ако го пресметаме ќе биде $(1/5)^4$ односно би било поголем, но вакво правило не е интуитивно за нас и нема некој логика баш, и заради тоа сакаме да најдеме некое приор знаење на вакви правила што баш немаат логика. И тоа е дел од бајесово размислување. Значи проблемот што го критикуваат многу луѓе на Бајесова веројатност во

однос на фреквенитичка веројатност е баш овој приор затоа што јас можам да имам различен приор од некој друг и тоа може да да ме доведи мене до различни заклучоци од другите. Но пак тоа многу ни помага во моделирање. Пример доколку имаме податоци кои што се 1200, 1500, 900 и 1400 и ни имаат кажано дека се добиени по некое аритметичко правило пологично е дека се добиени по правилото некој број поделен со 100 и можеме да дадеме таков приор каде што бројот 400 има многу голема веројатност да се појави но бројот 1183 има мала веројатност да се падни. Но ако знаеме пример дека овие податоци се добиени од ниво на шеќер тогаш 400 многу е неверојатно да се појави бидејќи човекот би бил мртов очигледно а 1183 е многу повеќетен да се падни. Целта е да воведиме некое експертско знаење во самата облас во зависност што моделираме, кое знаење експертско може да ни помогне во сите претпоставки натаму. Во оваа игра имаме првично 30 хипотези како прости броеви, броеви кои завршуваат на 9 итн на сите тие ќе им додадеме некоја приорна веројатност притоа веројатноста на неинтуитивни концепти како што спомнавме пример 2 на степен освен 32 ќе има мала приор веројатност додека овде се додава повеќе веројатност на парни броеви.

Posterior

И финалното што го моделираме за да ги опишеме овие податоци е постериорот. Како што знаеме по Бајесово правило се добива како множење на likelihood со prior, поделено со множеството на сите хипотези. Поради тоа што овој член ни е еднаков на било која хипотеза што ја разгледуваме може да тргнеме со горниот дел од формулата. За секоја хипотеза имаме приор да се појави хипотезата и likelihood со кој се има појавено таа хипотеза и може да се пресмета постериорот.

$$p(h|D) = \frac{p(D|h)p(h)}{\sum_{h' \in \mathcal{H}} p(D, h')}$$

Како што почнуваме без податоци постериорот ни е еднаков на приорот, тоа е знаењето кое што во моментот го имаме и кое што го очекуваме. Додека како што расти бројот на податоците влијанието на приорот се намалува каде што на крај имаме премногу податоци приорот не е веќе ни битен, битно е да се фокусираме на хипотезата која што најмногу ги опишува податоците/моделот.

$$\delta_x(A) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases} \quad p(h|D) \rightarrow \delta_{\tilde{h}_{MAP}}(h)$$

Ова објаснува дека делта функцијата има вредност 1 доколку нешто важи во случајот хипотезата, и има вредност 0. Иначе ова кажува дека веројатноста да се појави H за податоци D , веќе имаме многу голем број податоци и ќе издвоиме една хипотеза тогаш ќе е е 1 веројатност сите други ќе се нули.

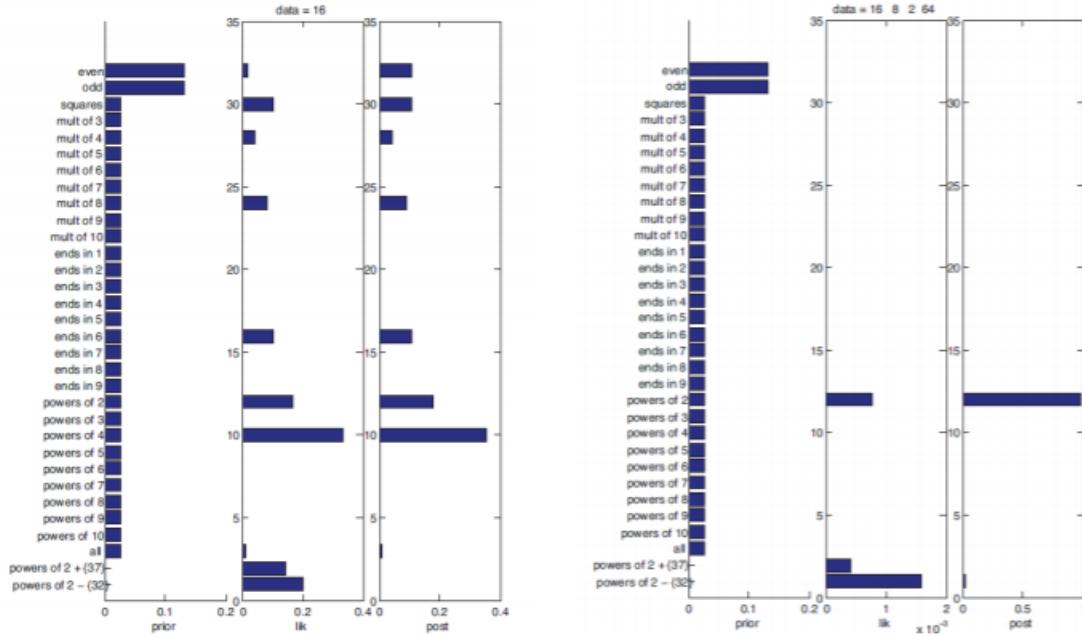
MAP estimate имаме некоја дистрибуција на хипотези и секоја хипотеза од почеток има некоја веројатност да се појави. На крај после многу податоци одлучуваме дека веќе знаеме таа е таа хипотеза и се зема само таа што има најголем апостериор веројатност.

$$\tilde{h}^{MAP} = \operatorname{argmax}_h p(D|h)p(h) = \operatorname{argmax}_h [\log p(D|h) + \log p(h)]$$

Likelihood зависи експоненцијално зависи од D, односно како што се зголемува бројот на податоци толку е поголем експонентот. МАР оценуваат на крај конвергира кон МЛЕ, односно доколку имаме што е можно повеќе податоци тогаш приорот е небитен додека со многу малку податоци и приорот ни дава некое значење.

$$\tilde{h}^{mle} \triangleq \operatorname{argmax}_h p(D|h) = \operatorname{argmax}_h \log p(D|h)$$

Доколку хипотезата од почеток ги дефинира податоците, и ја има во hypothesis space тогаш и двата овие оценувачи конвергираат кон таа хипотеза. Тоа кажува дека тие се конзистентни оценувачи односно нема да очекуваме некои грешки во предвидувањто. Исто така како што се зголемува бројот на податоци очекуваме дека на крај ќе ја добиеме баш правилото кое ги опишува податоците односно во случајов ако се соберат финално 100 податоци веќе знаеме која е хипотезата која што ги опишува податоците.



Сега ја имаме оваа слика горе, индивидуално ќе ги гледаме податоците, доколку податокот е 16 ја имаме првата слика каде за парни и непарни даваме поголем приор, lk (likelihood) кое што се добива од податоците, некои концепти за 16 имаат делумно голем lk а некои

имаат 0 како за непарни и во зависност кое е правилото пример степен на 4 има најголем lk. И во однос на тоа сакаме да одредиме која е постериор веројатноста што ја добиваме само со множење на овие два делови на приорот и lk, воочуваме дека имаме постприор да хипотезата е парни броеви но најголема веројатност има за хипотезата да е *степен на 4*. Ако ја гледаме другата слика каде податоците се 16, 8, 2, 64, најголем е постприорот да е некој степен на 2 а другите ги нема или се мн мали. Каде lk долу степени на 2 освен 32 има многу голем lk заради тоа што се совпаѓа со податоците најмногу но има неприродна интуиција за нас и има приор 0 па затоа постприорот е многу мал. Сега треба да одредиме за нов X дали припаѓа на класата или не имаме постериор предиктивна дистрибуција *Posterior predictive distribution*. За тоа ни треба оваа формула:

$$p(\tilde{x} \in C|D) = \sum_h p(y=1|\tilde{x}, h)p(h|D)$$

Која е веројатноста да припаѓа на класата C за дадените податоци, по веројатноста да важи хипотезата и веројатноста да припаѓа на класата доколку важи хипотезата. Тука како што гледаме немаме избрано специфична хипотеза односно имаме множество на хипотези, и понатаму ќе се спомне бидејќи имаме сума и разгледуваме во однос на хипотези за ова во случајов за секој од овие хипотези што претходно сме ги дефинирале ја пресметуваме веројатноста и на крај добиваме некоја сума која претпоставуваме дали новиот X би припаѓал на класата или не. Тука значи немаме избрано специфична хипотеза туку имаме множество на хипотези кои ги користиме во предикција за следен вектор. Доколку имаме мали податоци, бидејќи оваа дистрибуција е со широка варијанса и како што се зголемуваат податоците со MAP estimate веќе одлучуваме која хипотеза е точна според податоците и неа ја користиме. За да се поедностави моделот да не ги гледаме сите хипотези можи да се одлучиме да користиме за почеток пред да ги добиеме тие податоци да избереме дека имаме една најдобра хипотеза и таа да ја користиме и тоа се вика **plug-in approximation**. Во случај на, еден од оценувачите MAP/MLE одлучуваме дека таа е најдобрата хипотеза и само таа ќе ја користиме за апроксимација.

**Имаме MAP оценувач, тука на некој начин моделираме, пример имаме 10 хипотези сите имаат некоја веројатност да се појави, MAP оценувачот работи со тоа што ја земаме таа највисока веројатност т.е мода додека MLE како што учевме по веројатност, ги гледаме податоците и гледаме која е веројатноста да се појават и во однос на тоа оценуваме со помош на логаритмирање и извод, каде изводот ни дава екстреми на функцијата и добиваме која е максимум вредноста на даден оценувач.

Оваа **plug-in approximation** се користи многу бидејќи е многу едноставна но како што очекуваме како што имаме една хипотеза веројатноста на крај веројатноста дистрибуција на X ќе е прилично со мала варијанса односно ќе има мал пик во однос на претходното кое користи повеќе хипотези.

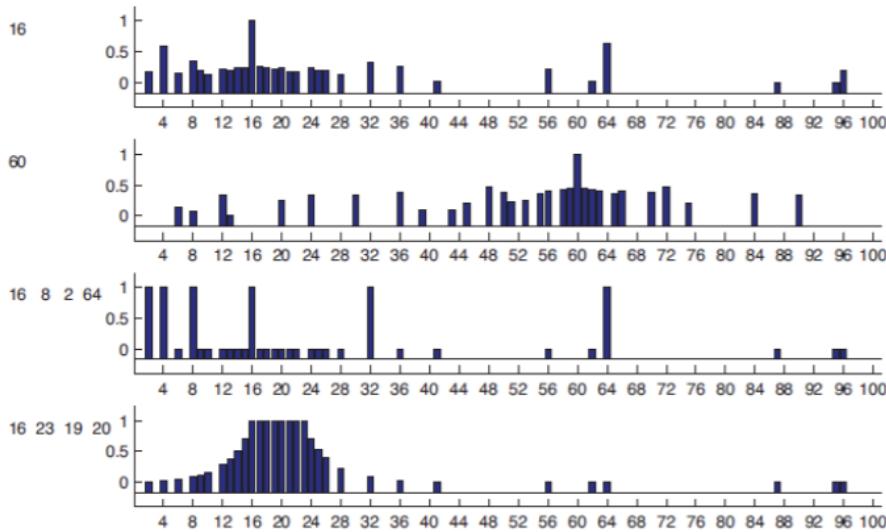
Споредба на Bayes model averaging и на Plug-in approximation

Каде дека е некој степен на 4 и ја избирааме само таа хипотеза и за секое ново X го класифицираме само во однос на таа хипотеза. Поради тоа што имаме многу малце податоци т.е $D=16$ најверојатната хипотеза за ова би била дека е очекуваме дека само од еден податок можеме да направиме таква одлука и ова пример за *overfitting* од мал број на податоци одлучиваме нешто многу екстремно поради тоа дури други идни предвидувања би ги довело до голема грешка. Додека се додаваат бројот на податоци на множеството, па хипотезата се проширува на некој начин, првично почнавме со степен на 4 што е прилично мало множество, но како што доаѓаат податоците хипотезата која најмногу ги опишуваме е степен на 2 и гледаме тука дека како што се зголемува бројот на податоци кај **plug-in approximation**, се зголемува дистрибуцијата на хипотезата односно хипотезата опфаќа повеќе примери. Од друга страна ако користиме Бајесовиот пристап за $D=16$, како што видовме има 20 хипотези кои имаат минимална веројатност и ги користиме сите, и заради тоа на почеток веројатноста за предвидување е прилично широка и како што доаѓаат податоците се намалува, односно дел од тие постериор веројатности се изгубија и се намалува множеството на хипотези и заради тоа како што добиваме податоци во Бајесовиот пристап рангот на хипотези се намалува и поради тоа и предиктивната дистрибуција ни е помала, така да од почеток имаме имаат многу различна веројатностна предиктивност дистрибуција овие два модели но како што додаваме податоци двете конвергираат кон исто.

A more complex prior

Овој приор е покомплексен за разлика од претходно каде што само додававме хипотези и целта му била на некој начин да го најде правилото како самите ние луѓе размислеваме и би апроксимирале вакви хипотези, така да приор веројатноста сега се дели на два дела каде едниот дел е во однос на правила како што беа сите погоре наведени, додека другиот е интервал делот, односно ако видиме 20 логично е да очекуваме некој број околу 20, и тука имаме еден параметар кој дефинира дали повеќе во концепти или правила или повеќе на интервал некој и добиени се резултати прилично исти како и лутето што ги дадоле.

Examples

**The beta-binomial model**

Целта тута е разликата што до сега ја гледавме имавме множество на хипотези каде што секоја хипотеза е различна и имаме дискретно множество но доколку сакаме ова да е реално множество имаме покомплицирани математики. Тука многу работи се исти само се прави промена од суми во интеграли. Проблемот што ќе го разгледаме е, фрламе паричка и треба да го разгледаме која е веројатноста да е глава и петка. Likelihood – фрлање на паричка има бернулиева распределба, или е петка или е глава.

$$p(D|\theta) = \theta^{N_1} (1 - \theta)^{N_0}$$

И сега треба да апроксимираме да го најдеме параметарот тета и после да се најде за ново фрлање во однос на тета која е веројатноста да е глава или петка. N_1 & N_2 се броеви колку пати падnilo глава или петка, доколку не ни се битни сите податоци туку ни се битни туку само $s(D)$ кажуваме дека е $s(D)$ is a sufficient statistic. Сега истот ќе го правиме но за разлика од претходно само за едно фрлање сега ќе го правиме за к изведби и имаме биномна распределба.

$$\text{Bin}(k|n, \theta) \triangleq \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

Параметарот тета не менува ништо во формулата и може да го запоставиме бидејќи е ист за сите класи. Така да на крај моделот ни е ист како кај бернулиева распределба.

The beta-binomial model – Prior

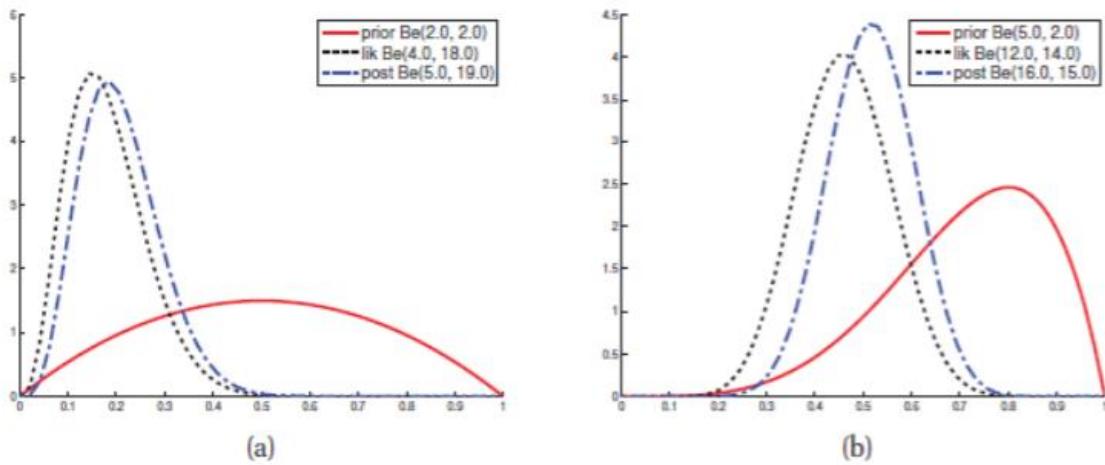
Првично приорот ни е на интервал од [0,1]. Ова е приор на параметарот тета, ако тета ни е веројатноста на паѓање глава односно тоа што очекуваме. Во претходниот случај имавме хипотези и имавме X , првично имаме податоци некои што се паднати како што биле 16,18,32 според нив требаше да најдеме која беше хипотезата што го описуваше моделот и во однос на таа хипотеза да одлучиме за следно фрлање дали припаѓа на класата или не. Тука имаме фрлање на паричка, веќе ги имаме и податоците, првично треба да го најдеме параметарот на моделот, првично знаеме дека има биномна распределба знаме дека параметарот е тета па аналогно на барање на хипотеза, претходно прво баравме хипотеза а сега го бараме кој е параметарот тета и откако го имаме параметарот тета, веројатноста да следно фрлање е глава или петка го пишуваме истото од претходно веројатноста да е нов број и да припаѓа на таа класа/хипотеза. За овој модел користиме којнугиран приор, тоа ни е приорот кој што има иста форма како likelihood. За да пресметаме постприорот е многу полесно ако имаат иста форма како приорот и поради тоа во случајов само за да добиеме постериор само ги множиме т.е ги додаваме експонентите.

$$p(\theta|D) \propto p(D|\theta)p(\theta) = \theta^{N_1}(1-\theta)^{N_0}\theta^{\gamma_1}(1-\theta)^{\gamma_2} = \\ \theta^{N_1+\gamma_1}(1-\theta)^{N_0+\gamma_2}$$

Приор за бернулиева е бета дитрибуцијата таа има и плус некој параметри кои се нормализациони константи но поради тоа што не зависат од самиот параметар не зависат од моделот и затоа тие може скроз да ги знамариме. Имаме

$$\text{Beta}(\theta|a, b) \propto \theta^{(a)}(1-\theta)^{b}$$

Исто така треба да се каже дека параметрите на приорот се **хипер-параметри**, може да ги сетираме по ред за да го енкодираме по нашето приорно вервање. Ако паричката е бајес односно поголема е веројатноста да падне глава од одколку петка можиме од почеток параметрите да ги подесиме на некој редослед кој што би го описане тоа и со тек на време со податоците ќе се докаже дали е вистина или не. Тоа е целата поента на приор од почеток претпоставуваме нешто и со тек на време во зависност од податоците како се движат постериорот ни покажува дали се совпаѓаат со приорот и како се зголемуваат податоците одиме кон Likelihood-от него кон приорот. **Приорот е независен од податоците**



Црвената линија е приорот, црна lk а сина posterior, доколку имаме не информиран приор што е прилично слаб posterior-от ќе е многу сличен на lk, додека ставиме појак приор че видиме дека постериорот се приближува кон него односно ни требаат повеќе податоци за да стигнеме до реалната слика.

Пример за појасно, имаме фрлање на праичка, имаме параметар кој кажува колку е фер паричката, кога ќе ја фрлиме колку е веројатноста да се погоди глава ни е параметарот тета. Првично сега ние треба да одлучиме колку ни е тој параметар, доколку тој што ја фрла паричката му верувам и знам дека нема да манипулира со паричката треба да ставам некоја приор распределба на параметарот. На самото тета да правам приор дистрибуција каде што веројатноста тета = 0,5 е најголема поради тоа што знам дека човекот што ја фрла паричката нема да лаже, а ако тета = 0,7 што е пример на нефер паричка најверојатно ќе падне глава или петка. Поентата е земаме пример, за параметарот да е 0,5 гледаме колкав е likelihood, ако од 5 пати фрлање 3 пати падне глава и тоа го пресметуваме за сите параметри и одлучуваме колку е тета значи имаме нова дистрибуција што ни е тета доколку ги имаме видено податоците.? Прво земаме паричката да е фер, тета = 0,5 имаме податоци каде што 3 пати паднало глава а 2 пати петка, веројатноста да се случило ова за тета = 0,5 ни е $(1/2)^3 * (1/2)^2$ и ги множиме и даваме веројатност p=0,6, сега земаме тета=0,7 тогаш веројатноста p=0,1 помала од претходното во однос на параметарот веројатноста ова да се деси е $(0,7)^3 * (0,3)^2$ ги множиме овие и добиваме нова вредност претпостериор веројатноста и ова го правиме за сите точки од приор дистрибуцијата.

The beta-binomial model - Posterior mean and mode

- The MAP estimate is given by

$$\hat{\theta}_{MAP} = \frac{a + N_1 - 1}{a + b + N - 2}$$

- If we use a uniform prior, then the MAP estimate reduces to the MLE, which is just the empirical fraction of heads:

$$\hat{\theta}_{MLE} = \frac{N_1}{N}$$

- By contrast, the posterior mean is given by,

$$\bar{\theta} = \frac{a + N_1}{a + b + N}$$

The beta-binomial model - Posterior variance

Како што пресметавме точкасти оценувачи на тета добро е да ги знаеме кој се најдобрите, односно која е варијансата т.е колку можеме да им веруваме на параметрите во претходните примери имавме одбрано тета како најдобар параметар, и сега сакаме да види колку може да им веруваме на тоа тета, затоа пресметуваме варијанса и стандардна девијација.

- The mean and mode are point estimates, but it is useful to know how much we can trust them. The variance of the posterior is one way to measure this. The variance of the Beta posterior is given by

$$var[\theta|D] = \frac{(a + N_1)(b + N_0)}{(a + N_1 + b + N_0)^2(a + N_1 + b + N_0 + 1)}$$

- We can simplify this formidable expression in the case that $N \gg a, b$, to get

$$var[\theta|D] \approx \frac{N_1 N_0}{NNN} = \frac{\hat{\theta}(1 - \hat{\theta})}{N}$$

Hence the “error bar” in our estimate, is given by

$$\sigma = \sqrt{var[\theta|D]} \approx \sqrt{\frac{\hat{\theta}(1 - \hat{\theta})}{N}}$$

- We see that the uncertainty goes down at a rate of $1/\sqrt{N}$. Note, however, that the uncertainty (variance) is maximized when $\hat{\theta} = 0.5$. and is minimized when $\hat{\theta}$ is close to 0 or 1.

Оваа варијанса е максимизирана кога тета = $\frac{1}{2}$, полесно е да сме сигурни дека тета = 1 отколку да сме сигурни дека не е бајес паричката. Откако ќе ги пресметаме параметрите од

тета моделот следно што треба да се направи е да одлучиме во следното фрлање која е веројатноста да се појави глава или петка, поради тоа што тоа е едноставен модел, ако тета е 0,5 и следната веројатност =0,5, ако немаме точкаст оценувач користиме дистрибуција на тета место да избериме најдобар тета, бидејќи не можеме да сме сигурни дека тоа е скроз вистина, како што праевме пре за различни тета тестираме која е веројатноста и се зема просек.

- ▶ So far, we have been focusing on inference of the unknown parameter(s). Let us now turn our attention to prediction of future observable data. Consider predicting the probability of heads in a single future trial under a $Beta(a, b)$ posterior. We have

$$\begin{aligned} p(\tilde{x} = 1|D) &= \int_0^1 p(x = 1|\theta)p(\theta|D)d\theta \\ &= \int_0^1 \theta Beta(\theta|a, b)d\theta = \mathbb{E}[\theta|D] = \frac{a}{a+b} \end{aligned}$$

- ▶ Thus we see that the mean of the posterior predictive distribution is equivalent to plugging in the posterior mean parameters: $p(\tilde{x}|D) = Ber(\tilde{x}|\mathbb{E}[\theta|D])$.

Overfitting and the black swan paradox!!

Доколку естимираме тета со точкаст оценувач или со MLE и се паѓаат 5 пати 5ка од 5 фрлања тоа може да се деси ако тета=0, ама имаме само 5 фрлања и тоа е проблем на overfitting значи параметарот кој го естимираме е премногу зависен од податоците што ги имаме што е голем проблем кога имаме малце податоци, бидејќи да фрлим 5 пати со ред паричка **петка** може да се падни сите 5 пати реално, и не би требало само на тие 5 фрлања да направиме проценка дека секогаш ќе се падне глава. И ова се вика zero-count problem поврзано со мали множества на податоци. И ова се сведува на black swan paradox филозофијата каде што во 17 век сите мислеле дека има само бели лебди поради тоа што немале видено поинакви, подоцна се дознало дека имало и црни, тоа го опишува и нашиот проблем бидејќи ние видовме само глави да паднат си рековме дека секогаш ќе падне глава и заради тоа е параметарот е overfitting, за да се реши ова со бајес е лесно само додаваме приор каде што најдноставно е униформен приор каде А и Б се 1 и тогаш веројатноста на паѓање **глава** ќе биде мала но поголема од нула.

Аналогно на Бета бајномниот модел каде што имаме фрлање паричка биномно 0/1 тука имаме multinomial model каде што фрламе коцка која има повеќе страни, скоро сите работи се исти. Likelihood-от не доаѓа од бернулиева ни биномна туку од Мултиномна распределба

- If we assume the data is iid, the likelihood has the form

$$p(D|\theta) = \prod_{k=1}^K \theta_k^{N_k}$$

where N_k is the number of times event k occurred (these are the sufficient statistics for this model).

- The likelihood for the multinomial model has the same form as the binomial model, up to an irrelevant constant factor.

Сега пак ни треба некој приор каде што е којнугиран на ова likelihood како истите причини од прееска

- Fortunately, the Dirichlet distribution satisfies both criteria. So we will use the following prior

$$Dir(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^K \theta_k^{\alpha_k-1} \mathbb{I}(x \in S_k)$$

- Multiplying the likelihood by the prior, we find that the posterior is also Dirichlet:

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

$$\propto \prod_{k=1}^K \theta_k^{N_k} \theta_k^{\alpha_k-1} = \prod_{k=1}^K \theta_k^{N_k + \alpha_k - 1} = Dir(\theta|\alpha + N)$$

- We see that the posterior is obtained by adding the prior hyper-parameters (pseudo-counts) α_k to the empirical counts N_k .

Постериор- секоја класа има посебна дистрибуција и посебен параметар

- The posterior predictive distribution for a single multinoulli trial is given by the following expression:

$$p(X = j|D) = \int p(X = j|\theta)p(\theta|D)d\theta$$

$$= \int \theta_j p(\theta_j|D)d\theta_j = \mathbb{E}[\theta_j|D] = \frac{\alpha_j + N_j}{\alpha_0 + N}$$

- The above expression avoids the zero-count problem. In fact, this form of Bayesian smoothing is even more important in the multinomial case than the binary case, since the likelihood of data sparsity increases once we start partitioning the data into many categories.

Naive Bayes Classifiers

Имаме класификација на вектор во класа С, треба пак да дефинираме likelihood и приор. Поентата во овој модел што е специфичен е во стандарден случај имаме вектор кој што доаѓа со специфична класа и сеа тој вектор доколку се содржи во Д карактеристики сите се координирани измеѓу себе, така доколку сакаме да пресметаме веројатност, имаме некој вектор X кој има D атрибути, па веројатноста да се појави овој вектор е веројатноста да се појави првиот атрибут X1 по веројатноста да се појави X2 доколку се појави X1 итн верижно правило. Поентата е бидејќи се координирани не можеме оделно да ги третираме. Наивен се вика бидејќи претпоставува дека сите атрибути се независни.

$$p(x|y = c, \theta) = \prod_{j=1}^D p(x_j|y = c, \theta_{jc})$$

Model fitting - MLE for NBC

- ▶ The probability for a single data case is given by

$$\begin{aligned} p(x_i, y_i|\theta) &= p(y_i|\pi) \prod_j p(x_{ij}|\theta_j) = \\ &\prod_c \pi_c^{\mathbb{I}(y_i=c)} \prod_j \prod_c p(x_{ij}|\theta_{jc})^{\mathbb{I}(y_i=c)} \end{aligned}$$

- ▶ We have one term concerning π and DC terms containing the θ_{jc} 's and we can optimize these parameters separately.

- ▶ The MLE for the class prior π is given by

$$\hat{\pi}_c = \frac{N_c}{N}$$

- ▶ The MLE for the likelihood θ depends on the type of distribution we choose to use for each feature. For simplicity, let us suppose all features are binary. In this case, the MLE becomes

$$\hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$$

Bayesian naive Bayes

- ▶ The trouble with maximum likelihood is that it can overfit.
- ▶ A simple solution to overfitting is to be Bayesian.
- ▶ We will use a $Dir(\alpha)$ prior for π and a $Beta(\beta_0, \beta_1)$ prior for each θ_{jc} . Often we just take $\alpha = 1$ and $\beta = 1$, corresponding to add-one or Laplace smoothing.
- ▶ Combining the factored likelihood with the factored prior above gives the following factored posterior:

$$p(\theta|D) = p(\pi|D) \prod_j \prod_c p(\theta_{jc}|D)$$

$$p(\pi|D) = Dir(N_1 + \alpha_1, \dots, N_c + \alpha_c)$$

$$p(\theta_{jc}|D) = Beta(N_c - N_{jc} + \beta_0, N_{jc} + \beta_1)$$

Using the model for prediction

- ▶ At test time, the goal is to compute

$$p(y = c|x, D) \propto p(y = c|D) \prod_{j=1}^D p(x_j|y = c, D)$$

- ▶ The correct Bayesian procedure is to integrate out the unknown parameters:

$$p(y = c|x, D) \propto \int Cat(y = c|\pi) p(\pi|D) d\pi$$

$$\prod_{j=1}^D \int Ber(x_j|y = c, \theta_{jc}) p(\theta_{jc}|D) d\theta$$

Classifying documents using bag of words

- ▶ Document classification is the problem of classifying text documents into different categories. One simple approach is to represent each document as a binary vector, which records whether each word is present or not, so $x_{ij} = 1$ iff word j occurs in document i , otherwise $x_{ij} = 0$.
- ▶ We can then use the following class conditional density:

$$p(x_i|y_i = c, \theta) = \prod_{j=1}^D Ber(x_{ij}|\theta_{jc})$$

- ▶ This is called the **Bernoulli product model**, or the **binary independence model**.

LECTURE 2 - Gaussian models / Гаусови модели

Прво ќе почнеме со Гаусова распределба, значи имаме едно-деменционална и повеќе-деменционална. Повеќе-деменционалната е всушност генерализација на едно-д. Во повеќед сигма мало се менува со големо сигма и тоа е тоа. Значи општо за секоја форма кога размислуваме треба да имаме некоја логика зошто е тоа така. Па основата на нормална дистрибуција се наоѓа во експонентот, а пре експонентот е некоја си константа за нормализација. Едно-деменционалната ја описуваме: што ни кажува всушност делот во експонентот, за некоја случајна променлива x , и притоа гаусовата дистрибуција е дефинирана со μ што е просекот и σ^2 што е варијансата, што поблиску е до просекот толку е поголема веројатноста тоа да се случи.

Основни работи пред да почнеме, прво да ги кажеме правилата како и што ќе означуваме низ предавањето. Дефинираме со \mathbf{x} – мало болдирано x , означуваме вектор. \mathbf{X} – големо болдирано X , матрици, а елементите на матриците со X_{ij} . Сега ќе збровуваме за повеќе деменционална Гаусова распределба или нормална распределба со D димензии која е најкористена заедничка веројатност кај непрекинати променливи. Дефинирана Гаусова рапределба е запишана вака:

$$\mathcal{N} = (\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2} |\boldsymbol{\Sigma}|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

Буквално е истата формула како едно-деменционална но има некои промени, значи 2π станува на степен D , што воглавно не треба, $\boldsymbol{\Sigma}^{1/2}$ – коваријанса матрица. Изразот во експонентот е Махалонобис растојание помеѓу податочниот вектор \mathbf{x} и векторот на просек $\boldsymbol{\mu}$, и место со варијанса овде делиме со матрицата $\boldsymbol{\Sigma}$. За да го разбереме подобро ова равенство ќе направиме еигенција на $\boldsymbol{\Sigma}$. Транспонирана е $(\mathbf{x}-\boldsymbol{\mu})$ затоа што е вектор, бидејќи зборуваме за D -деменционална имаме D – деменционални вектори.

Во линеарна алгебра еигенција(Eigendecomposition) претставува метод да разложиш во нејзините eigenvalues – сопствени вредности and eigenvectors – сопствени вектори. Пример за матрицата A , ако $Av=\lambda v$ тогаш v е eigenvector на матрицата A , а λ е соодветен eigenvalue. Ова е, ако матрицата A е помножена со вектор и резултатот е истиот вектор ама е скалар вектор, тогаш тој вектор е eigenvector на матрицата, а скалирачкиот фактор е eigenvalue.

После добиваме, $\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, каде \mathbf{U} е ортонормална матрица од eigenvectors кои го задоволуваат условот $\mathbf{U}^T \mathbf{U} = \mathbf{I}$, и $\boldsymbol{\Lambda}$ дијагонална матрица од eigenvalues, па со употреба на еигенција добиваме:

$$\boldsymbol{\Sigma}^{-1} = \mathbf{U}^{-T} \boldsymbol{\Lambda}^{-1} \mathbf{U}^{-1} = \mathbf{U} \boldsymbol{\Lambda}^{-1} \mathbf{U}^T = \sum_{i=1}^D \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T$$

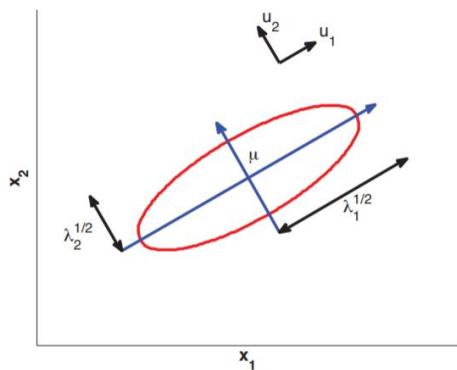
Каде U_i е i -тата колона во матрицата U , и го содржи и-тиот eigenvector, па може да го презапишеме Махалобис растојанието:

$$\begin{aligned} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) &= (\mathbf{x} - \boldsymbol{\mu})^T \left(\sum_{i=1}^D \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T \right) (\mathbf{x} - \boldsymbol{\mu}) \\ &= \sum_{i=1}^D \frac{1}{\lambda_i} (\mathbf{x} - \boldsymbol{\mu})^T \mathbf{u}_i \mathbf{u}_i^T (\mathbf{x} - \boldsymbol{\mu}) = \sum_{i=1}^D \frac{y_i^2}{\lambda_i} \end{aligned}$$

Па равенството на еклипсата во две димензии е:

$$\frac{y_1^2}{\lambda_1} + \frac{y_2^2}{\lambda_2} = 1$$

Оттука, можеме да видиме дека контурите со еднаква густина на веројатност на Гаус лежат покрај елипите, т.е се описаны со елипсоида.



Eigenvectors (сопствени вектори) ја утврдува ориентацијата на еклипсата, додека пак eigenvalues (сопствени вредности), колку ќе биде издолжена. Заклучивме дека Махаланобис растојанието одговара со Евклидово растојание во трансформиран координатен систем, каде шифтаме за μ и ротираме за U .

Исто така да кажеме дека една матрица може да се претстави како производ од 3 матрици, едната ќе е дијагонална, на дијагоналата се наоѓаат сопствените вредности, ако е 5 демензионална ќе има 5 сопствени

Освен тие сопствени вредности на секој од тие сопствени вредности има и сопствен вектор и тој е означен со u . Во две демензионални е елипсата погоре го кажавме истото.

MLE for an MVN

Првата работа што треба да направиме е, претпоставувавме дека ние имаме со D -демензионален вектор, а со N количеството на податоци. Претпоставката дека имаме N iid примероци од нормална распределба, она што сакаме ние да направиме е да пресметаме е следното:

$$\begin{aligned}\hat{\mu}_{mle} &= \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \stackrel{\text{def}}{=} \bar{\mathbf{x}} \\ \hat{\Sigma}_{mle} &= \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \\ &= \frac{1}{N} \left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right) - \bar{\mathbf{x}} \bar{\mathbf{x}}^T\end{aligned}$$

Практично μ , во едно-д и во повеќе-д е просек од империските веројатности, додека варијансната и коваријансната матрица се добиени од самите податоци.

Овие два параметри уште се нарекуваат sufficient statistic на MVN мулти нормлана распределба. Тоа го правиме со MLE, имаме табела пример која кажува дека можеме да ги пресметаме параметрите ако имаме N iid (идентично независни примероци) од распределбата, тогаш тоа можеме да го направиме со горните формули. Средната вредност се пресметува стандардно и матрицата стандардно преку долната формула.

**Кога станува збор за податоци, за модели, ние претпоставуваме дека можеме податоците да ги моделираме со Гаус, т.е всушност тие податоци што ги имаме се семплирани од некоја распределба, ако речеме пример тоа е гаус, целта на анализата е да ги најдиме параметрите. Она што е важно кога нешто правиме ние прво претпоставуваме, но треба да запамтиме дека сите модели во статистика се грешни но целта е да го најдиме најкорисниот и најсоодветниот модел.

Доказ (книга):

За да ги докажеме резултатите, нам ќе ни треба неколку резултати од алгебра за матрици. Во равенството \mathbf{a} и \mathbf{b} се вектори, а \mathbf{A} и \mathbf{B} се матриците, исто така нотацијата $\text{tr}(\mathbf{A})$ се однесува на трака на матрицата што е сума од сопствените дијагонали $\text{tr}(\mathbf{A}) = \sum_i A_{ii}$.

$$\begin{aligned}\frac{\partial(\mathbf{b}^T \mathbf{a})}{\partial \mathbf{a}} &= \mathbf{b} \\ \frac{\partial(\mathbf{a}^T \mathbf{A} \mathbf{a})}{\partial \mathbf{a}} &= (\mathbf{A} + \mathbf{A}^T) \mathbf{a} \\ \frac{\partial}{\partial \mathbf{A}} \text{tr}(\mathbf{B} \mathbf{A}) &= \mathbf{B}^T \\ \frac{\partial}{\partial \mathbf{A}} \log |\mathbf{A}| &= \mathbf{A}^{-T} \triangleq (\mathbf{A}^{-1})^T \\ \text{tr}(\mathbf{ABC}) &= \text{tr}(\mathbf{CAB}) = \text{tr}(\mathbf{BCA})\end{aligned}$$

Последното равен е наречено циклично – пермутирачко свойство на tr – операторот.

Proof. We can now begin with the proof. The log-likelihood is

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \log p(\mathcal{D}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Lambda} (\mathbf{x}_i - \boldsymbol{\mu})$$

where $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ is the precision matrix.

Using the substitution $\mathbf{y}_i = \mathbf{x}_i - \boldsymbol{\mu}$ and the chain rule of calculus, we have

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\mu}} (\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) &= \frac{\partial}{\partial \mathbf{y}_i} \mathbf{y}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{y}_i \frac{\partial \mathbf{y}_i}{\partial \boldsymbol{\mu}} \\ &= -1(\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}^{-T}) \mathbf{y}_i \end{aligned} \quad \text{оттука}$$

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\mu}} \ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= -\frac{1}{2} \sum_{i=1}^N -2\boldsymbol{\Sigma}^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) = \boldsymbol{\Sigma}^{-1} \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu}) = 0 \\ \hat{\boldsymbol{\mu}} &= \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i = \bar{\mathbf{x}} \end{aligned}$$

So the MLE of $\boldsymbol{\mu}$ is just the empirical mean.

Now we can use the trace-trick to rewrite the log-likelihood for $\boldsymbol{\Lambda}$ as follows:

$$\begin{aligned} \ell(\boldsymbol{\Lambda}) &= \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \sum_i \text{tr}[(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \boldsymbol{\Lambda}] \\ &= \frac{N}{2} \log |\boldsymbol{\Lambda}| - \frac{1}{2} \text{tr}[\mathbf{S}_{\boldsymbol{\mu}} \boldsymbol{\Lambda}] \end{aligned}$$

where

$$\mathbf{S}_{\boldsymbol{\mu}} \triangleq \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

is the scatter matrix centered on $\boldsymbol{\mu}$. Taking derivatives of this expression with respect to $\boldsymbol{\Lambda}$ yields

$$\frac{\partial \ell(\boldsymbol{\Lambda})}{\partial \boldsymbol{\Lambda}} = -\frac{N}{2} \boldsymbol{\Lambda}^{-T} - \frac{1}{2} \mathbf{S}_{\boldsymbol{\mu}}^T = 0 \quad (4.21)$$

$$\boldsymbol{\Lambda}^{-T} = \boldsymbol{\Lambda}^{-1} = \boldsymbol{\Sigma} = \frac{1}{N} \mathbf{S}_{\boldsymbol{\mu}} \quad (4.22)$$

so

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^N (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \quad (4.23)$$

which is just the empirical covariance matrix centered on $\boldsymbol{\mu}$. If we plug-in the MLE $\boldsymbol{\mu} = \bar{\mathbf{x}}$ (since both parameters must be simultaneously optimized), we get the standard equation for the MLE of a covariance matrix. \square

Имаме извод, бидејќи бараме екстремности, потоа со изедначуваме на 0. Значи имаме N iid, заради тоа што се независни веројатностите ќе се произволни. Пример $x_1 * x_2..$ сите тие помножени ќе дојдат во експонент збир.

Она што ни треба е изводот, за да ги најдеме параметрите и да изедначиме на 0.

Gaussian discriminant analysis

Една главна апликација на MVN е да ја дефинираме класата условни густини во

$$p(\mathbf{x}|y=c, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

генеративен модел.

Резултирачката техника е наречена GDA – Gaussian discriminant analysis (иако е генеративен не дискриминативен класификатор). Ако $\boldsymbol{\Sigma}_c$ е дијагонално ова е еквивалентно на наивен Бајес.

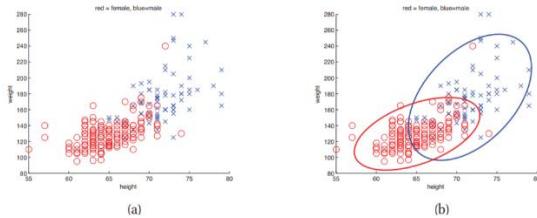


Figure 4.2 (a) Height/weight data. (b) Visualization of 2d Gaussians fit to each class. 95% of the probability mass is inside the ellipse. Figure generated by gaussHeightWeight.

Може да класифицираме вектор со атрибути користеќи го следното правило за одлука:

$$\hat{y}(\mathbf{x}) = \operatorname{argmax}_c [\log p(y = c | \boldsymbol{\pi}) + \log p(\mathbf{x} | \boldsymbol{\theta}_c)]$$

При што c – е ознака на класата, пример 10 класи(0,1,2..), тета се параметрите, заедничка ознака за *ми и сигма*.

Кога ќе ја пресметаме веројатноста на \mathbf{x} од секоја условна класна густина, ние ја мериме дистанцата од \mathbf{x} до центарот на секоја класа μ_c користеќи Маханолобис растојание, ова може да се смета како најблизок центриран касификатор. Ако имавме приор, можеме да направиме нов тест вектор како:

$$\hat{y}(\mathbf{x}) = \operatorname{argmin}_c (\mathbf{x} - \boldsymbol{\mu}_c)^T \boldsymbol{\Sigma}_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c)$$

► Generative classifier

$$p(y = c | \mathbf{x}, \theta) = \frac{p(y = c | \theta)p(\mathbf{x} | y = c, \theta)}{\sum_{c'} p(y = c' | \theta)p(\mathbf{x} | y = c', \theta)}$$

Целта овде е да го пресметаме постериорот тоа го правиме со горе наведената формула.

Class-conditional density $p(x|y = c)$ and the class prior $p(y = c)$

Quadratic discriminant analysis (QDA)

Може да добиеме понатамошен увид во овој модел со вклучување на дефиницијата на Гаусова густина:

$$p(y = c | \mathbf{x}, \boldsymbol{\theta}) = \frac{\pi_c |2\pi\boldsymbol{\Sigma}_c|^{-\frac{1}{2}} \exp \left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_c)^T \boldsymbol{\Sigma}_c^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) \right]}{\sum_{c'} \pi_{c'} |2\pi\boldsymbol{\Sigma}_{c'}|^{-\frac{1}{2}} \exp \left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_{c'})^T \boldsymbol{\Sigma}_{c'}^{-1} (\mathbf{x} - \boldsymbol{\mu}_{c'}) \right]}$$

Преминувањето на ова ќе резултира со квадратна функција на \mathbf{x} . Резултатот е познат како квадратна резултантна анализа QDA . На slikата може да видиме како изгледаат одлучните граници во 2Д.

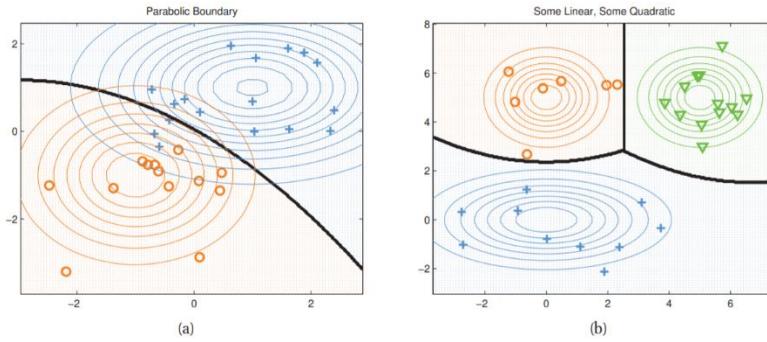


Figure 4.3 Quadratic decision boundaries in 2D for the 2 and 3 class case. Figure generated by `discrimAnalysisDboundariesDemo`.

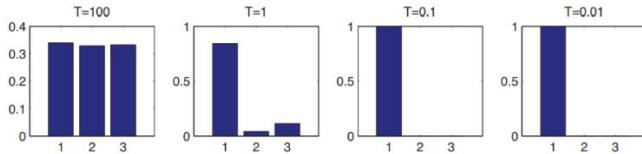


Figure 4.4 Softmax distribution $S(\boldsymbol{\eta}/T)$, where $\boldsymbol{\eta} = (3, 0, 1)$, at different temperatures T . When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is “spiky”, with all its mass on the largest element. Figure generated by `softmaxDemo2`.

Linear discriminant analysis (LDA)

Сега имаме специјален случај каде коваријните на матриците се врзани за една или споделени низ класите, $\Sigma_c = \Sigma$.

$$p(y = c | \mathbf{x}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\beta}_c^T \mathbf{x} + \gamma_c}}{\sum_{c'} e^{\boldsymbol{\beta}_{c'}^T \mathbf{x} + \gamma_{c'}}} = S(\boldsymbol{\eta})_c$$

where $\boldsymbol{\eta} = [\boldsymbol{\beta}_1^T \mathbf{x} + \gamma_1, \dots, \boldsymbol{\beta}_C^T \mathbf{x} + \gamma_C]$, and S is the **softmax** function, defined as follows:

$$S(\boldsymbol{\eta})_c = \frac{e^{\eta_c}}{\sum_{c'=1}^C e^{\eta_{c'}}} \quad (4.39)$$

The softmax function is so-called since it acts a bit like the max function. To see this, let us divide each η_c by a constant T called the **temperature**. Then as $T \rightarrow 0$, we find

$$S(\boldsymbol{\eta}/T)_c = \begin{cases} 1.0 & \text{if } c = \operatorname{argmax}_{c'} \eta_{c'} \\ 0.0 & \text{otherwise} \end{cases} \quad (4.40)$$

Со други зборови, на ниски температури, распределбата поминува цело време во најверојатната состојба, но на високи температури ги поминува сите состојби рамномерно како на slikata погоре со сините графици.

The decision boundary between any two classes, say c and c' , will be a straight line:

$$\mathbf{x}^T (\boldsymbol{\beta}_{c'} - \boldsymbol{\beta}_c) = \gamma_{c'} - \gamma_c$$

Two-class LDA

За да имаме понатамошни видувања, во овие равенства, да го земеме бинарниот случај. Во овој случај постериорот е даден:

$$p(y=1|\mathbf{x}, \theta) = \frac{e^{\beta_1^T \mathbf{x} + \gamma_1}}{e^{\beta_1^T \mathbf{x} + \gamma_1} + e^{\beta_0^T \mathbf{x} + \gamma_0}} \quad (4.44)$$

$$= \frac{1}{1 + e^{(\beta_0 - \beta_1)^T \mathbf{x} + (\gamma_0 - \gamma_1)}} = \text{sigm}\left((\beta_1 - \beta_0)^T \mathbf{x} + (\gamma_1 - \gamma_0)\right) \quad (4.45)$$

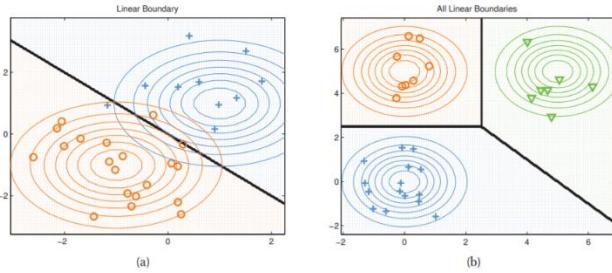
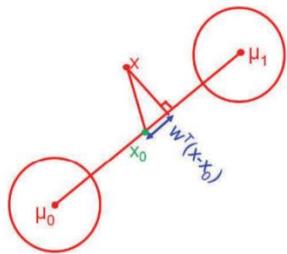


Figure 4.5 Linear decision boundaries in 2D for the 2 and 3 class case. Figure generated by `discrimAnalysisDboundariesDemo`.



По некои изведувања следи

$$p(y=1|\mathbf{x}, \theta) = \text{sigm}(\mathbf{w}^T(\mathbf{x} - \mathbf{x}_0))$$

На крајното правило на на одлука е : помести \mathbf{x} за \mathbf{x}_0 , прелсијај го на линијата \mathbf{w} , и види дали резултатот е позитивен или негативен. Ако $\Sigma = \sigma^2 I$, тогај \mathbf{w} е на првец $\mu_1 - \mu_0$. Па класифицираме дали проекцијата е поблиску до μ_1 или до μ_0 . Исто така ако $\pi_1 = \pi_0$, тогаш $\mathbf{x}_0 = \frac{1}{2} * (\mu_1 + \mu_0)$, што е на пола пат помеѓу пресеците. Ако $\pi_1 > \pi_0$ \mathbf{x}_0 е поблиску до μ_0 па повеќе од класата припаѓа на класата 1, приор. И обратно $\pi_1 < \pi_0$, гледаме дека класниот приор π_0 го смени својот праг на одлука, но не на целокупната геометрија.

MLE for discriminant analysis

- We now discuss how to fit a discriminant analysis model.
- The log-likelihood function is as follows:

$$\begin{aligned}\log p(\mathcal{D}|\theta) &= \sum_{i=1}^N \sum_{c=1}^C \mathbb{I}(y_i = c) \log \pi_c + \\ &\quad \sum_{c=1}^C \left[\sum_{j:y_j=c} \log \mathcal{N}(\mathbf{x}|\mu_c, \Sigma_c) \right]\end{aligned}$$

- For the class prior, we have $\hat{\pi}_c = N_c/N$.
- For the class-conditional densities, we partition the data based on its class label, and compute the MLE for each Gaussian.

До сега ни беше тема на дискусија како да ги фитуваме податоците на моделот, а сега ова да ги најдиме параметрите, во оваа анализа, ние освен параметри кои што доаѓаат од Гаус имаме уште пие параметарот т.е колкава е веројатноста да податоците припаѓаат на една од класите. μ и Σ ги наоѓаме на начин кој што го спомнавме претходно.

Inference in jointly Gaussian distributions

За дадена заедничка распределба $p(x_1, x_2)$, корисно е да можеме да ги пресметаме маргиналните веројатности $p(x_1)$ и условната $p(x_1|x_2)$.

Theorem 4.3.1 (Marginals and conditionals of an MVN). *Suppose $\mathbf{x} = (x_1, x_2)$ is jointly Gaussian with parameters*

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} \quad (4.67)$$

Then the marginals are given by

$$\begin{aligned}p(x_1) &= \mathcal{N}(x_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ p(x_2) &= \mathcal{N}(x_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})\end{aligned} \quad (4.68)$$

and the posterior conditional is given by

$$\begin{aligned}p(x_1|x_2) &= \mathcal{N}(x_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1} \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2} (\boldsymbol{\Lambda}_{11} \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2)) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1}\end{aligned} \quad (4.69)$$

Marginals and conditionals of a 2d Gaussian

- ▶ Let us consider a 2d example. The covariance matrix is

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

- ▶ The marginal $p(x_1)$ is a 1D Gaussian, obtained by projecting the joint distribution onto the x_1 line:

$$p(x_1) = \mathcal{N}(x_1 | \mu_1, \sigma_1^2)$$

- ▶ Suppose we observe $X_2 = x_2$; the conditional $p(x_1 | x_2)$ is obtained as:

$$p(x_1 | x_2) = \mathcal{N}\left(x_1 | \mu_1 + \frac{\rho\sigma_1\sigma_2}{\sigma_2^2}(x_2 - \mu_2), \sigma_1^2 - \left(\frac{\rho\sigma_1\sigma_2}{\sigma_2}\right)^2\right)$$

Linear Gaussian systems

Suppose we have two variables, \mathbf{x} and \mathbf{y} . Let $\mathbf{x} \in \mathbb{R}^{D_x}$ be a hidden variable, and $\mathbf{y} \in \mathbb{R}^{D_y}$ be a noisy observation of \mathbf{x} . Let us assume we have the following prior and likelihood:

$$\begin{aligned} p(\mathbf{x}) &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x) \\ p(\mathbf{y} | \mathbf{x}) &= \mathcal{N}(\mathbf{y} | \mathbf{A}\mathbf{x} + \mathbf{b}, \boldsymbol{\Sigma}_y) \end{aligned} \tag{4.124}$$

where \mathbf{A} is a matrix of size $D_y \times D_x$. This is an example of a **linear Gaussian system**. We can represent this schematically as $\mathbf{x} \rightarrow \mathbf{y}$, meaning \mathbf{x} generates \mathbf{y} . In this section, we show how to “invert the arrow”, that is, how to infer \mathbf{x} from \mathbf{y} . We state the result below, then give several examples, and finally we derive the result. We will see many more applications of these results in later chapters.

Statement of the result

Bayes rule for linear Gaussian systems. Given a linear Gaussian system, the posterior $p(\mathbf{x} | \mathbf{y})$ is given by the following:

$$\begin{aligned} p(\mathbf{x} | \mathbf{y}) &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_{x|y}, \boldsymbol{\Sigma}_{x|y}) \\ \boldsymbol{\Sigma}_{x|y}^{-1} &= \boldsymbol{\Sigma}_x^{-1} + \mathbf{A}^T \boldsymbol{\Sigma}_y^{-1} \mathbf{A} \\ \boldsymbol{\mu}_{x|y} &= \boldsymbol{\Sigma}_{x|y} \left[\mathbf{A}^T \boldsymbol{\Sigma}_y^{-1} (\mathbf{y} - \mathbf{b}) + \boldsymbol{\Sigma}_x^{-1} \boldsymbol{\mu}_x \right] \end{aligned}$$

In addition, the normalization constant $p(\mathbf{y})$ is given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} | \mathbf{A}\boldsymbol{\mu}_x + \mathbf{b}, \boldsymbol{\Sigma}_y + \mathbf{A}\boldsymbol{\Sigma}_x\mathbf{A}^T)$$

Inferring an unknown scalar from noisy measurements

Пртпоставуваме дека правиме N звучни мерења на y_i од некои основни вредности x , да претпоставиме дека мерката на звук има поравена прецизност $\lambda_y = 1/\sigma^2$, па лајклихудот е

$$p(y_i|x) = \mathcal{N}(y_i|x, \lambda_y^{-1})$$

Сега да употребиме Гаусов приор за вредноста на непознатиот извор:

$$p(x) = \mathcal{N}(x|\mu_0, \lambda_0^{-1})$$

Сакаме да пресметаме $p(x|y_1, \dots, y_N, \sigma^2)$. Ова можеме да го конвертираме во форма каде што може да употребиме Бајесово правило за Гаус, добиваме:

$$\begin{aligned} p(x|y) &= \mathcal{N}(x|\mu_N, \lambda_N^{-1}) \\ \lambda_N &= \lambda_0 + N\lambda_y \\ \mu_N &= \frac{N\lambda_y \bar{y} + \lambda_0 \mu_0}{\lambda_N} = \frac{N\lambda_y}{N\lambda_y + \lambda_0} \bar{y} + \frac{\lambda_0}{N\lambda_y + \lambda_0} \mu_0 \end{aligned}$$

Овие равенства се доста интуитивни, постериорната прецизност λ_N е приорната прецизност λ_0 плус N мерења прецизност λ_y . Исто така постериорниот просек е μ_N е конвексна комбинација од MLE \bar{y} и приорниот просек μ_0 . Ова го расчистува фактот дека постериорниот просек е компромис помеѓу MLE и приорот. Ако приорот е релативно слаб во однос на сигналот, (λ_0 is small relative to λ_y), даваме повеќе тежина на MLE. Ако приорот е релативно силен во однос на сигналот (λ_0 is large relative to λ_y), ние ставаме повеќе тежина на приорот.

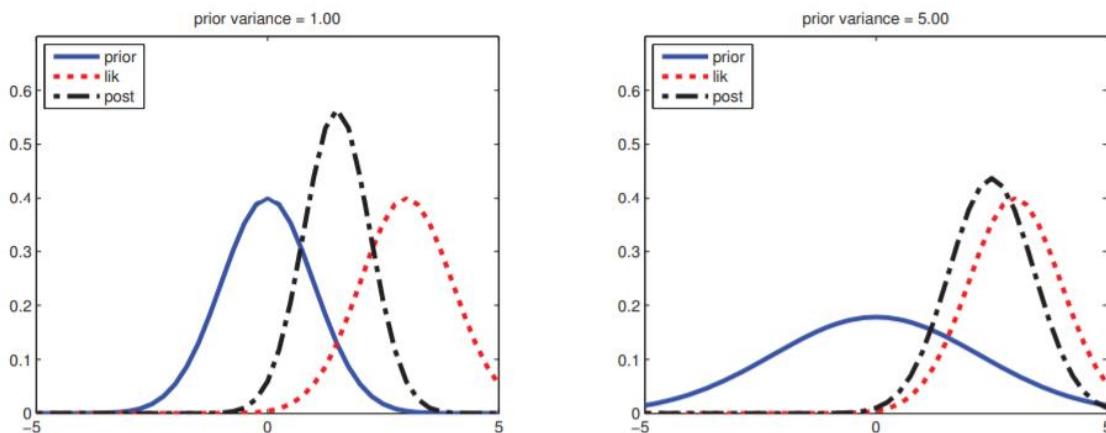


Figure 4.12 Inference about x given a noisy observation $y = 3$. (a) Strong prior $\mathcal{N}(0, 1)$. The posterior mean is “shrunk” towards the prior mean, which is 0. (b) Weak prior $\mathcal{N}(0, 5)$. The posterior mean is similar to the MLE. Figure generated by gaussInferParamsMean1d.

These equations are quite intuitive:

The posterior precision λ_N is the prior precision λ_0 plus N units of measurement precision λ_y .

The posterior mean μ_N is a convex combination of the MLE \bar{y} and the prior mean μ_0 .

This makes it clear that the posterior mean is a compromise between the MLE and the prior.

If the prior is weak relative to the signal strength (λ_0 is small relative to λ_y), we put more weight on the MLE.

If the prior is strong relative to the signal strength (λ_0 is large relative to λ_y), we put more weight on the prior.

Inferring an unknown vector from noisy measurements

Now consider N vector-valued observations, $\mathbf{y}_i \sim \mathcal{N}(\mathbf{x}, \Sigma_y)$, and a Gaussian prior, $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. Setting $\mathbf{A} = \mathbf{I}$, $\mathbf{b} = \mathbf{0}$, and using $\bar{\mathbf{y}}$ for the effective observation with precision $N\Sigma_y^{-1}$, we have

$$p(\mathbf{x}|\mathbf{y}_1, \dots, \mathbf{y}_N) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_N, \Sigma_N) \quad (4.142)$$

$$\Sigma_N^{-1} = \Sigma_0^{-1} + N\Sigma_y^{-1} \quad (4.143)$$

$$\boldsymbol{\mu}_N = \Sigma_N(\Sigma_y^{-1}(N\bar{\mathbf{y}}) + \Sigma_0^{-1}\boldsymbol{\mu}_0) \quad (4.144)$$

Нас \mathbf{x} ќе ни претствува вистината, но непозната, локација на објект во 2Д просторот, како авион, а \mathbf{y}_i е звучно набљудување како биповите на радарот. Како што добиваме се повеќе бипови, ние сме поспособни да ја лоцираме позицијата на објектот, за тоа користиме Калман филтер алгоритмот. Сега да претпоставиме дека имаме повеќе сензори за мерење на сигналот и ги спојуваме заедно, ова е познато како сензорска фузија. Ако имаме повеќе набљудувања со различни коваријанси (кој одговара на секој сензор со различна точност) постериорт ќе биде соодветна тежина за просечните податоци. На сликата долу 4.14 ние употребуваме неинформативен приор на \mathbf{x} , $p(\mathbf{x}) = N(\boldsymbol{\mu}_0, \Sigma_0) = N(0, 10^{10}\mathbf{I}_2)$ па добиваме две noisy observations $\mathbf{y}_1 \sim N(\mathbf{x}, \Sigma_{y,1})$ and $\mathbf{y}_2 \sim N(\mathbf{x}, \Sigma_{y,2})$. We then compute $p(\mathbf{x}|\mathbf{y}_1, \mathbf{y}_2)$. Поставуваме 1.14(a) $\Sigma_{y,1} = \Sigma_{y,2} = 0.01\mathbf{I}_2$, па двата сензори се еднакво доверливи. Во овој случај постериорниот просек е на средина помеѓу \mathbf{y}_1 и \mathbf{y}_2 . 1.14(a) ние поставуваме $\Sigma_{y,1} = 0.05\mathbf{I}_2$ and $\Sigma_{y,2} = 0.01\mathbf{I}_2$ сега сензорот 2 е подоверлив од сензорот 1, во овој случај постериорниот просек е поблиску до \mathbf{y}_2 . 1.14(ц)

$$\Sigma_{y,1} = 0.01 \begin{pmatrix} 10 & 1 \\ 1 & 1 \end{pmatrix}, \quad \Sigma_{y,2} = 0.01 \begin{pmatrix} 1 & 1 \\ 1 & 10 \end{pmatrix} \quad (4.145)$$

so sensor 1 is more reliable in the y_2 component (vertical direction), and sensor 2 is more reliable in the y_1 component (horizontal direction). In this case, the posterior mean uses \mathbf{y}_1 's vertical component and \mathbf{y}_2 's horizontal component.

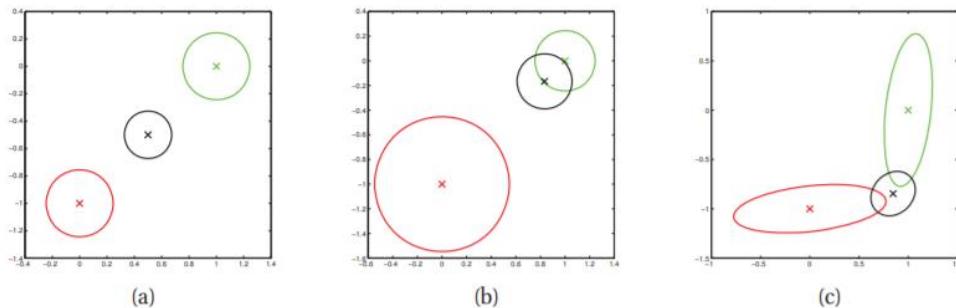


Figure 4.14 We observe $\mathbf{y}_1 = (0, -1)$ (red cross) and $\mathbf{y}_2 = (1, 0)$ (green cross) and infer $E(\boldsymbol{\mu}|\mathbf{y}_1, \mathbf{y}_2, \boldsymbol{\theta})$ (black cross). (a) Equally reliable sensors, so the posterior mean estimate is in between the two circles. (b) Sensor 2 is more reliable, so the estimate shifts more towards the green circle. (c) Sensor 1 is more reliable in the vertical direction, Sensor 2 is more reliable in the horizontal direction. The estimate is an appropriate combination of the two measurements. Figure generated by `sensorFusion2d`.

Infering the parameters of an MVN

Да претпоставиме дека податоците имаат Гаусова распределба $x_i \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for $i=1:N$ и целосно се набљудува, па немаме податоци кои може да ги пропуштиме а ни се од голема важност. За да поседноставиме најбитни работи ов овој дел се дека постериорот го делиме на три дела $p(\boldsymbol{\mu}|D, \boldsymbol{\Sigma})$; после пресместуваме $p(\boldsymbol{\Sigma}|D, \boldsymbol{\mu})$; на крај ја пресметуваме заедничката веројатност $p(\boldsymbol{\mu}, \boldsymbol{\Sigma}|D)$.

Posterior distribution of $\boldsymbol{\mu}$

Зборувавме како да пресметаме MLE $\boldsymbol{\mu}$, сега ќе зборуваме како да го пресметаме неговиот постериор, што е корисен за моделирање за нашата несигурност за неговата вредност.

The likelihood has the form

$$p(D|\boldsymbol{\mu}) = \mathcal{N}(\bar{\mathbf{x}}|\boldsymbol{\mu}, \frac{1}{N}\boldsymbol{\Sigma})$$

За поедноставно ќе користиме конјугиран приор, што во случајов е Гаусов. Во главно, ако $p(\boldsymbol{\mu}) = N(\boldsymbol{\mu}|m_0, V_0)$, тогаш можеме да го изведеме гаусиот постериор $\boldsymbol{\mu}$ базирано на резултатите

$$p(\boldsymbol{\mu}|D, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu}|m_N, V_N) \quad (4.172)$$

$$V_N^{-1} = V_0^{-1} + N\boldsymbol{\Sigma}^{-1} \quad (4.173)$$

$$m_N = V_N(\boldsymbol{\Sigma}^{-1}(N\bar{\mathbf{x}}) + V_0^{-1}m_0) \quad (4.174)$$

Ова е истиот процес со заклучување на локацијата на објектото со „бип“ сигнали, само што сега го заклучуваме просекот на распределбата на звучни примероци. (To a Bayesian, there is no difference between uncertainty about parameters and uncertainty about anything else.)

Posterior distribution of μ and Σ

Likelihood

$$p(\mathcal{D}|\mu, \Sigma) = (2\pi)^{-\frac{DN}{2}} |\Sigma|^{-\frac{N}{2}} \exp \left[-\frac{1}{2} \sum_{i=1}^N (\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu) \right]$$

Prior has the form of a Normal-inverse-wishart or NIW distribution $\text{NIW}(\mu, \Sigma | \mathbf{m}_0, \kappa_0, \nu_0, \mathbf{S}_0)$

The parameters of the NIW can be interpreted as follows: \mathbf{m}_0 is our prior mean for μ , and κ_0 is how strongly we believe this prior; and \mathbf{S}_0 is (proportional to) our prior mean for Σ , and ν_0 is how strongly we believe this prior.

Posterior

$$\begin{aligned} p(\mu, \Sigma | \mathcal{D}) &= \text{NIW}(\mu, \Sigma | \mathbf{m}_N, \kappa_N, \nu_N, \mathbf{S}_N) \\ \mathbf{m}_N &= \frac{\kappa_0}{\kappa_0 + N} \mathbf{m}_0 + \frac{N}{\kappa_0 + N} \bar{\mathbf{x}} \\ \kappa_N &= \kappa_0 + N \\ \nu_N &= \nu_0 + N \\ \mathbf{S}_N &= \mathbf{S}_0 + \mathbf{S}_{\bar{\mathbf{x}}} + \frac{\kappa_0 N}{\kappa_0 + N} (\bar{\mathbf{x}} - \mathbf{m}_0)(\bar{\mathbf{x}} - \mathbf{m}_0)^T \end{aligned}$$

This result is actually quite intuitive: the posterior mean is a convex combination of the prior mean and the MLE; and the posterior scatter matrix \mathbf{S}_N is the prior scatter matrix \mathbf{S}_0 plus the empirical scatter matrix $\mathbf{S}_{\bar{\mathbf{x}}}$ plus an extra term due to the uncertainty in the mean.

LECTURE 3 – Linear regression/Линеарна регресија

Линеарна регресија е еден од основните модели со кои започнале статистика и машинско учење да работат како целина.

Model specification

Претпоставуваме дека има нормална распределба регресијата, сега овде на има и сличности и разлики со класификација. Во класификација имаме инпут вектор \mathbf{X} со должина D – должина на атрибути, и имаме класа која е дискретно променлива, тука во линеарна регресија инпут векторот е ист само што имаме M на аутпут променлива која што е непрекината променлива, со која може да се симулираат многу работи како крвен притисок, висина, тежина итн. Во случајов работиме со едно-демензионална нормална дистрибуција заради тоа што аутпутот/излезот ни е едно-демензионален.

$$p(y|x, \theta) = N(y|w^T x, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{1/2} \exp\left(-\frac{1}{2\sigma^2}(y - w^T x)^2\right)$$

Претпоставуваме дека веројатноста да се добие аутпут y , е зависна од просекот и варијансата на нормална дистрибуција. Варијансата претпоставуваме дека е некоја зависна од податоците и зависна од тоа што моделираме, и побитниот дел е како да го моделираме просекот, односно за даден инпут вектор \mathbf{X} ќе да имаме истренирано кој се тежините. Во тренинг податоците може да имаме за исто \mathbf{X} се добива се две различни вредности за у тоа е должно на шумот. Во претходната лекција во истата формула имаме $x - \mu$, истот е ако ги замениме, во претходната лекција имавме x а тука имаме y , тука μ го заменуваме со ова w^t што е скаларен производ на тежините и на инпут векторот \mathbf{X} . Исто како што имаме инпут вектор со атрибути висина, тежина итн и сега за секое од тие атрибути имаме некоја тежина која ќе претставува колку тие се битни во таа на крај што е таргет променливата, колку се поврзани може да се позитивно негативно. Доколку сакаме да моделираме нешто нелинеарно можеме да користиме некој мапинг на самиот \mathbf{X} , може да е X^2, X^3 , секој би имале посебни коефициенти, притоа останува моделот дека е линеарна регресија поради w тоа што е линеарен. Ова е делумна основа и на многу други модели натаму.

$$p(y|x, \theta) = N(y|w^T \phi(x), \sigma^2)$$

Ова формула е позната како Баиси функцијско проширување, и линеарен е ради w .

Пример доколку би имале некакви податоци кои што се во форма на круг и се свезички а во тој круг од свездички има уште една класа од крукчина, тука во овој случај линеарната функција нема никаква цел да ја употребуваме, најдоброто што може да се добие е некоја квадратна функција што баш нема некоја цел, и нас ни треба линеарна функција која ќе ги раздели класите, класифицира. Ако имаме кружница измеѓу тие две калси би имале некаков вид на класификација. Но ако ги мапираме во друг простор многу полесно може да ги разделиме, на пример 3D.

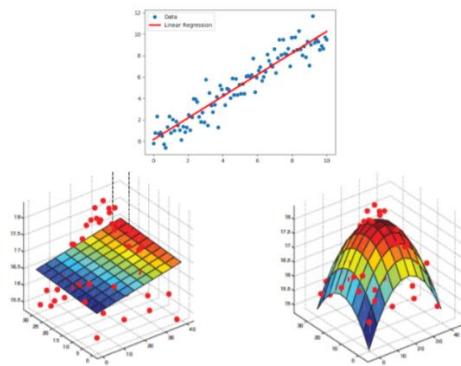


Figure 1: Linear regression applied to 1d and 2d data. Vertical axis is temperature, horizontal axes are location within a room. (a) The fitted plane has the form $f(x) = w_0 + w_1x_1 + w_2x_2$. (b) Temperature data is fitted with a quadratic of the form $f(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2$.

Првата слика е во еднодемензионален случај, како што кажавме податоците имаат шум, шумот може да претставува грешка на апаратот за мерење, заради тоа секогаш очекуваме дека ќе има некое отстапување од стандардната линија, но треба да најдеме која е линијата која најдобро ги опишува податоците, тоа е и целта на л.регресија да се најоптималната линија која што минимизира некоја метрика. Сите овие методи се за да се најде подобра функција. Сите методи надолу го прават истото на различен начин.

Тоа беше за горниот пример, сега за долнито, имаме повеќе-деменионални податоци, кои претставуваат темепература во скlop на соба, така да моделираме во кој дел на собата е пожешко ни треба најверојатно нешто понеквадратно како тоа десно на сликата, пошто може да го фати тој брз скок на температура за разлика од левата линеарна функција. И во тој случај ние додаваме и квадратни степени.

Maximum likelihood estimation (least squares)

Како што правевме и за класификација треба да се пресметаат кој се параметрите на моделот, нормално ги имаме μ и Σ , треба да ги најдеме кој се тие во случајов Σ ни е на некој начин дадено, ќе кажеме пак ако можеме да направиме некои претпоставли. ГЛАВНА ЦЕЛ ТУКА Е да се моделира просекот, односно $w^T x$, и би ги моделирале со равенка за права, имаме отстапување од координатен почеток и имаме отстапување од атрибутите на насоката на самата права. Првично бараме MLE, е сега многу често во вакви оптимациски процедури не секогаш се бара MLE, поради тоа што голем дел од случаите поленсо е да се минимизира некоја функција отколку да се максимизира. И заради тоа дефинираме минус од MLE што се вика negative log likelihood – NLL.

$$\mathbb{L}(\theta) \triangleq \log p(D|\theta) = \sum_{i=1}^N \log p(y_i|x_i, \theta)$$

$$NLL(\theta) \triangleq - \sum_{i=1}^N \log p(y_i|x_i, \theta)$$

Многу е полесно во линеарно програмирање, и операциони истражувања и има причина зошто е побитно да имаме конвексна функција во однос на конкавна. Доколку во MLE заменим нормална дистрибуција го добиваме буквально истото од прееска, и ова можеме

да го поделиме на два делови. Првиот дел ни е независен од w , и другиот дел е тој десно експонентот кој што после логаритам ќе се избриши останува сума. Log likelihood, има најголема вредност ако RSS е најмало, така да тоа е истото доколку бараме negative log likelihood треба да го минизиме RSS.

Now let us apply the method of MLE to the linear regression setting. Inserting the definition of the Gaussian into the above, we find that the log likelihood is given by

$$\begin{aligned}\mathbb{L}(\theta) &= \sum_{i=1}^N \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{1/2} \exp\left(-\frac{1}{2\sigma^2}(y_i - w^T x_i)^2\right)\right] \\ &= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \text{RSS}(w)\end{aligned}$$

RSS stands for **residual sum of squares** and is defined by

$$\text{RSS}(w) \triangleq \sum_{i=1}^N (y_i - w^T x_i)^2$$

Прилично битно во регресија, како што во класификација имаме прецизност, accuracy што ни кажива колку е точно тоа што ни е класифицирано, така во регресија бидејќи немаме специфична класа имаме некој начин да кажеме колку грешиме во некоја претпоставка, ако за некој вектор имаме некој аутпут после тоа претпоставиме и добијеме нов аутпут, разликата помеѓу оригиналниот што бил помеѓу овој ние што ќе го пресметаме, и тоа грешката ни се максимизира ако е прецизност се минимизира.

The RSS is also called the **sum of squared errors**, or **SSE**, and **SSE/N** is called the **mean squared error** or **MSE**. It can also be written as the square of the L_2 norm of the vector of residual errors:

$$\text{RSS}(w) = \|\epsilon\|^2 = \sum_{i=1}^N \epsilon_i^2$$

where $\epsilon_i = y_i - w^T x_i$

We see that the MLE for w is the one that minimizes the RSS, so this method is known as **least squares**.

Тоа што имаме ние во регресионата анализа, дадени се овие црвени кругчина кои ни се тренирани податоци на некој начин, наша цел е да ја најдеме црвената линија, и сега гредаме овие црвени точки повеќето не ни лежат на таа линија, како што спомнавме тие податоци немаат некој шум кој ги отстапувам и треба да најдеме која е линијата која најмногу одговара на податоците, така да не ги бараме специфично самите податоци, не ни треба нешто што се движи низ нив тоа е overfitting туку треба да најдеме линија која најдобро ги опишува нашите предвидувања, тоа би било тие x кои лежат на линијата, така доколку кругчината се оригиналните податоци, x -овите ни покажуваат ние што сме предвиделе,

линијата сина измеѓу о-х е грешката, а х ние го пресметваме, и целта ни е да се минимизира нели збирот на квадратите индивидуално.

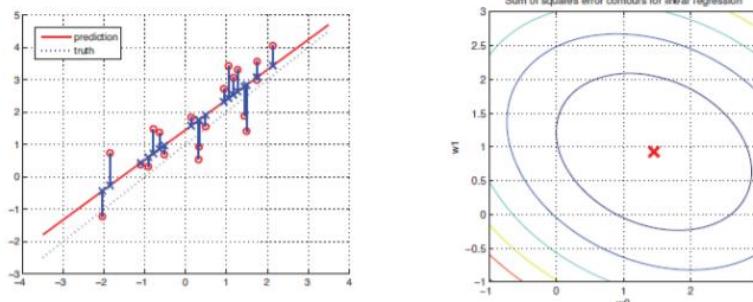


Figure 2: (a) In linear least squares, we try to minimize the sum of squared distances from each training point (denoted by a red circle) to its approximation (denoted by a blue cross), that is, we minimize the sum of the lengths of the little vertical blue lines. The red diagonal line represents $y(x) = w_0 + w_1x$, which is the least squares regression line. Note that these residual lines are not perpendicular to the least squares line. (b) Contours of the RSS error surface for the same example. The red cross represents the MLE, $w = (1.45, 0.93)$.

Derivation of the MLE

First, we rewrite the objective in a form that is more amenable to differentiation

$$NLL(w) = \frac{1}{2}(y - Xw)^T(y - Xw) = \frac{1}{2}w^T(X^T X)w - w^T(X^T y)$$

The gradient of this is given by

$$g(w) = [X^T Xw - X^T y] = \sum_{i=1}^N x_i(w^T x_i - y_i)$$

Equating to zero we get

$$X^T Xw = X^T y$$

This is known as the **normal equation**.

The corresponding solution to this linear system of equations is called the **ordinary least squares** or **OLS** solution, which is given by

$$\bar{w}_{OLS} = (X^T X)^{-1} X^T y$$

Десната слика е приказ на RSS, на сликата не се гледа неменионалноста но точката е минимум и како да имаме некој конус што расте нагоре и се зголемува. Конус што расте обратно со врвот долу. Колку отстапуваме толку грешката се зголемува и целта на овие методи е да ги најде на некој начин да конвергираме кон глобалниот минимум.

Сега ако пресметаме, во однос на негативниот лајклихуд, ова е буквально истото само со матрична форма каде што, X е матрицата со сите податоци, каде секоја редица е податок а секоја колона атрибут.

OLS – метод на најмали квадрати, тука имаме проблеми бидејќи не секогаш може ова да се пресмета ради матриците кои се неинверзабилни. Но ова е првиот начин на наоѓање на W .

Geometric interpretation

According to the upper result, our projected value of y is given by

$$\hat{y} = X\hat{w} = (X^T X)^{-1} X^T y$$

which corresponds to an orthogonal projection of y onto the column space of X .

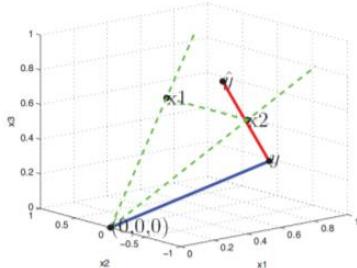


Figure 3: Graphical interpretation of least squares for $N = 3$ examples and $D = 2$ features.

Convexity

In least squares the NLL has a bowl shape with a unique minimum. The technical term for functions like this is convex. We say a set S is convex if we draw a line from θ to θ' , and all points on the line lie inside the set.

A function $f(\theta)$ is called convex if its epigraph (the set of points above the function) defines a convex set.

A function $f(\theta)$ is concave if $-f(\theta)$ is convex.

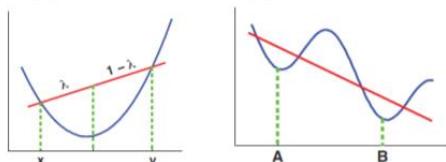
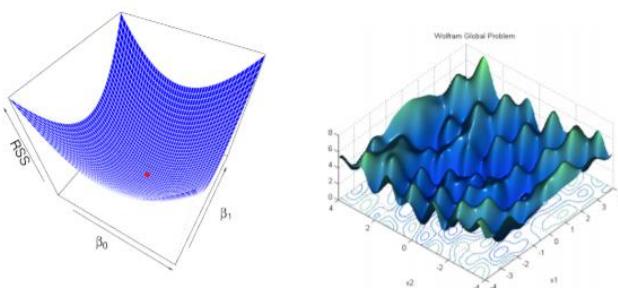


Figure 4: (a) Illustration of a convex function. We see that the chord joining $(x, f(x))$ to $(y, f(y))$ lies above the function. (b) A function that is neither convex nor concave. A is a local minimum, B is a global minimum

The RSS function vs a more complex Error function



Поентата е, имаме 3Д простор и во рој простор имаме 2Д простор кој што го формираат x -овите, тоа е траголникот испрекинат, у на друга страна лежи надвор од тој простор ради шумот и целта е да најдеме вредноста која е најблиску на у до овој простор со триаголникот, и таа е нашата предикција, и тоа се добива со равенката над цртежот.

Ова е битно, ако имаме конвексна функција, тута имаме еден минимум и тоа е оптимален, глобален минимум за целата функција, така да доколку го најдеме минимумот тоа е тој што го бараме, доколку не е конвексна тута имаме повеќе минимуми, тута имаме проблем на локален минимум, може да заглавиме на некој локален глобален, а нас ни треба глобален, па ние ќе мислиме дека локалниот е глобален и нема да продолжене да пребаствува понатаму.

Ако е конвекса има вид на чинија и како што видовме кај RSS, вториот е е секогаш позитивен кога функцијата е конвексна, па гледаме дека може да најдеме минимум.

На сликата десно, имаме многу минимуми и може од старт да заглавиме на тоа што ќе добиваме различно решение во зависност од стартните услови.

Robust linear regression

На почетокот претпоставивме дека линеарната регресија може да ја моделираме со нормална дистрибуција, заради тоа што претпоставивме дека шумот има нормална дистрибуција, што во општ случај важи. Но освен шум кај л.г има случаи каде што може да се појават исклучоци, кога ги имаме податоците може да имаме outliers на многу чидни места податоци на графикот.

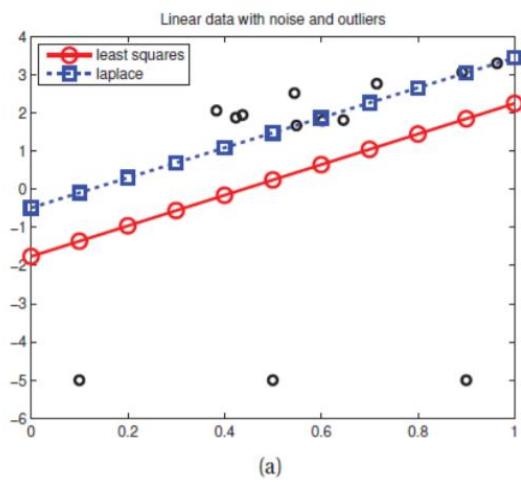


Figure 5: Illustration of robust linear regression.

Ridge regression

** Стандарден проблем со MLE е што може да први overfitting, тута имаме дали податоците ќе ги моделираме со линија или со нека функција која ни дава многу мала грешка за тренинг податоците, но за нови податоци нема ништо корисно да ни значи. Така да за ова да го решиме може да додадеме приор, на гаусова може да додадеме гаусов приор, и оваа регресија се нарекува Ridge regression. Примерот долу е на overfitting каде што имаме функција на 14 степен, и со 21 податок, овие се резултатите што се добиени, 6, 96, 100, 1000 итн а самите инпут вектори имаат вредност од 0-10, така да нема логика некој вектор совредност 0-10, во самата вредност да дава 10000 вредност, но дозволено е во овој случај поради тоа што некој соседен дава -5000 па така се изедначуваат од една страна и на крај се добива функција

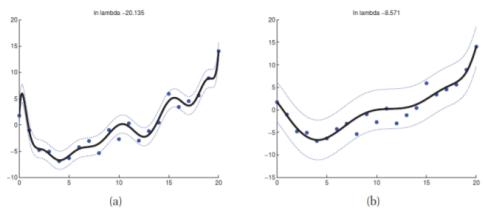


Figure 6: Degree 14 Polynomial fit to $N = 21$ data points with increasing amounts of L_2 regularization. Data was generated from noise with variance $\sigma^2 = 4$. The error bars, representing the noise variance σ^2 , get wider as the fit gets smoother, since we are ascribing more of the data variation to the noise.

Доколку се поконцентрирани на едната страна тие што се многу подалеку толку поголем квадратот. За еден од начините да се соочиме со овој проблем е место нормална дитрибуција да искористиме некој од методите кои ги спомнавме првиот час, како студентова Т или Лапласова распределба, тие се поотпорни на outliers бидејќи имаат heavy tails. Овој метод се нарекува **Robust linear regression**. Outliers гледаме да не ги земаме во предвид.

За ова да го решиме од почеток даваме приор, со просек 0 и со некоја прецизност односно сакаме од почеток помали да w , кога ќе го воведиме тоа ќе ја добијеме функцијата десно. Имаме уште еден дел што е зависен од параметрите, имаме епор функција која треба да ја минимизираме.

We can encourage the parameters to be small, thus resulting in a smoother curve, by using a zero-mean Gaussian prior:

$$p(w) = \prod_j N(w_j | 0, \tau^2)$$

where $1/\tau^2$ controls the strength of the prior.

The corresponding MAP estimation problem becomes

$$\operatorname{argmax}_{w} \sum_{i=1}^N \log N(y_i | w_0 + w^T x_i, \sigma^2) + \sum_{j=1}^D \log N(w_j | 0, \tau^2)$$

This is equivalent to minimizing the following

$$J(w) = \frac{1}{N} \sum_{i=1}^N (y_i - (w_0 + w^T x_i))^2 + \lambda ||w||^2$$

where $\lambda \triangleq \sigma^2 / \tau^2$ and $||w||^2 = w^T w$

Here the first term is the MSE/ NLL as usual, and the second term, $\lambda \geq 0$, is a complexity penalty. The corresponding solution is given by

$$\hat{w}_{\text{ridge}} = (\lambda I_D + X^T X)^{-1} X^T y$$

This technique is known as **ridge regression**, or **penalized least squares**.

In general, adding a Gaussian prior to the parameters of a model to encourage them to be small is called **L_2 regularization** or weight decay.

By penalizing the sum of the magnitudes of the weights, we ensure the function is simple (since $w = 0$ corresponds to a straight line, which is the simplest possible function, corresponding to a constant).

We illustrate this idea in Figure 6, where we see that increasing λ results in smoother functions. The resulting coefficients also become smaller. For example, using $\lambda = 10^{-3}$, we have 2.128, 0.807, 16.457, 3.704, -24.948, -10.472, -2.625, 4.360, 13.711, 10.063, 8.716, 3.966, -9.349, -9.232

It is common to use cross validation to pick λ .

It is possible to use a variety of different priors in this book. Each of these corresponds to a different form of regularization. This technique is very widely used to prevent overfitting. Interestingly, ridge regression, which works better statistically, is also easier to fit numerically, since $(\lambda I_D + X^T X)$ is much better conditioned (and hence more likely to be invertible) than $X^T X$, at least for suitable large λ .

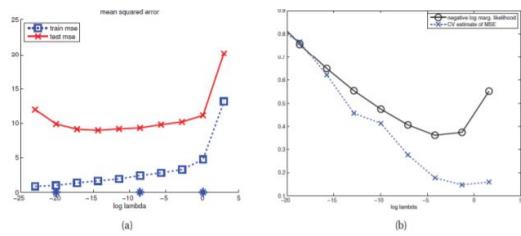


Figure 7: (a) Training error (dotted blue) and test error (solid red) for a degree 14 polynomial fit by ridge regression, plotted vs $\log(\lambda)$. Data was generated from noise with variance $\sigma^2 = 4$ (training set has size $N = 21$). Note: Models are ordered from complex (small regularizer) on the left to simple (large regularizer) on the right. (b) Estimate of performance using training set. Dotted blue: 5-fold cross-validation estimate of future MSE. Solid black: negative log marginal likelihood, $\log p(D|\lambda)$.

Прилично битно тута, каде во еттор функцијата додаваме некој, каде што во случајот е зависно од приорот, се вила **L_2 regularization**. Специфично за регуларизација е тоа што додаваме Гаусов приор но овој трик на етор функцијата може да додадеме нешто друго каде што за да ја намалиме комплексноста на алгоритмот и наши претпоставки за моделот да ги задоволиме, за тоа е прилично користена тактика.

Пример во програмирање каде што сакаме да оптимизираме некоја функција каде што мора сите тежини да се 1 или 0, и тоа само по себе ако пресметуваме би добиле некој си реален број, ретко 1 или 0 и заради тоа може да додадеме ваква претпоставка, во самата еттор функција каде што, ќе регулираме ако правиме отстапување од 1 или 0, па на крај се добива нешто пооптимално, така да L_2 регуларизација е битна, исто битно е дека во етор функцијата може да додаваме некој наши претпоставки за моделот направени.

Ова доведува ова w да се можно помали, така доколку ни требаат големи тежини на w или overfitting-уваме, на тренинг множеството ќе имаме многу добри резултати а на податочното множество ќе се многу лоши.

Ако користиме $\lambda=10^{-3}$ се добиваат понормални коефициенти за сите овие параметри. За да го најдеме оптималното ламбда препорачувачко е да се користи крос валидација, тренинг множеството го делиме на 5 дела, тренираме со првите 4, гледаме колку е гршката на последното 5-то, и така го менуваме само едното множество. На десната слика на долната оска е ламбда а на горната оска е еророт и можеме да видеме дека тие точки се од крос валидација, за негде каде ламбда тежи некај 10^{-3} се добиваат најоптимални резултати.

Regularization effects of big data

Доколку имаме многу податоци не би требало да имаме overfitting во зависност кој дел го користиме, али воглавно се намалува шансата за тоа да се случи. Тука можеме да видиме која е грешката на моделот како што се додаваат податоците, која е грешката ако имаме 10 податоци, па кога ќе се истренира на 100 податоци која е грешката и така натаму и тоа се вика learning curve на моделот и поради тоа што самото по себе има некој шум во податоците и не можеме да имаме грешка нула, значи секогаш имаме некоја грешка од самиот шум на податоците, која не може да се моделира и таа се вика noise floor. И другото structural error е тоа што сакаме да го минимизираме, која е грешката на нашиот модел во однос на вистината.

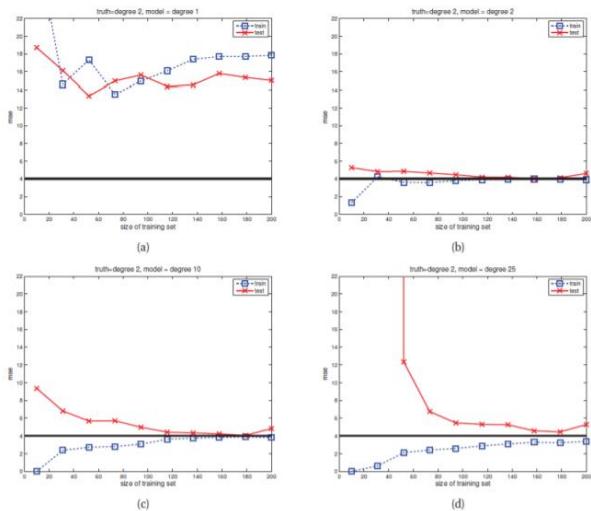


Figure 8: MSE on training and test sets vs size of training set, for data generated from a degree 2 polynomial with Gaussian noise of variance $\sigma^2 = 4$. We fit polynomial models of varying degree to this data.

Примерот е, вистинската функција на податоците кои се симплиирани од степен два и тука се прави модел еден со коефициентите се на степен 1, 2, 10, 25, едната оста означива време а другата грешка. Може да видиме, податоците ако се дојдени од модел со степен 2, ние моделираме со степен 1, ќе се конвергираат кон нешто каде што пак имаме грешка поради тоа што, underfitting-уваме бидејќи не го фаќаме моделот, а доколку имаме модел од степен 2 може да видиме дека брзо се конвергира кон тој noise floor, и за модели за степен 10 и 25 е исто.

Но тука е битно дека за првиот модел со степен 2 многу побрзо се конвергира во однос на модел со степен 25, односно колку е покомплексен моделот, имаме повеќе параметри за пресметување и за оптимизирање, и за тоа најдноставниот модел е најдобар, и како расте бројот на податоци кон бесконечност, очекуваме дека ќе стигне моделот до noise floor ако е точен. Но сепак иако живееме во време на big data каде што имаме милиони податоци, но евве ако земеме некое поле како податоци од интернет, за рекламирање пример на инстаграм, нас пак ни требаат и за small data модели.

Bayesian linear regression

Претходно се што зборувавме, требаше да најдеме која најоптималната w т.е кој се коефициентите на моделот, но сега сакаме да направиме цела дистрибуција во однос на нив, како што имавме на претходните моде, за тета во моделите баравме кој е MLE, Point estimate, но баравме и постериор параметри за тие параметри, бидејќи тие може да не се

секогаш точни најдобро е да се користи цела дистрибуција за нив. Поради тоа што имаме тука два параметри w , σ . Двата параметри се описаны со нормална дистрибуција. Ќе разгледуваме само случај даде што варијансата на податоците е непозната, а само пак ќе пресметуваме оптимално w , т.е цела дистрибуција на w .

Although ridge regression is a useful way to compute a point estimate, sometimes we want to compute the full posterior over w and σ^2

For simplicity, we will only look at the example where the noise variance σ^2 is known, so we focus on computing $p(w|D, \sigma^2)$.

We assume throughout a Gaussian likelihood model.

Performing Bayesian inference with a robust likelihood is also possible, but requires more advanced techniques.

Computing the posterior

- ▶ The conjugate prior to the above Gaussian likelihood is also a Gaussian, which we will denote by $p(w) = N(w|w_0, V_0)$.
- ▶ Using Bayes rule for Gaussians, the posterior is given by

$$\begin{aligned} p(w|D, \sigma^2) &= p(w|X, y, \sigma^2) \propto \\ &\propto N(w|w_0, V_0)N(y|Xw, \sigma^2 I_N) = N(w|w_N, V_N) \\ w_N &= V_N V_0^{-1} w_0 + \frac{1}{\sigma^2} V_N X^T y \\ V_N^{-1} &= V_0^{-1} + \frac{1}{\sigma^2} X^T X \end{aligned}$$

- ▶ If $w_0 = 0$ and $V_0 = \tau^2 I$, then the posterior mean reduces to the ridge estimate, if we define $\lambda = \sigma^2/\tau^2$.

To gain insight into the posterior distribution (and not just its mode), let us consider a 1D example:

$$y(x, w) = w_0 + w_1 x + \epsilon$$

where the "true" parameters are $w_0 = 0.3$ and $w_1 = 0.5$.

In Figure 9 we plot the prior, the likelihood, the posterior, and some samples from the posterior predictive.

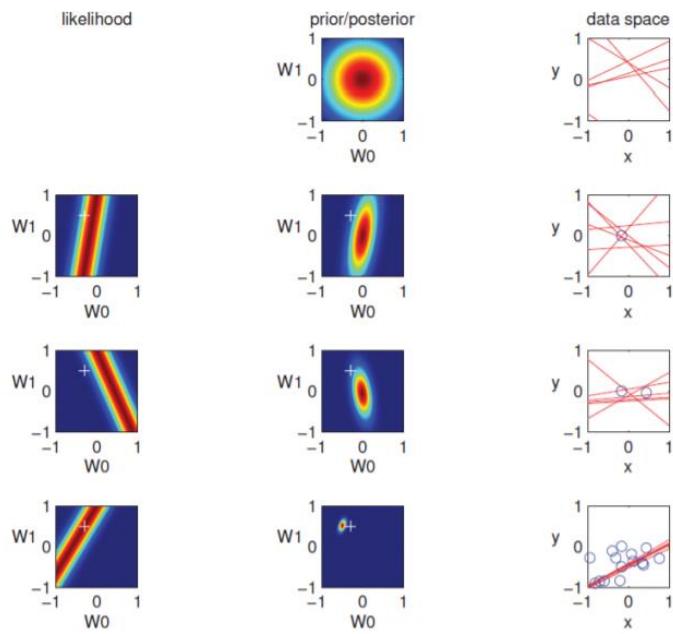


Figure 9: Sequential Bayesian updating of a linear regression model $p(y|x) = N(y|w_0x_0 + w_1x_1, \sigma^2)$.

На сликата пишува припр/постеиор, ова е секвенционален модел каде што од почеток имаме 0 податоци, на сликата имаме гаусов приор кога ќе го добиеме, првиот податок постериорот станува приор за првиот податок, така да секвенцијално го update-раме моделот, од почеток немаме податоци, кога ќе стигне податок, пориорт се менува во однос на лајклихудот. На крај идеално е да конвергираме кон x на сликата. Гледаме дека со тек на време, со поголем број на податоци и сите прави поминуваат во или близку до податоците о.

Разликата во тоа што претходно го правевме и сега го правиме е претходно, споено се бета бајномијал предавњето, таму имавме баес модел averaging и plug-in approximation, каде во бајес се земаше целата дистрибуција, немаме само едно w , туку имаме дистрибуција на прави и сите прави се од таа дистрибуција што ги семплираме, plug-in земаме една максимална, каде тука би зеле таа што е на центар на дистрибуцијата на податоците.

Computing the posterior predictive

Предноста на једниот метод во однос на другиот е, можеме да видиме дека, десно имаме σ^2 функција од x , т.е е зависна од x , во однос на варијансата на податоците и во однос на приорот на параметрите V . Тука големата предност е што доколку имаме податоци кои што се надвор од доменот на тренинг множеството, пример моделот е истрениран на едно место во просторот и се појави точка на друго место, реално не сме сигурни да кажеме за вредноста на точките во даден простор без разлика што некои се подалеку од тренирачкото множество, и тоа е предноста на оваа функција која што зависи од x , варијансата се згоелмува, колку што x е подалеку од тренинг податоците, така да доколку користиме, plug in апроксимација, т.е доколку избереме едно w , таа функцијата ја имаме е еднаква варијанса на истите податоци, тука поради тоа што користиме постериор предиктив, колку што бегаме од тренинг податоците толку сме понесигурни дека ќе го погодиме, варијансата е поголема.

The posterior predictive distribution at a test point x is also Gaussian:

$$p(y|x, D, \sigma^2) = \int N(y|x^T w, \sigma^2) N(w|w_N, V_N) dw = N(y|x^T w_N, \sigma_N^2)$$

$$\sigma_N^2(x) = \sigma^2 + x^T V_N x$$

The variance in this prediction, $\sigma_N^2(x)$, depends on two terms: the variance of the observation noise, σ^2 , and the variance in the parameters, V_N .

The latter translates into variance about observations in a way which depends on how close x is to the training data D .

This is illustrated in Figure 10, where we see that the error bars get larger as we move away from the training points, representing increased uncertainty. This is important for applications such as active learning, where we want to model what we don't know as well as what we do.

By contrast, if we use a plugin approximation of the parameters we have

$$p(y|x, D, \sigma^2) = \int N(y|x^T w, \sigma^2) \delta_{\hat{w}} dw = N(y|x^T \hat{w}, \sigma^2)$$

Note that the error bars now have a constant size.

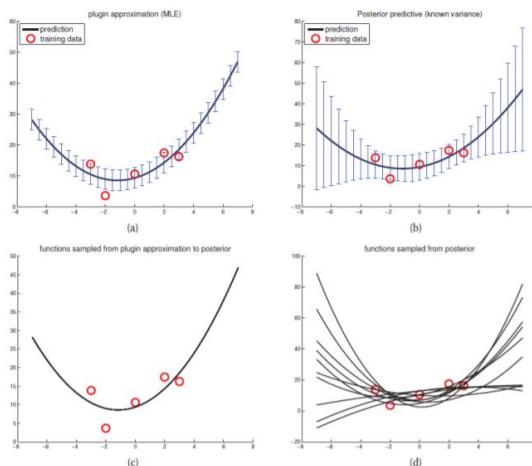


Figure 10: (a) Plug-in approximation to predictive density (we plug in the MLE of the parameters). (b) Posterior predictive density, obtained by integrating out the parameters

LECTURE 4 / Bayesian statistics – Бајесова статистика

До сега од тоа што поминавме се среќававме многу пати со пресметување на постериор вредности, како и на МАР – при што користевме различни приори. Исто така е дискутирана целата постериор веројатност. Во ова предавање ќе се задржиме на тоа што претставува јадро на целиот пристап тоа се вика Бајесов пристап. Пристапот на бајесиан, филозофски гледано е никој од нас, кога доаѓаме ништо не знаеме но после некое време имаме некакво знаење и тоа знаење би требало да го вклучиме во приорот. Всушност нас веројатноста ни претставува некаков вид на верување колку податоците т.е како да ги моделираме или објасниме. Е тука е целата таа субјективна веројатност т.е субјективен пристап за разлика од објективниот-класичниот каде што веројатноста се дефинира на класичен начин – фрекфентистичка веројатност. Во многу работи тие два пристапи се еквивалентни а за некои не, во некои работи кога ги користиме двата пристапи добиваме различни резултати од било кои причини. Битно е да се знае дека се прави разлика, со оглед на фактот што се многу слични. Кога станува збор за машинско учење многу повеќе е во школата на Бајесиан и тоа се гледа и од книгата од која учиме. Видовме некакви сумирачки работи кои се однесуваат на постериор дистрибуцијата, потоа ќе се зборуваме за мани кои ги има МАР естиматорот. Постериор некако ги сумира се она што знаеме за непознатите величини, до сега она што најчесто го работевме беше проценка на точка Point estimation, и тоа најчесто се прави со пресметување на средната вредност на mean. Најчесто овие величини се најчест извор за кој прима дискретни вредности тоа се постериор маргиналните. Она што го правиме со МАР, ние всушност го пресметуваме mod-от.

Сега да ги наведиме кои се маните на МАР естимацијата. Неколку се јасни доколку некој се малце покомплицирани. Првата мана е јасна, бидејќи пресметуваме точка, дали е тоа средна вредност или мода, таа е точка со други зборови не пресметуваме какво е нашето непознавање, немаме мерка за незнанењето, односно за грешките кои се прават. Во класична фрекфентистички пристап тука се пресметуваат граници, интервали на доверба, итн итн. Тоа треба секогаш да биде придрижено кога ние пресметуваме било каква вредност, она што е многу подобро место да се пресметува точкаст оценувач, да се пресметува дистрибуција тогаш имаме многу појаки информации. Резултира во overfitting то е artificial point – тоа што најчесто што го правиме е што оваа естимација не е инваријантна во однос на ако се менува или се врши промена во податоците и како тие се параметризираат на некој начин. Овие се неколку мани кога станува збор за естимација или проценка на една точка.

Mode се дефинира како точка која што најчесто се јавува во податочното множество и затоа таа пример и во книгата ако имаме дискретна дистрибуција и ако имаме некој пик во некоја мала околина, мерата на тој пик е мала но меѓутоа тоа е нешто што се јавува најчесто, па може да се каже дека средната вредност подобро го претставува податочното множество, ако сакаме податочното множество да го претставиме со среден број тогаш средната вредност е многу подобра од модата. Модата е всушност точка која што најчесто ја има, па кога правиме проценка подобро се претставува дистрибуцијата со просекот.

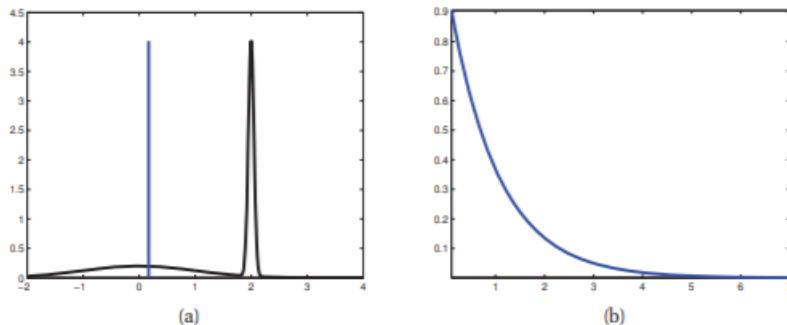


Figure 5.1 (a) A bimodal distribution in which the mode is very untypical of the distribution. The thin blue vertical line is the mean, which is arguably a better summary of the distribution, since it is near the majority of the probability mass. Figure generated by `bimodalDemo`. (b) A skewed distribution in which the mode is quite different from the mean. Figure generated by `gammaPlotDemo`.

Проблемот е тоа што го кажавме погоре, всушност кога пресметуваме просек или мода всушност ние на некоков начин го правиме на простер чија мера е различна од нула, целиот простор го земаме во обзир на мараметрите на податоците, во зависност од тоа како пресметуваме. Го решаваме проблемот со минимизирање на загуба при што бираме различни функции на загуба, тогаш ако ја избереме квадратната ќе добијеме posterior mean, ако избереме линеарната тоа ќе биде постериор модата.

- ▶ For 0-1 loss, the optimal estimate is the posterior mode.
- ▶ $L(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2$ the optimal estimator is the posterior mean.
- ▶ $L(\theta, \hat{\theta}) = |\theta - \hat{\theta}|$ the optimal estimate the posterior median.

Еден од начините како да го избегнеме тоа кога работиме, со кроистење на функции на загуба.

Друга мана на МАР естимацијата е тоа што не е инваријантна на репараметризацијата Ако ние сме пресметале постериор x и потоа направиме промена, место со x работиме со y , значи работиме со други т.е податоците ги представиме на друг начин со y тогаш дистрибуцијата на следниот начин.

Thus the MAP estimate depends on the parameterization.

Suppose we compute the posterior for x . If we define $y = f(x)$, the distribution for y is given

$$p_y(y) = p_x(x) \left| \frac{dx}{dy} \right|$$

The $|dx/dy|$ term is called the Jacobian, and it measures the change in size of a unit volume passed through f .

Let $\hat{x} = \operatorname{argmax}_x p_x(x)$ be the MAP estimate for x . In general it is not the case that $\hat{y} = \operatorname{argmax}_y p_y(y)$ is given by $f(\hat{x})$.

The MLE does not suffer from this since the likelihood is a function, not a probability density.

Ако пресметаме МАР за μ и таа ја означиме со x када и пресметаме со y меѓу нив нема релација. Овој проблем го нема за МЛЕ затоа што тоа не е дистрибуција туку е функција, и од таму кога бараме МЛЕ го немаме тој проблем што се јавува погоре.

- ▶ Example (due to Michael Jordan):
- ▶ The Bernoulli distribution is typically parameterized by its mean μ , so $p(y=1|\mu) = \mu$, where $y \in \{0, 1\}$.
- ▶ Suppose: a uniform prior on the unit interval: $p_\mu(\mu) = 1$.
- ▶ Let $\theta = \sqrt{\mu}$. Then

$$\begin{aligned} p_\theta(\theta) &= p_\mu(\mu) \left| \frac{d\mu}{d\theta} \right| = p_\mu(\mu) 2\theta = 2\theta \\ \hat{\theta}_{MAP} &= \arg \max_{\theta \in [0,1]} 2\theta = 1 \end{aligned}$$

- ▶ Let $\phi = 1 - \sqrt{1 - \mu}$. Then

$$\begin{aligned} p_\phi(\phi) &= p_\mu(\mu) \left| \frac{d\mu}{d\phi} \right| = p_\mu(\mu) 2(1 - \phi) = 2(1 - \phi) \\ \hat{\phi}_{MAP} &= \arg \max_{\phi \in [0,1]} 2(1 - \phi) = 0 \end{aligned}$$

Credible intervals

Со МАР ние најчесто сакаме да измериме интервали на доверба, тоа на некаков начин е поврзано со постериор дистрибуцијата, се дефинира со ова долу.

In addition to point estimates, we often want a measure of confidence.

A standard measure of confidence in some (scalar) quantity θ is the “width” of its posterior distribution.

This can be measured using a $100(1 - \alpha)\%$ credible interval, which is a region $C = (\ell, u)$ which contains $1 - \alpha$ of the posterior probability mass, i.e.,

$$C_\alpha(\mathcal{D}) = (\ell, u) : P(\ell \leq \theta \leq u | \mathcal{D}) = 1 - \alpha$$

There may be many such intervals, so we choose one such that there is $(1 - \alpha)/2$ mass in each tail; this is called a central interval

Она што сакаме да го добијеме е некој интервал кој што ќе содржи 1-алфа од постериор дистрибуцијата.

If the posterior has a known functional form, we can compute the posterior central interval using $\ell = F^{-1}(\alpha/2)$ and $u = F^{-1}(1 - \alpha/2)$, where F is the cdf of the posterior.

If the posterior is Gaussian, $p(\theta | \mathcal{D}) = \mathcal{N}(0, 1)$, and $\alpha = 0.05$, then $\ell = \Phi(\alpha/2) = -1.96$, and $u = \Phi(1 - \alpha/2) = 1.96$, where Φ denotes the cdf of the Gaussian.

This justifies the common practice of quoting a credible interval in the form of $\mu \pm 2\sigma$, where μ represents the posterior mean, σ represents the posterior standard deviation, and 2 is a good approximation to 1.96.

If we don't know the functional form, but we can draw samples from the posterior, then we can use a Monte Carlo approximation: we sort the S samples, and find the one that occurs at location α/S along the sorted list. As $S \rightarrow \infty$, this converges to the true quantile.

Bayesian model selection

When faced with a set of models (i.e., families of parametric distributions) of different complexity, how should we choose the best one?

This is called the model selection problem.

Bayesian model selection: first compute the posterior over models

$$p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{\sum_{m \in M} p(m, \mathcal{D})}$$

Then, compute the MAP model, $\hat{m} = \operatorname{argmax} p(m|\mathcal{D})$.

If we use a uniform prior over models, this results in choosing the model which maximizes

$$p(\mathcal{D}|m) = \int p(\mathcal{D}|\theta)p(\theta|m)d\theta$$

This quantity is called the marginal likelihood, the integrated likelihood, or the evidence for model m .

Bayesian Occam's razor

One might think that using $p(\mathcal{D}|m)$ to select models would always favor the model with the most parameters.

This is true if we use $p(\mathcal{D}|\hat{\theta})$ to select models, where $\hat{\theta}_m$ is the MLE or MAP estimate of the parameters for model m , because models with more parameters will fit the data better, and hence achieve higher likelihood.

However, if we integrate out the parameters, rather than maximizing them, we are automatically protected from overfitting: models with more parameters do not necessarily have higher marginal likelihood.

This is called the Bayesian Occam's razor effect, named after the principle known as Occam's razor, which says one should pick the simplest model that adequately explains the data.

Она што го правиме тука е, го пресметуваме постериорот преку сите модели и потоа правиме МАР за моделот. Ако користиме униформен приор можеме работите да ги појдноставиме. Тоа се следува на бирање на модел кој го максимира интегралот, тоа се вика маргинален лајклиход. Во некои случаи кога станува збот за релативни приори можеме да го пресметаме, но пред да дојдеме до проблемот како се пресметува маргиналниот лајклиход, треба да видима како се пресметува оштицата на орхан,

Принципот вели дека од повеќе модели, најдобро е да работиме со наједноставниот модел, кој што адекватно ги опишува податоците, од многу различни модели најдобриот е наједноставниот, ако е многу сложен може да не ги опишува добро податоците, треба да се работи со тие што имаат намалце параметри. Ако користиме обичен MLE/MAP може да се сличи да имаме overfitting.

One way to understand the Bayesian Occam's razor effect is to note that probabilities must sum to one.

Hence

$$\sum_{\mathcal{D}'} p(\mathcal{D}'|m) = 1,$$

where the sum is over all possible data sets.

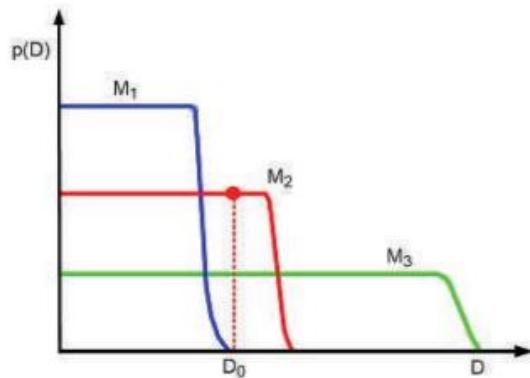
Complex models, which can predict many things, must spread their probability mass thinly, and hence will not obtain as large a probability for any given data set as simpler models.

This is called the conservation of probability mass principle.

Самото интегрирање на параметрите нас ни дава на некој начин да сме заштитени во overfitting, со други зборови моделите кои имаат повеќе параметри нема да имаат поголема MLE бидејќи всушност ние правиме сумирање во тој систем на модели и секој добива, различна тежина кога се сумираат моделите, уште една причина зошто работи овој принцип кога станува збоир за маргиналниот likelihood.

1.3.1

5.3. Bayesian model selection



Conservation of probability mass principle, and is illustrated in Figure 5.6. On the horizontal axis we plot all possible data sets in order of increasing complexity (measured in some abstract sense). On the vertical axis we plot the predictions of 3 possible models: a simple one, M1; a medium one, M2; and a complex one, M3. We also indicate the actually observed data D0 by a vertical line. Model 1 is too simple and assigns low probability to D0. Model 3 also assigns D0 relatively low probability, because it can predict many data sets, and hence it spreads its probability quite widely and thinly. Model 2 is “just right”: it predicts the observed data with a reasonable degree of confidence, but does not predict too many other things. Hence model 2 is the most probable model.

Computing the marginal likelihood (evidence)

When discussing parameter inference for a fixed model, we often wrote

$$p(\theta|\mathcal{D}, m) \propto p(\theta|m)p(\mathcal{D}|\theta, m)$$

thus ignoring the normalization constant $p(\mathcal{D}|m)$. This is valid since $p(\mathcal{D}|m)$ is constant wrt θ .

However, when comparing models, we need to know how to compute the marginal likelihood, $p(\mathcal{D}|m)$.

In general, this can be quite hard, since we have to integrate over all possible parameter values, but when we have a conjugate prior, it is easy to compute.

Кога пресметуваме имаме повеќе модели, ние сакаме да сумираме и да поделираме по сите модели, нас ни е битен, маргиналниот likelihood и тоа што сакаме да го направиме. Тоа е важно затоа што меѓудругото автоматски го регулира overfitting-от бидејќи дава поголема перспектива од повеќе го бираате најоптималниот, тоа е поголема ширина во нашата обработка на податоците, но тоа внесува и момент на пресметување на интеграл кој е записан во лекцијата, и е многу тежок.

Кога имаме парови на конјугирани приори, тогаш во некои случаи може да го направиме во многу едноставен начин. Доказ:

Let $p(\theta) = q(\theta)/Z_0$ be our prior, where $q(\theta)$ is an unnormalized distribution, and Z_0 is the normalization constant of the prior.

Let $p(\mathcal{D}|\theta) = q(\mathcal{D}|\theta)/Z_\ell$ be the likelihood, where Z_ℓ contains any constant factors in the likelihood.

Let $p(\theta|\mathcal{D}) = q(\theta|\mathcal{D})/Z_N$ be our posterior, where $q(\theta|\mathcal{D}) = q(\mathcal{D}|\theta)q(\theta)$ is the unnormalized posterior, and Z_N is the normalization constant of the posterior.

$$\begin{aligned} p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\ p(\mathcal{D}) &= \frac{Z_N}{Z_0 Z_\ell} \end{aligned}$$

Beta-binomial model

$$\begin{aligned}
 p(\theta|\mathcal{D}) &= \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\
 p(\mathcal{D}) &= \frac{Z_N}{Z_0 Z_\ell} \\
 \text{Beta}(\theta|a + N_1, b + N_0) &= \frac{1}{p(\mathcal{D})} \left[\frac{1}{B(a, b)} \theta^{a-1} (1-\theta)^{b-1} \right] \\
 &\quad \left[\binom{N}{N_1} \theta^{N_1} (1-\theta)^{N_0} \right] \\
 p(\mathcal{D}) &= \binom{N}{N_1} \frac{B(a + N_1, b + N_0)}{B(a, b)}
 \end{aligned}$$

Beta-Bernoulli model

Effect of the prior

Sometimes it is not clear how to set the prior.

When performing posterior inference, the details of the prior may not matter too much.

When computing the marginal likelihood, the prior plays a much more important role, since we are averaging the likelihood over all possible parameter settings.

If the prior is unknown, the correct Bayesian procedure is to put a prior on the prior. That is, we should put a prior on the hyper-parameter α as well as the parameters w .

To compute the marginal likelihood, we should compute

$$p(\mathcal{D}|m) = \int \int p(\mathcal{D}|w)p(w|\alpha, m)p(\alpha|m)dwd\alpha$$

This requires specifying the hyper-prior. The higher up we go in the Bayesian hierarchy, the less sensitive are the results to the prior settings. We can make the hyper-prior uninformative.

Да одиме секогаш со повеќе ниво, ако не го знаеме тоа одиме уште еден чекор понатаму, работата е во тоа што колку повеќе влегуваме во длабочина и ставаме хипер параметри тоа одење во длабочина станува не чувствителен во промена на параметрите и тоа нас помалку

Целата работа на Бејсиан е вклучување на приор и некогаш не е многу јасно, како би го вклучиле или како би работеле со него. Целата таа приор likelihood, каде имаме на повеќе нивоа на параметри. Тоа оди итеративно, ние викаме дека тоа се случајни променливи и каква ќе биде дистрибуцијата за нив од податоците на едно ниво. Кога правиме интегрираме и пресметуваме маргинални вер, многу се битни параметрите, во духот на бејсиан е да претставиме на приорот приор

ни овозможува кога станува збор за хипер приори да земеме неинформиран приор, оној кој што ништо не ни кажува. Кога пресметуаме маргинален Likelihood види претходна слика, прво имаме една длабочина повеќе, што значи на приорот му поставуваме приор, хипер параметрите итн. Целата филозофија е дека секогаш претпоставуваме спецификација на хиперприорот значи ние не знаеме што е приорот за обичниот, и не знаеме какви ќе се омега параметрите и ќе речиме окај, ние не сакаме да ги третираме како параметри ќе ги третираме како сл.променливи, за нив ќе претпоставиме дека н треба приор и затоа викаме на приорот приор, тие ги моделираме со параметри, тие што се ниво погоре или подоле се хипер параметри итн, и потоа интегрираме го правиме по просторот на сите параметри. Јасно ако ние за овој приор обичниот за омега параметрит за кои сме рекле дека с случајни па сме отишле едно ниво подоле сме претпоставиле дека може да го моделираме тоа со некој приор нас ни треба спецификација на тој хупер приор, тоа не може да оди до бескрај, духот на бајес е колку одиме подлабоко толку подобро.

A computational shortcut is to optimize α rather than integrating it out. That is, we use

$$\begin{aligned} p(\mathcal{D}|m) &\approx \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w}|\hat{\alpha}, m)d\mathbf{w} \\ \hat{\alpha} &= \operatorname{argmax}_{\alpha} p(\mathcal{D}|\alpha, m) \\ &= \operatorname{argmax}_{\alpha} \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w}|\alpha, m)d\mathbf{w} \end{aligned}$$

Еден од на чините е да го оптимизираме алфа а не да го ставаме во интегралот, хипер параметарот алфа да не го интегрираме туку да го максимизираме со MAP.

This approach is called empirical Bayes (EB).

Bayes factors

Suppose our prior on models is uniform, $p(m) \propto 1$.

Then model selection is equivalent to picking the model with the highest marginal likelihood.

Now suppose we just have two models: the null hypothesis, M_0 , and the alternative hypothesis, M_1 .

Define the Bayes factor as the ratio of marginal likelihoods:

$$BF_{1,0} = \frac{p(\mathcal{D}|M_1)}{p(\mathcal{D}|M_0)} = \frac{p(M_1|\mathcal{D})}{p(M_0|\mathcal{D})} / \frac{p(M_1)}{p(M_0)}$$

If $BF_{1,0} > 1$ then we prefer model 1, otherwise we prefer model 0.

This is a Bayesian alternative to the frequentist concept of a *p*-value

Разгледуваме два модели, нулта хипотеза и алтернативната, во тој случај се дефинира однос помеѓу likelihoods marginal.

Example: Testing if a coin is fair

Suppose we observe some coin tosses, and want to decide if the data was generated by a fair coin, $\theta = 0.5$, or a potentially biased coin, where θ could be any value in $[0, 1]$.

Let us denote the first model by M_0 and the second model by M_1 .

The marginal likelihood under M_0 is simply

$$p(\mathcal{D}|M_0) = \left(\frac{1}{2}\right)^N$$

where N is the number of coin tosses.

The marginal likelihood under M_1 , using a Beta prior, is

$$p(\mathcal{D}|M_1) = \int p(\mathcal{D}|\theta)p(\theta)d\theta = \frac{B(\alpha_1 + N_1, \alpha_0 + N_0)}{B(\alpha_1, \alpha_0)}$$

We plot $\log p(\mathcal{D}|M_1)$ vs the number of heads N_1 , assuming $N = 5$ and $\alpha_1 = \alpha_0 = 1$.

The shape of the curve is not very sensitive to α_1 and α_0 , as long as $\alpha_0 = \alpha_1$.

If we observe 2 or 3 heads, the unbiased coin hypothesis M_0 is more likely than M_1 , since M_0 is a simpler model (it has no free parameters) – it would be a suspicious coincidence if the coin were biased but happened to produce almost exactly 50/50 heads/tails.

However, as the counts become more extreme, we favor the biased coin hypothesis.

Priors

19;30

Фрекфентистичка статистика

Принципот каде што не употребуваме т.е не ги третираме параметрите како случајни променливи и каде што во избегнуваме да користиме приор и Бајесово правило, таквиот пристап се нарекува фрекфентистичка статистика. Наместо да се базираме на постериорната дистрибуција тука се базираме на дистрибуцијата на примерокот, често овој пристап е наречен класичен пристап т.е класична статистика. Ова е идејата од варијации над повторувачки случајни настани што го прават бајесовиот пристап за поделирање несигурен, се употребува кај фрекфентистичка статистика.

Sampling distribution of an estimator

Во фрекфентистичка, естимација на некој параметар тета-капа се пресметува со естиматорот δ над некој дата сет D , па $\hat{\theta} = \delta(D)$. Параметарот се гледа како фиксен а податоците се случајни. Несигурноста на параметарот може да се измери со пресметување на дитрибуцијата на примерокот на естиматорот. Да замислим дека семплираме различни дата сетови $D^{(s)}$ from some true model, $p(\cdot | \theta^*)$, i.e., let $D^{(s)} = \{x_i^{(s)}\}_{i=1}^N$, where $x_i^{(s)} \sim p(\cdot | \theta^*)$, and θ^* е вистинскиот параметар. Нека $S=1$, S индексира семплираниот дата сет и N е големината на секој дата сет, сега го употребуваме естиматорот $\hat{\theta}(\cdot)$ to each $D^{(s)}$ to get a set of estimates, $\{\hat{\theta}(D^{(s)})\}$. Ако S тежи кон бескрај дистрибуцијата добиена од $\hat{\theta}(\cdot)$ е sampling distribution of the estimator.

Bootstrap

Бутстррап е едноставна Монте Карло техника за апроксимирање на семплирачката дистрибуција. Ако го знаеме вистинскиот параметар θ^* може да генерираме S вештачки дата сетови, секој со големина N , од вистинската распределба.

$$x_i^s \sim p(\cdot | \theta^*), s = 1 : S, i = 1 : N.$$

Тогаш можеме да го пресметаме естиматорот за секој примерок $\hat{\theta}^s = f(x_{1:N}^s)$ и даја искористиме емпириската распределба на резултирачките примероци како наши проценки за семплирачката распределба. Со тоа што тета не ни е познато, идејата на параметарски бутстррап е да ги генерираме примероците користеќи $\hat{\theta}(\hat{D})$. Не параметарскиот бутстррап сеплирај x_i^s со замена од оригиналниот дата сет и тогаш пресметај добиената распределба

Frequentist decision theory

Во фрекфентистичка теорија, нема функција на загуба и лајклихуд, а со тоа нема ни приор ни постериор, нема загуба на постериорот. Избирајме било кој естиматор или процедура на одлука $\delta : X \rightarrow A$. Ние го дефинираме очелкуваниот губиток или ризик како

$$\begin{aligned} R(\theta^*, \delta) &= \mathbb{E}_{p(\hat{D}|\theta^*)} [L(\theta^*, \delta(\hat{D}))] \\ &= \int L(\theta^*, \delta(\hat{D})) p(\hat{D}|\theta^*) d\hat{D} \end{aligned}$$

Следи низа от слики на слайдови:

Каде D^* податоците семплирани од “природната распределба”, што е репрезентирано со тета*. Со други зборови очекувањето е со почит кон семплирачката дистрибуција на естиматорот.

- ▶ Expected loss or risk:

$$\begin{aligned} R(\theta^*, \delta) &= \mathbb{E}_{p(\hat{D}|\theta^*)} [L(\theta^*, \delta(\hat{D}))] \\ &= \int L(\theta^*, \delta(\hat{D})) p(\hat{D}|\theta^*) d\hat{D} \end{aligned}$$

where \hat{D} is data sampled from “nature’s distribution”, which is represented by parameter θ^* .

- ▶ Bayesian posterior expected loss:

$$\rho(a|\mathcal{D}, \pi) = \mathbb{E}_{p(\theta|\mathcal{D}, \pi)} [L(\theta, a)] = \int_{\theta} L(\theta, a) p(\theta|\mathcal{D}, \pi) d\theta$$

- ▶ The Bayesian approach averages over θ (unknown) and conditions on \mathcal{D} (known), whereas the frequentist approach averages over \hat{D} (ignoring the observed data), and conditions on θ^* (unknown).

Bayes risk

Очекуваната загуба не може да биде пресметата затоа што тета* е непознато. Не може да споредуваме различни естиматори со нивните фрекфентистички ризици. Бајесов ризик или интегриран ризик на естиматор.

Expected loss cannot be computed, because θ^* is unknown.

We cannot compare different estimators in terms of their frequentist risk.

Bayes risk or integrated risk of an estimator:

$$R_B(\delta) = \mathbb{E}_{p(\theta^*)} [R(\theta^*, \delta)] = \int R(\theta^*, \delta) p(\theta^*) d\theta^*$$

A Bayes estimator or Bayes decision rule is one which minimizes the expected risk:

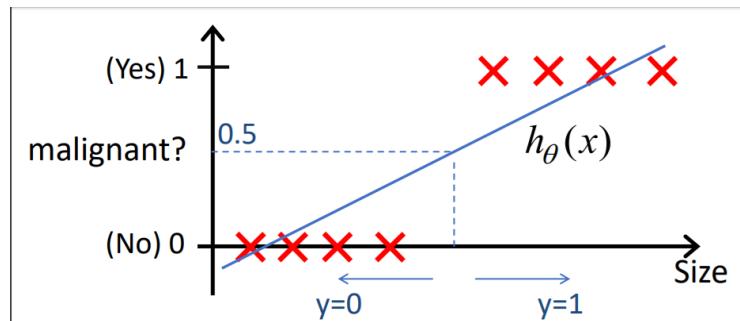
$$\delta_B = \operatorname{argmin}_{\delta} R_B(\delta)$$

The following theorem connects the Bayesian and frequentist approaches to decision theory.

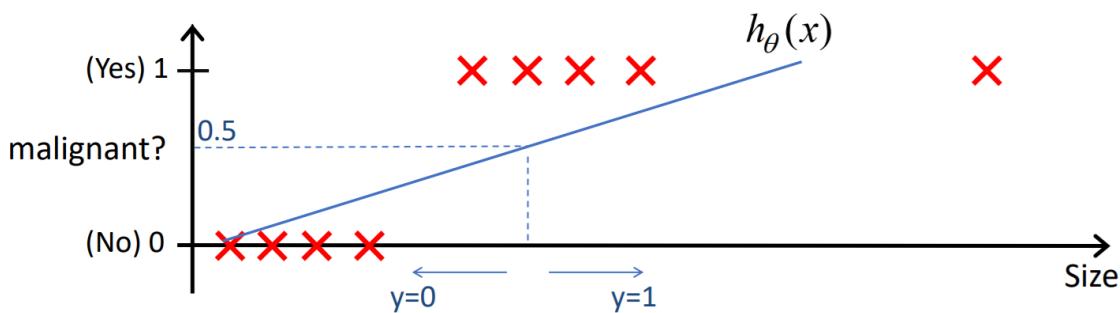
Theorem. A Bayes estimator can be obtained by minimizing the posterior expected loss for each x .

LECTURE 4 Logistic regression/Логистичка регресија

Безразлика името што вика регресија, сепак се работи за класификација, значи логистичката регресија како резултат не дава т.е y – што го предвидуваме не е реален број туку е 0 или 1 односно или припаѓа или не припаѓа на одредена класа. Досега генерално моделите што ги работевме користеја генеративен модел т.е пристап односно прво се креира здружена веројатност $p(y, x)$ а врз основа на тоа се изведува условната веројатност, да биде y – ако се дадени карактеристиките x . Сега ќе зборуваме за дискриминативниот пристап односно ќе креираме модел во формата $p(y|x)$ без здружената веројатност ќе го добиеме y – за дадени карактеристики x , и ќе претпоставиме дека овие дискриминативни модели се линеарни по нивните параметри, т.е се во поедноставна верзија. Кај логистичка регресија треба да добиеме резултат дали една порака е спам или не, дали одредена трансакција е измама или не, дали туморот е малиген или белиген врз основа на некој карактеристики да ги добиеме тие резултати, тие можат да се 0 или 1 односно припаѓа на позитивна или на негативна класа. Е сега што ќе се случи ако примениме линеарна регресија на овој проблем, прво ќе добиеме реален број меѓутоа, и ќе поставиме одреден праг на излезот на класификаторот така што ако резултатот од нашата хипотеза на моделот е поголем од 0,5 да предвидиме дека припаѓа на класа 1 ако не да предвидиме дека припаѓа на класа 0.



На цртежот ја имаме хипотезата од линеарната регресија и прагот, па се што е од десна страна припаѓа на h_0 и обратно. И за ова податочно множество ова изгледа океј, но меѓутоа што ако додадеме некој податок како на цртежот долу.



Е сега хипотезата од линеарна рег. што ќе ја добијеме сега ќе биде поместена и сега ако го поставиме прагот на 0,5 гледаме дека не е добар класификатор, значи во суштина во претходниот пример имавме среќа иначе генерално линеарна регресија не се користи за класификација. Друга работа е тоа што излезот од хипотезата е некој број кој не се движи од нула до 1 а нас ни треба број кој се движи од 0 до 1 што ќе ни претставува веројатноста да ни припаѓа во таа класа. Сакаме да добијеме 0 или 1 и таа хипотеза т.е логистичката регресија да биде помеѓу 0 и 1. Е сега за таа цел линеарната регресија т.е ќе го наречеме стариот модел користи следната хипотеза:

$$h_{\theta}(x) = \theta^T x$$

каде тета ни се параметрите по x, ако имаме линеарна p, го користиме ова и прашањето е само кои се вредностите на тета. Новиот модел т.е логистичка регресија ја користи следната хипотеза:

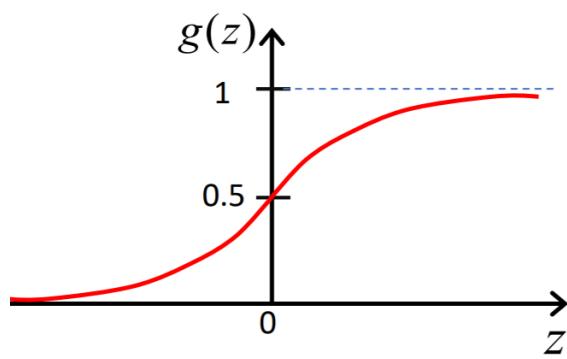
$$h_{\theta}(x) = g(\theta^T x)$$

е сега овде во новиот модел ќе биде пак линеарна регресија на параметрите значи тета транспонирано по x ама од тета ќе земеме одредена функција g е сега прашањето е кое е g, значи бараме некоја функција која што ќе биде во опсегот од 0 до 1 значи ќе биде нешто што ќе врати резултат од опсегот од 0 до 1, значи нешто што ќе ни врати резултат во опсегот од 0 до 1 затоа што тоа е нас што моментално не интересира. Таа функција g која што се користи кај логистичката регресија е сигмоидална функција или логистичка функција.

Uses sigmoid (logistic)
function $g(z) = \frac{1}{1 + e^{-z}}$

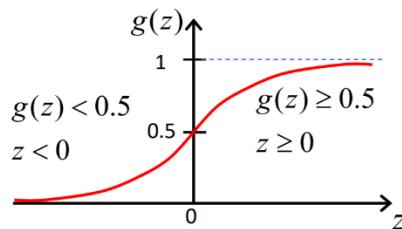
$$z = \theta^T x$$

$$g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$



Оваа логистичка регресија, се што се зборува во оваа лекција не важи само за логистичката регресија туку и за невронските мрежи, бидејќи логистичка регресија претставува невронска мрежа со еден слој, така да е многу важен концепт не само дека логистичката регресија како таква ќе ја користиме туку и за понатаму се што ќе се зборува важи и за невронските мрежи. Е сега да ја видиме сигмоидалната функција, значи таа на опсегот од 0-1. За излезот ако резултатот од л.г е проценка на веројатноста дека у = 1 ќе даде, значи не е сигмоидалната функција 1 туку некој број помеѓу 0 и 1 и ние тоа го викаме проценка на веројатноста дека у = 1, пример добиваме дека е 0,7 тогаш тоа значи дека у = 1 за дадено x.

Сега да ја разгледаме сигмоидалната функција, значи кога сигмоидалната функција ќе предвиди 1 ако хипотезата е $\geq 0,5$, таа е поголема во опсегот на цртежот.



Predict "y=0" if $h_\theta(x) < 0.5$

$$g(\theta^T x) < 0.5$$



$$\theta^T x < 0$$

Значи ако има $g(z)$, тоа е поголемо од 0,5 кога z е позитивно т.е поголемо од или еднакво од нула или од друга страна сигмоидалната функција е помала од 0,5, тогаш имаме вредност на g и тоа е кога $z<0$, ова z ние нас во суштина тета транспонирано по x . Е сега што ќе биде тоа тета транспонирано во овој случај на класификација, овде користиме сигмоидална функција од тетат транспонирано по x , е сега ако имаме податочно множество кај што имаме две класи и хипотезата е еднаква на сигмоидалната функција , т.е некоја линеарна комбинација:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Е сега да ги претпоставиме вредностите на тета:

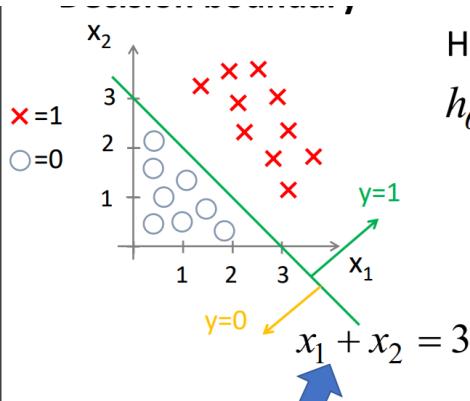
$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

со некој метод на оптимизација сме добиде дека овие се оптималните вредности на тета и ако замениме во равенката добиваме:

Predict "y=1" if $-3 + x_1 + x_2 \geq 0$

$$x_1 + x_2 \geq 3$$

, оваа равенка е права која била границата на одлука.

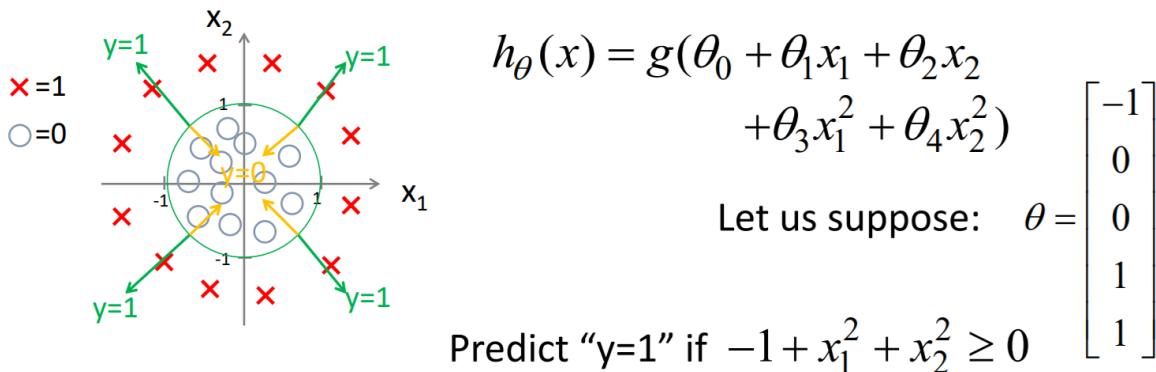


Decision boundary (place of distinction between $y = 0$ and $y = 1$)

$$h_\theta$$

Значи заклучивме она што ни е внатре во сигмоидалната функција е функција на одлука. И сега викаме дека таа е граница која го дели податочното множество на две класи, е сега викаме ако $x_1+x_2 \geq 0$ предвиди дека $y=1$, за сите примероци, ако е обратното тогаш $y=0$. Значи сега не ни е хипотезата права туку правата во овој случај за разлика од линеарната регресија ни е граница на одлука и од ова земаме сигмоидална функција, сигмоидалната функција во суштина ќе ни каже она што претходно го зборувавме, ќе предвиди 1 кога тета транспонирано по x е ≥ 0 , и обратно. Сега имаме граница на одлука и од таа земаме сиг. Ф.

која што ни кажува ако таа граница е поголемо од 0 се предвидува едната класа или другата. Еве ако имаме други параметри нелинеарна граница на одлука.



Ова е генерално претставување на логистичка регресија и во однос на линеарната регресија имаме две промени, првата промена е тоа што ќе ја замениме Гаусовата дистрибуција со Бернулиева дистрибуција бидејќи у нас ни е 0 или 1, и второ она што го зборувавме треба да се пресмета линеарна комбинација на влезот исто како претходно таа линеарна комбинација на влезот се поминува низ одредена функција која што ќе ни оценува дека излезот ни е помеѓу 0 и 1 а таа функција е сигмоидалната функција. Ако ги поврзиме двете работи добиваме:

The sigmoid function is defined as

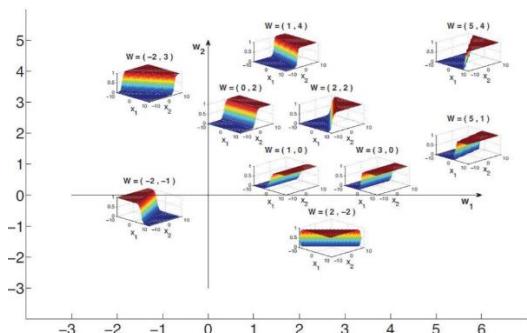
$$\text{sigm}(\eta) \triangleq \frac{1}{1 + \exp(-\eta)} = \frac{e^\eta}{e^\eta + 1}$$

Putting these two steps together we get

$$p(y|x, w) = \text{Ber}(y | \text{sigm}(w^T x))$$

This is called **logistic regression** due to its similarity to linear regression (although it is a form of classification, not regression!).

И покрај името тоа е класификација но не регресија, т.е класифицираме примероци кои одлучуваме дека припаѓаат на една или на друга класа. Овој пример на даден влез на сигмоидалната функција за различни вредности на параметрите и тука исто ако ставиме граница од 0,5 добиваме линеарна граница на одлука, тука е претставена сигмоидалната функција за две класи.



Сега ни треба алгоритам, досега викавме дека некои вредности на тета ни се дадени е сега прашањето е како да ги најдеме тие вредности на тета, како да ја најдеме оптималната граница т.е оптималните вредности на тета кои што ќе ни минимизираат одредена функција на грешка. Затоа прво ни треба функција за грешка, за да видиме во кој правец треба да се движиме. Најчесто се користи MLE во суштина се зема негативниот логаритам од лајклиходт за логистичка регресија значи конкретно дека зборуваме за бернулива дистрибуција.

MLE – maximum likelihood estimation

- The negative log-likelihood for logistic regression is given by

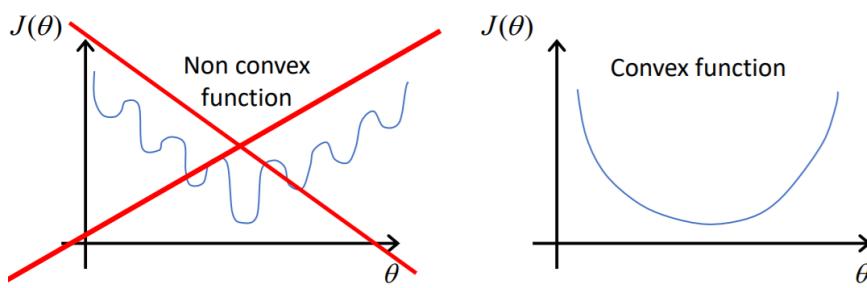
$$\begin{aligned} \text{NLL}(w) &= - \sum_{i=1}^N \log[\mu_i^{\mathbb{I}(y_i=1)} \times (1 - \mu_i)^{\mathbb{I}(y_i=0)}] \\ &= - \sum_{i=1}^N [y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i)] \end{aligned}$$

Ова функција е исто така наречена cross-entropy функција на грешка. Тука е проблем тоа што за разлика од линеарната регресија не може да се напише MLE во затворена форма, што значи дека треба да имаме нов начин за оптимизација за наоѓање на оптималните вредности. Тоа ќе го правиме во суштина со итерации. Но прво да видиме што значи функцијата за грешка што се користи тута, претходно кај линеарна регресија најчесто се користи средна квадратна грешка но сега што ако примениме средна квадратна грешка, имаме линеарни параметри, но што ако ја искористам оваа функција за логистичка регресија, важно тука за функцијата на грешка е тоа што треба да правиме оптимизацији по таа функција на грешка, што значи треба да бараме минимум на таа функција на грешка, но за да најдеме минимум многу е полесно да најдеме минимум ако таа функција е конвексна ако не е конвексна ако не е конвексна имаме проблем со заглавување во повеќе минимуми тогаш се комплицира, дали ако ја искористам оваа функција на грешка за логистичка регресија ќе имаме проблем, да добиваме неконвексна функција. Во овој случај бараме некоја функција која што ќе биде конвексна онаа претходно што ја пресметавме за MLE во суштина е конвексна функција и тогаш нема заглавување во минимуми. Да видиме дал таа функција на цена на чинење беше составена од два дела.

Cost function

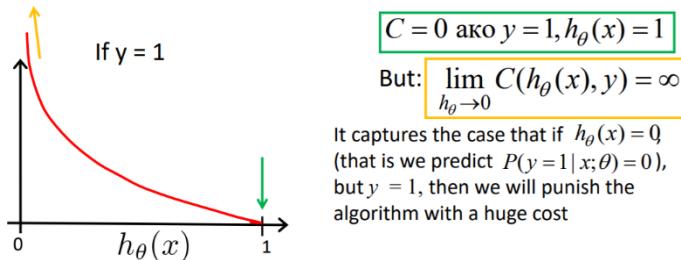
Linear regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$

Cost: $C(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$



Cost function for logistic regression

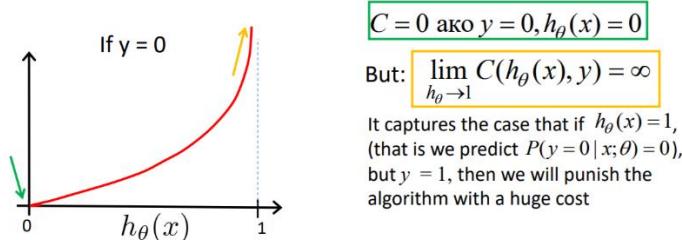
$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{ако } y = 1 \\ -\log(1 - h_\theta(x)) & \text{ако } y = 0 \end{cases}$$



Тука заклучуваме ако цената е најмала тогаш класата е 1 а како почнува да носи грешна одлука цената се зголемува, т.е се казнува моделот. Истото е и за обратниот случај.

Cost function for logistic regression

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{ако } y = 1 \\ -\log(1 - h_\theta(x)) & \text{ако } y = 0 \end{cases}$$



КОГА ТОЧНО ЌЕ ПРЕДВИДИ ТОГАШ ЦЕНАТА Е НУЛА, КОГА ГРЕШНО ЌЕ ПРЕДВИДИ ТОГАШ ЦЕНАТА Е ГОЛЕМА, ГО КАЗНУВА МОДЕЛОТ.

Сега кога ќе ги споиме овие две видувања на функцијата добиваме:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m C(h_\theta(x^{(i)}), y^{(i)})$$

$$C(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



$$C(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

Е сега бидејќи не може да се запиши MLE во затворена форма, тогаш треба да имаме некој оптимизациски алгоритам за да ги пресметаме оптималните тета. За тоа ни треба **градиентот и хезиан** матрицата, значи сите тие оптимизациски алгоритми кои се итеративни тие почнуваат од некој иницијални вредности на тета и потоа се движат кон оптималното решение со помош на градиентот и хезиен патрицата.

$$\begin{aligned} \mathbf{g} &= \frac{d}{d\mathbf{w}} f(\mathbf{w}) = \sum_i (\mu_i - y_i) \mathbf{x}_i = \mathbf{X}^T (\boldsymbol{\mu} - \mathbf{y}) \\ \mathbf{H} &= \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^T = \sum_i (\nabla_{\mathbf{w}} \mu_i) \mathbf{x}_i^T = \sum_i \mu_i (1 - \mu_i) \mathbf{x}_i \mathbf{x}_i^T \\ &= \mathbf{X}^T \mathbf{S} \mathbf{X} \end{aligned}$$

where $\mathbf{S} \triangleq \text{diag}(\mu_i(1 - \mu_i))$

Н треба да е позитивна и конечна за да имаме конвексна функција, односно да имаме единствено глобално решение.

Када логистичка регресија треба да се користат вакви алгоритми кои што се генерално итеративни, почнуваме од едни тета вредности и итреираме и ги упдејтираме вредностите на параметрите така што се движими кон минимумот на цена на движење затоа ни треба градиентот кој што е првиот извод од функцијата, во суштина кој ни кажува во која насока треба да се движиме, паралелизмот во суштина ако имаме функција е тангентата во некоја точка и таа ни го дава правецот по кој што треба да се движиме за да стигнеме до минимумот. Тоа е првиот извод.

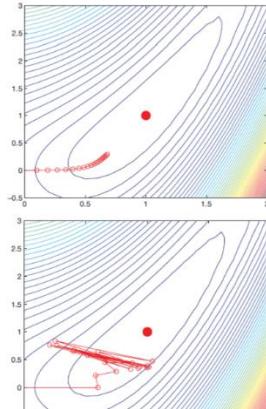
Steepest descent

- The simplest algorithm for unconstrained optimization is **gradient descent** or **steepest descent**: $\theta_{k+1} = \theta_k - \eta_k g_k$

where η_k is the step size or **learning rate**.

- The main issue here is: **how to set the step size?**
- Gradient descent on a simple function, starting from $(0, 0)$, for 20 steps, using a fixed learning rate (step size) η . The global minimum is at $(1, 1)$. (a) $\eta = 0.1$. (b) $\eta = 0.6$

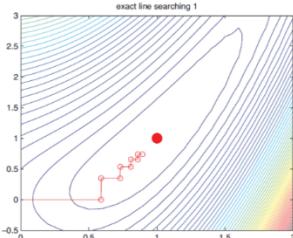
$$f(\theta) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$$



Е сега хесиен матрицата ми претставува прв извод на градиентот, вусшност јакутијан од градиентот. Тука добивавме тангента, но со хезиан матрицата добиваме како се менува таа тангента, врз основа на овие два параметри се базираат сите оптимизациски алгоритми, првиот и наједноставниот кој е поминат на вежби е Steepest descent овој е наједноставниот метод кој го зема во предвид апдејтирањето на параметрите тета минус н-ка по градиентот g од k . Градиентот ни кажува насоката на каде треба да се движиме за да го најдеме минимумот на функцијата. А ратата на учење ни кажува со колкав чекор треба да се движиме натаму. Одредувањето на чекорот е голем проблем, иако е конвексна функција, но ако ставиме погрешна рата на учење, алгоритмот може никогаш да не конвергира. На цртежот имаме дадено функција каде глобалниот минимум е во точката $(1, 1)$ и сега првиот пример е ако ратата на учење е поставена на 0.1 а вториот е ратата 0.6 , кога е мала ратата на учење гледаме дека чекорчињата стануваат се помали како се добиљуваме до минимумот, иако имаме константна рата како што се доближуваме до минимумот така оваа тангента има помала вредност односно наклон и се превземаат се помали чекори иако ратата ни е премногу мала ќе имаме премногу чекори за да стигнеме кај минимумот, бидејќи секогаш имаме ограничување на бројот на чекори може да не стигнеме до таму, а ако пак е

премногу гојлма тогаш превземаме чекои кои го прескокнуваат минимумот. Тука постојат два начини за одредување на оптималната вредност на одредување на ратата на учење, едниот е со проба. Генерално и многу алгоритми ова го користат, така да се пробуваат некои вредности, пример почнуваме од некоја поголема вредност и намалуваме се додека не ја дознаеме која е оптималната вредност, како ќе го дознаеме овој случај, кога функцијата на грешка во секој чекор се зголемува наместо да се намалува, па ако расти функцијата ја намалуваме ратата на учење се додека не дојдеме до ситуација која што ни ја намалува функцијата на грешка.

Steepest descent



- Picking the step size – let us develop more stable method that is guaranteed to converge to a local optimum no matter where we start (global convergence).
- Let us pick η to minimize $\phi(\eta) = f(\theta_k + \eta d_k)$ (**line minimization** or **line search**)
- To reduce the zig-zag effect a **momentum** term can be added

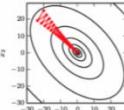
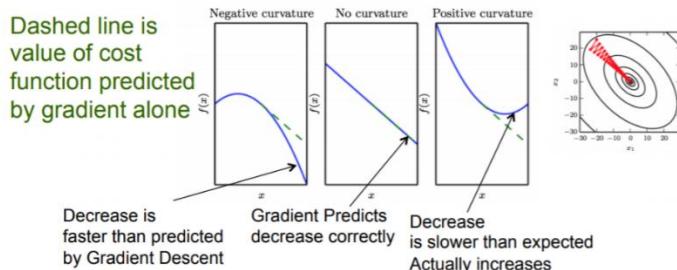
$$\theta_{k+1} = \theta_k - \eta_k g_k + \mu_k (\theta_k - \theta_{k-1})$$

where $0 \leq \mu_k \leq 1$ controls the importance of the momentum term

Првиот извод ако го земеме и го ставиме да е нула добиваме минимизација и тогаш се добива некакво привиджување до минимумот, плус може да се даде моментум кој ќе го редуцира овој зик-зак, со тоа што ќе имаме одреден параметар кој ја компромизира вредноста на овој моментум. Е сега покрај овој Градиент дисент, што ако земем втор извод кој во суштина ја мери и искривеноста, односно како се менува тангентата.

Second derivative measures curvature

- Derivative of a derivative
- Quadratic functions with different curvatures



Сега имаме друг метод, Ќутнов метод кој е исто така многу често корисен тој ја користи хезијан матрицата, значи со тоа што таа хезијан матрица во суштина ги зема и вторите изводи во предвид и со тоа ја зема и искривеноста на просторот, и овие се нарелени оптимизациски метододи од втор ред. Заради тоа што не се зема само градиентот туку и вторите изводи по различните параметри. Најосновен пример е Ќутновиот алгоритам и тој

е мајка на сите вакви алогиртми кои ги вклучуваат вторите изводи. Ќутновиот метод се користи многу, можеби не неговата основна форма, но неговите многубројни варијации, кога работиме со невронски мрежи оваа е методот кој најчесто би го користеле.

One can derive faster optimization methods by taking the curvature of the space (i.e. the Hessian) into account

These are called **second order** optimization methods

The primary example is the **Newton's algorithm** (mother of all second-order optimization algorithms)

This is an iterative algorithm which consists of updates of the form

$$\theta_{k+1} = \theta_k - \eta_k \mathbf{H}_k^{-1} \mathbf{g}_k$$

The derivation of the algorithm considers second-order Taylor series approximation of $f(\theta)$ around θ_k and finding Newton step d_k that minimizes the function

овде се зема и искривеноста на

просторот т.е хезијан матрицата и градиентот значи изведувањето на оваа равенка се состои од развивање на функција во нејзината апроксимација со помош на тајлорова серија од втор ред во точката тета к.

Algorithm 8.1: Newton's method for minimizing a strictly convex function

```

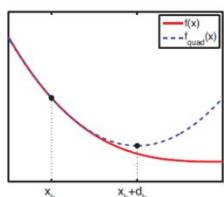
1 Initialize  $\theta_0$ ;
2 for  $k = 1, 2, \dots$  until convergence do
3   | Evaluate  $\mathbf{g}_k = \nabla f(\theta_k)$ ;
4   | Evaluate  $\mathbf{H}_k = \nabla^2 f(\theta_k)$ ;
5   | Solve  $\mathbf{H}_k d_k = -\mathbf{g}_k$  for  $d_k$ ;
6   | Use line search to find stepsize  $\eta_k$  along  $d_k$ ;
7   |  $\theta_{k+1} = \theta_k + \eta_k d_k$ ;

```

Значи во секој чекор сега покрај градиенот, ја пресметуваме и хезијан матрицата па го наоѓаме d_k кој што е њутновиот чекор т.е за колку треба да се зголеми сеха тета параметрите, кој d_k зависи и од хезијан матрицата и од градиентот, на истот начин може да се користи line streach и се апдејтираат тета-ка со помош на ратата на учење и овој ќутнов чекор. Сега пример

Newton's method – illustration for minimizing 1d function

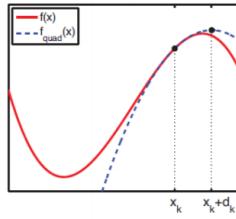
- (a) The solid curve is the function $f(x)$. The dotted line $f_{quad}(x)$ is its second order approximation at x_k . The Newton step d_k is what must be added to x_k to get to the minimum of $f_{quad}(x)$.



Е сега врз основа на ова се одредува минимумот на функцијата на таа функција што значи дека ова е чекорот кој ни кажува каде треба да се приближиме кон минимумот

Newton's method – illustration for minimizing 1d function

- (b) Illustration of Newton's method applied to a nonconvex function. We fit a quadratic around the current point x_k and move to its stationary point, $x_{k+1} = x_k + d_k$. Unfortunately, this is a local maximum, not minimum.
- In its simplest form Newton's method requires that H_k be positive definite, which will hold if the function is strictly convex. If not, the objective function is not convex, then d_k may not be a descent direction. In this case, one simple strategy is to revert to steepest descent, $d_k = -g_k$. The **Levenberg Marquardt** algorithm is an adaptive way to blend between Newton steps and steepest descent steps.



Заклучок Ќутновиот закон е соодветен кога имаме конвексна функција само! Други методи:

Iteratively reweighted least squares (IRLS) – Newton's algorithm applied to find MLE for binary logistic regression

$$\begin{aligned} w_{k+1} &= w_k - H^{-1} g_k \\ &= w_k + (X^T S_k X)^{-1} X^T (y - \mu_k) \\ &= (X^T S_k X)^{-1} [(X^T S_k X) w_k + X^T (y - \mu_k)] \\ &= (X^T S_k X)^{-1} X^T [S_k X w_k + y - \mu_k] \\ &= (X^T S_k X)^{-1} X^T S_k z_k \quad z_k \triangleq X w_k + S_k^{-1} (y - \mu_k) \end{aligned}$$

Quasi-Newton method – when it is too expensive to compute H explicitly. This method iteratively builds up an approximation of the Hessian using information extracted from the gradient vector at each step

Сега до сега ова што го зборувавме односно традиционалните методи се извршуваат offline имаме конечно множество на податоци и се применува алгоритмот на тоа конечно множество. Е сега може да примаме цело време влезови без да се чека крај, и тие податоци на некој начин да ги вклопиме, или пак да да имам конечно множество но да биде преголемо за да се стави во главната меморија, па можеме да го земеме тоа како бесконечно множество, ако претпоставивме дека во секој чекор се добива нов влез k , така што треба метододот за учење треба да ги пронајде параметрите врз овој примерок во теоријата на машинско учење има функција наречена функција на жалење која во суштина претставува разлика помеѓу просечната загуба и најдобриот избор кој можеме да го направиме.

Suppose that at each step, “nature” presents a sample z_k and the “learner” must respond with a parameter estimate θ_k . In the theoretical machine learning community, the objective used in online learning is the **regret**, which is the averaged loss incurred relative to the best we could have gotten in hindsight using a single fixed parameter value:

$$\text{regret}_k \triangleq \frac{1}{k} \sum_{t=1}^k f(\theta_t, z_t) - \min_{\theta^* \in \Theta} \frac{1}{k} \sum_{t=1}^k f(\theta^*, z_t)$$

For example, imagine we are investing in the stock-market. Let θ_j be the amount we invest in stock j , and let z_j be the return on this stock. Our loss function is $f(\theta, z) = -\theta^T z$. The regret is how much better (or worse) we did by trading at each step, rather than adopting a “buy and hold” strategy using an oracle to choose which stocks to buy.

Online gradient descent

- at each step k , update the parameters using

$$\theta_{k+1} = \text{proj}_{\Theta}(\theta_k - \eta_k g_k)$$
- where $\text{proj}_V(v) = \arg\min_{w \in V} \|w - v\|_2$ is the projection of vector v onto space V , $g_k = \nabla f(\theta_k, z_k)$ is the gradient, and η_k is the step size. (The projection step is only needed to ensure that the obtained solution stays within the feasible domain)
- projection of v back into V , i.e. it is the closest point (under the L_2 norm) in V to v

Stochastic gradient descent е сличена како Градиент но со таа разлика што, ги распределуваме по случајнос тподатоците и земаме по секој чекор еден примерок за да ги пресметаме параметрите а досега ги земавме сите можни примероци за пресметување на параметрите во секој чекор а тута се зема еден случаен еден примерок, и во зависност од примерокот може да залутаме од минимумот заради тоа може да трае малце повеќе. Од друга страна бидејќи не пресметуваме градиент на сите примероци, овде тоа е многу поедноставно.

Suppose we receive an infinite stream of samples from the distribution. One way to optimize stochastic objectives is to perform the update in the previous equation at each step.

Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example

It adds noise to the learning process that helps improving generalization error. However, this would increase the run time.

```
for i in range(num_epochs):
    np.random.shuffle(data)
    for example in data:
        grad = compute_gradient(example, params)
        params = params - learning_rate * grad
```

Setting the step size

- The set of values of η_k over time is called the learning rate **schedule**
- Various formulas are used such as $\eta_k = 1/k$
- The need to adjust these tuning parameters is one of the main drawbacks of stochastic optimization.
- One simple heuristic is as follows: store an initial subset of the data, and try a range of η values on this subset; then choose the one that results in the fastest decrease in the objective and apply it to all the rest of the data

c

Проблем овде е што мора да се постави ратат на учење во секој чекор, а множеството на вредности е наречено **schedule**, ова е недостаток.

SGD compared to batch learning

- If we don't have an infinite data stream, we can "simulate" one by sampling data points at random from our training set.
- A single pass over the entire data set is called an **epoch**.
- In this offline case, it is often better to compute the gradient of a **mini-batch** of B data cases. If $B = 1$, this is standard SGD, and if $B = N$, this is standard **steepest descent**. Typically $B \sim 100$ is used.

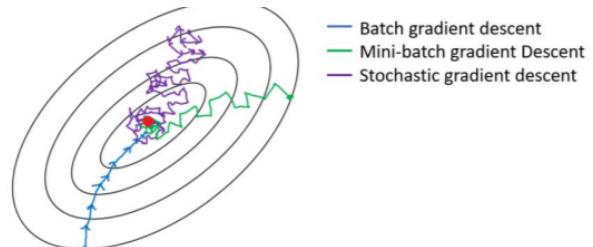
Algorithm 8.3: Stochastic gradient descent

```

1 Initialize  $\theta$ ,  $\eta$ ;
2 repeat
3   Randomly permute data;
4   for  $i = 1 : N$  do
5      $\mathbf{g} = \nabla f(\theta, \mathbf{z}_i)$ ;
6      $\theta \leftarrow \text{proj}_{\Theta}(\theta - \eta \mathbf{g})$ ;
7     Update  $\eta$ ;
8 until converged;

```

SGD compared to batch learning



- Although a simple first-order method, SGD performs surprisingly well on some problems, especially ones with large data sets
- The intuitive reason for this is that one can get a **fairly good estimate** of the gradient by looking at **just a few examples**. Carefully evaluating precise gradients using large datasets is often a waste of time, since the algorithm will have to recompute the gradient again anyway at the next step. It is often a better use of computer time to have a noisy estimate and to move rapidly through parameter space.
- As an extreme example, suppose **we double the training set** by duplicating every example. Batch methods will take twice as long, but online methods will be unaffected, since the direction of the gradient has not changed
- In addition to enhanced speed, SGD is often **less prone to getting stuck in shallow local minima, because it adds a certain amount of "noise"**. Consequently it is quite popular in the machine learning community for fitting models with non-convex objectives, such as neural networks and deep belief networks

The LMS algorithm

- As an example of SGD, let us consider how to compute the MLE for linear regression in an online fashion. The online gradient at iteration k is given by

$$\mathbf{g}_k = \mathbf{x}_i(\theta_k^T \mathbf{x}_i - y_i)$$

- where $i = i(k)$ is the training example to use at iteration k . If the data set is streaming, we use $i(k) = k$; we shall assume this from now on, for notational simplicity.
- The equation is easy to interpret: it is the feature vector \mathbf{x}_k weighted by the difference between what we predicted, $\widehat{y}_k = \theta_k^T \mathbf{x}_k$, and the true response, y_k ; hence the gradient acts like an error signal.

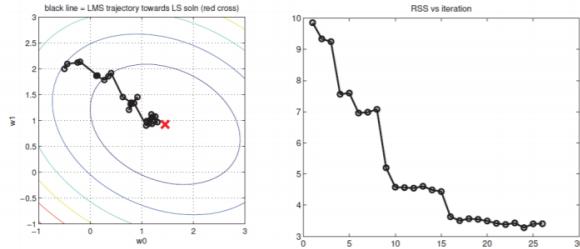
The LMS algorithm

- After computing the gradient, we take a step along it as follows:

$$\theta_{k+1} = \theta_k - \eta_k (\widehat{y}_k - y_k) \mathbf{x}_k$$

- This algorithm is called the **least mean squares** or **LMS** algorithm

- Left: we start from $\theta = (-0.5, 2)$ and slowly converging to the least squares solution of $\theta = (1.45, 0.92)$
- Right: plot of objective function over time. Note that it does not decrease monotonically.

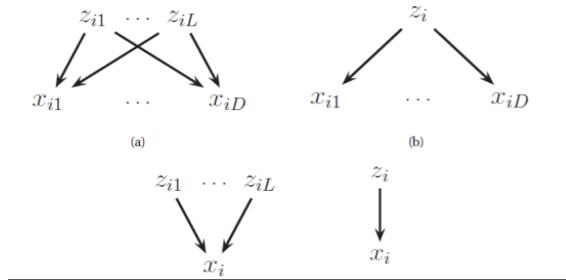


Online case for binary logistic regression

- Now let us consider how to fit a binary logistic regression model in an online manner.
 - In the online case, the weight update has the simple form
- $$\theta_k = \theta_{k-1} - \eta_k \mathbf{g}_i = \theta_{k-1} - \eta_k (\mu_i - y_i) \mathbf{x}_i$$
- where $\mu_i = p(y_i = 1 | \mathbf{x}_i, \theta_k) = \mathbb{E}[y_i | \mathbf{x}_i, \theta_k]$
- We see that this has exactly the same form as the LMS algorithm.

LECTURE 8 Mixture models and the EM algorithm

Сега прво ќе воведеме еден термин за скриени или латентни променливи, тие се во суштина не ги надгледуваме во податоците, на некој начин тие се скриени. Може да претпоставиме дека тие што се набљудувани променливи се колерираны поради некоја заедничка причина која што всушност е скриена, значи може да ја знаеме и може да не ја знаеме.



На пример имаме таква репрезентација на променливите, а, б, сликовита репрезентација на зависностите. Да речеме дека x -овите листовите претставуваат некакви симптоми на болест, а корените некакви причини како пушчење, начин на исхрана итн.

Е сега може покрај скриените симптоми да влијае и некоја скриена променлива која што на пример во овој случај може да е некоја болест на срцето која ние во овој случај не ја набљудуваме а од која што зависат медицинските симптоми, во овој случај може да имаме повеќе модели кои се познати како латентни модели или LVM, и тука може да имаме различни мапирања помеѓу скриените и набљудуваните променливи, значи може некоја од набљудуваните да зависи од повеќе скриени променливи, може да имаме сите набљудувани да зависат од една скриена променлива, една набљудувана да зависи од повеќе скриени и еден на еден т.е за секоја набљудувана да има посебна променлива.

Повеќето од моделите што ги гледавме се унимодални т.е имаат најмногу еден пик. Е сега некогаш во некоиреални ситуации може да ни се потребни повеќе модални функции на густина кој што ќе имаат повеќе типови е сега во тој случај еден вообичаен начин да се реши ова е да се креира мешан модел или Mixture Model кој што во суштина е сумата од комбинации од различни веројатностни густини.

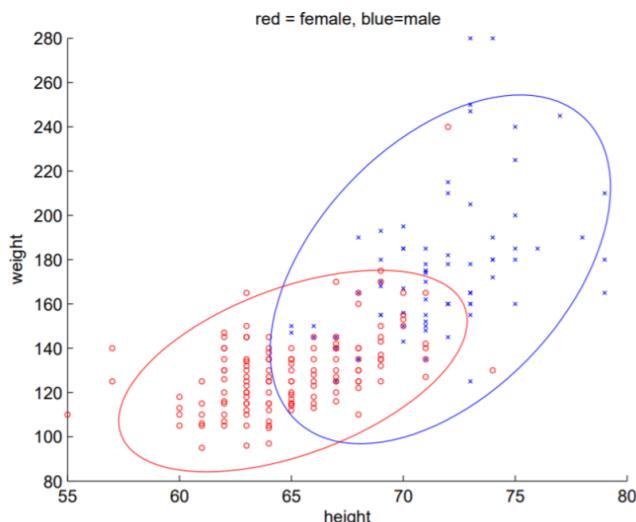
$$p(\mathbf{x}_i|\theta) = \sum_{k=1}^K \pi_k p_k(\mathbf{x}_i|\theta)$$

Значи имаме сумата од различни распределби помножени со одреден фактор на тежина значи K – е вкупниот број на компоненти на мешаниот модел, π – претставува тежината на секој од тие компонентите во межаниот модел, така да π е мешавина од 0-1, сумата по сите компоненти на π е 1, p_k – катата основна дистрибуција. Овде имаме различни променливи со различни дистрибуции и креираме мешан модел доколку ги сумираме сите тие со одредени тежини, најчесто најкористен модел е мешаниот модел од мешани променливи со гаусова распределба или Gaussians mixture modes (GMM). Во овој модел секоја дистрибуција е мешавина на мултиваријантна гаусова распределба и коваријансна матрица Σ_k .

$$p(\mathbf{x}_i|\theta) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

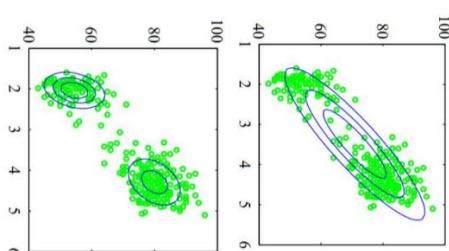
where $\theta = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ and K is the number of components

Значи тука имаме повеќе гаусови распределби кои што се комбинираат заедно со одредени тешини, сите тие гаусови распределби си имаат свои параметри, средна вредност K за соодветната компонента K и коваријансната матрица. Значи тука моделот се заменува со нормална дистрибуција за x_i ако се дадени параметрите за средната вредност и коваријансната матрица. Во овој модел сега не ни се параметри само во суштина средната вредност и коваријансната матрица туку и параметар исто кој треба да се најде е и π_k односно тежината на сите овие компоненти во мешаниот модел, значи имаме K компоненти и секоја функцијата на густина зависи од 3 параметри не само од 2 параметри – средна вредност и коваријансната матрица, туку дополнително и π_k . Е сега да видиме пример:



Доколку имаме податочно множество за висина и тежина на луѓе, е сега ако имаме висина и тежина на женски особи тогаш тие следат гаусова распределба исто и машките имат гаус. Но меѓутоа ако го комбинираме и машки и женски тогаш имаме комбинација на две гаусови распределби, значи тогаш ако сакаме, во рамките на еден кластер имаме една гаус распределба а другиот кастер друга гаус.

Но меѓутоа ако сакаме да ги комбинираме податоците во целина нас ни е потребен ваков мешан модел на гаусови распределби. Во оваа ситуација имаме набљудувани податоци или supervised учење бидејќи ја знаеме во суштина класата, имаме распределба машки и женски, но меѓутоа што ако не ја знаеме таа класификација имаме податоци само која немаат класа, како зелените податоци долу на сликата.



Тие немаат класа и сите се исто, не знаеме на која класа припаѓаат, ова се податоци за некој гејзер, податочно множество познато, на на хоризонталната оска е времетраењето во минута а вертикалната е на две последователни реакции во минути (сликата е свртена хоризонтално!!!) значи имаме гејзер и податоци за него

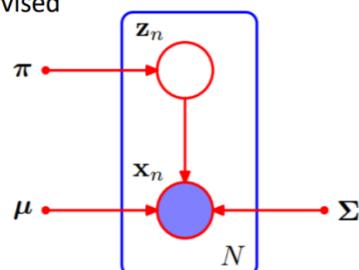
И сега тука бидејќи не ни се означени податоците ние на некој начин треба да најдеме можни кластери на овие податоци, и овде имаме не надгледувано учење затоа што не знаеме на крај, ако имаме нов податок јас не знам дека треба да се подели на две класи, овде прво не знаеме колку кластери ќе имаме, тоа е во суштина колку компоненти ќе имаме на мешаниот модел и исто така не знаеме како ќе бидат распределени податоците во тие кластери. Е сега овде да речиме во првиот случај имаме една гаусова распределба се гледа дека податоците не се добро фитувани на оваа една гаусова распределба ако ги поделиме на две тогаш добиваме многуподобро фитување на податоците тогаш овде во суштина може да кажеме дека се кластерирани во две групи. Една олеснителна околност за вакви мешани гаусови модели е да се замисли дека секоја податочна точка x_n е поврзан со скриена променлива z_n односно кажува дека т.е дефинира од која мешана компонента податокот доаѓа, т.е оваа z_n ни кажуваа од кој кластер доаѓа податокот x_n т.е за секој податок имаме соодветна асоцирана скриена променлива z_n која ни кажува на некој начин кластерот каде припаѓа податокот.

- These are **like the class labels** in a Bayesian classifier, except we assume the labels are hidden (since this is unsupervised learning).

- We can write the complete data log likelihood as

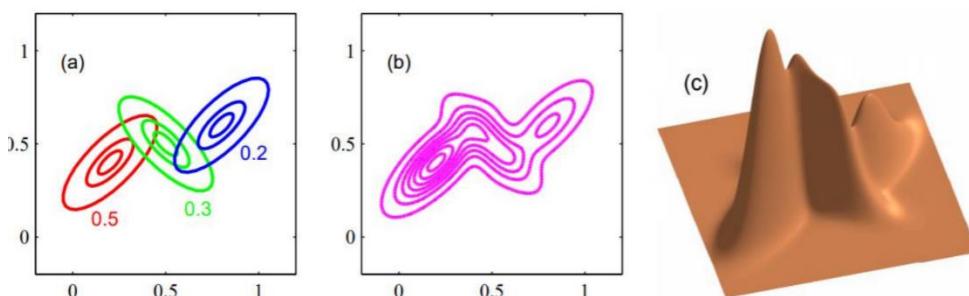
$$\begin{aligned}\ell_c(\theta) &= \log p(x_{1:N}, z_{1:N} | \theta) \\ &= \log \prod_n p(z_n | \pi) p(x_n | z_n, \theta)\end{aligned}$$

- Where $p(z_n = k | \pi) = \pi_k$ is a multinomial and $p(x_n | z_n = k, \theta) = \mathcal{N}(x_n | \mu_k, \Sigma_k)$ is a Gaussian



- A mixture of 3 Gaussians.

- (a) We show the contours of constant probability and the mixing weights.
- (b) A contour plot of the overall density.
- (c) A 3D surface plot of the overall density



Значи ова е контурите на веројатности и соодветно тежините за сите три кластери т.е 3 распределби, на це е 3Д површината за вкупната густина. Е сега ако додека имаме реални податоци користиме гаусова, но кога ќе дојде момент да имаме бинарни податоци тогаш наместо производ од гаусови ќе имаме производ од бернулиеви променливи така што сега тука имаме производ равенки од бернулиевата распределба, каде таа во суштина зависи μ_{jk} , т.е дека j -тиод бит се вклучува во соодветниот кластер k , и имаме Д-димензионални податоци.

- In this case, we can replace the Gaussian densities with a product of Bernoullis, and the class-conditional density is:

$$p(\mathbf{x}_i | z_i = k, \boldsymbol{\theta}) = \prod_{j=1}^D \text{Ber}(x_{ij} | \mu_{jk}) = \prod_{j=1}^D \mu_{jk}^{x_{ij}} (1 - \mu_{jk})^{1-x_{ij}}$$

- Where μ_{jk} is the probability that bit j turns on in cluster k , and the data consists of D -dimensional bit vectors

Е сега овие скриени променливи може и да немаат некое конкретно значење, може ние и да не знаеме точно што тие значат едноставно да ги користиме бидејќи моделот е помоќен, на пример може да се покаже дека средната вредност и коваријансата матрица се добиваа и се еднакви на овие две равенки.

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \sum_k \pi_k \mu_k \\ \text{cov}[\mathbf{x}] &= \sum_k \pi_k [\Sigma_k + \mu_k \mu_k^T] - \mathbb{E}[\mathbf{x}] \mathbb{E}[\mathbf{x}]^T\end{aligned}$$

where $\Sigma_k = \text{diag}(\mu_{jk}(1 - \mu_{jk}))$

Тука е важно што коваријансата матрица не е дијагонална матрица туку таа во суштина претставува ги фаќа и некои од корелациите од самите променливи, значи со помош на овој модел, без да знаеме кое е конкретното значење на тие променливи може да ги воведеме бидејќи тие на некаков начин може да ни доприносат за некоја скриена корелација помеѓу скриените променливи што ќе ни се даде во суштина од оваа коваријанса матрица, користењето на овие мешани модели најчесто се користи за кластерирање. Кластерирање е групирање на слични објекти заедно, разлилно е од класификација бидејќи кај класификација имаме точно колку класи и кои се тие класи, кај кластерирање само ги групирааме оние кои се најслични истанци. Ако имаме портал за вести и сите вести се класифицирани класифицирани во одредени класи како македонија, скопје, хроника, е сега во рамките на тие класи јас сакам да ги кластерирам тие вести кои се најслучни пример дека веста за цената на бензинот се покачила може да ја има во повеќе весници и јас сакам да ги кластерирам тие заедно во една вест. И сега јас незнам колку кластери ќе имаме и плус незнам кој се ќе припаѓа таму, јас само сакам да ги одредам во посебни групи оние кои имаат слична содржина. Сега ако користиме таков модел за кластерирање прво треба:

We first fit the mixture model, and then compute $p(z_i = k | \mathbf{x}_i, \boldsymbol{\theta})$ which represents the posterior probability that point i belongs to cluster k .

This is known as the **responsibility** of cluster k for point i , and can be computed using Bayes rule as follows:

$$r_{ik} \triangleq p(z_i = k | \mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k | \boldsymbol{\theta}) p(\mathbf{x}_i | z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^K p(z_i = k' | \boldsymbol{\theta}) p(\mathbf{x}_i | z_i = k', \boldsymbol{\theta})}$$

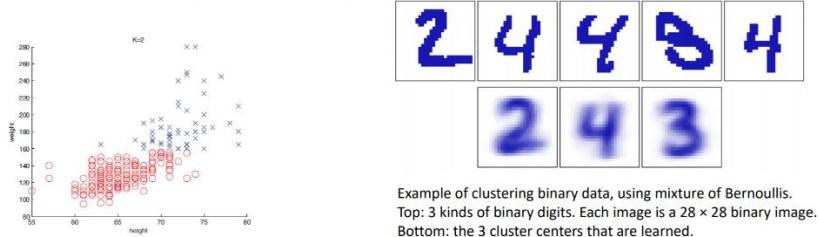
This procedure is called **soft clustering**

Всушност добивам веројатност дека точката x_i припаѓа на класата k . Soft clustering затоа што еден податок не припаѓа на точно еден кластер, туку има веројатност за припаѓање на секој од кластерите

Наспоти тоа има Hard clustering каде што сега го бараме параметарот К кој што го максимизира оваа функција на одговорност, односно го земам кластерот со најголема одговорност, тука имаме стрикно определн еден кластер, точно на еден, т.е тој со најголема одговорност.

- Hard clustering can be computed using the MAP estimate, using the following equation:

$$z_i^* = \arg \max_k r_{ik} = \arg \max_k \log p(\mathbf{x}_i | z_i = k, \theta) + \log p(z_i = k | \theta)$$



На примерот имаме податоци кои ги издвојуваме во два кластери и другиот пример е за мешавина од бернулиеви распределби. Всушностолните 3 класи се центрите кои што ги добиваме за тие 3 кластери, ова се оние μ_{jk} дека битот j ќе се уклучи во кластерот k .

До сега кажавме како да се пресмета постериорот на скриените променливи ако се дадени. Е сега прашањто е како да се научат овие параметри. Сега log likelihood

- The incomplete data log likelihood is

$$\begin{aligned}\ell(\theta) &= \log p(x_{1:N} | \theta) \\ &= \sum_n \log p(x_n | \theta) \\ &= \sum_n \log \sum_{z_n} p(x_n, z_n | \theta) \\ &= \sum_n \log \sum_{z_n=1}^K \pi(z_n) \mathcal{N}(x_n | z_n, \mu(z_n), \Sigma(z_n))\end{aligned}$$

Сега веќе немаме два параметри, туку имаме 3 параметри кои се зависни меѓу себе, бидејќи овие z_n се скриени тоа нас ни ја комплицира пресметката, на постериорите и на max likelihood. И тука имаме различен унимодален лајклиход е сега ако сакаме да ги маргинизираме по различните z_i добиваме мултимодален постериор, заради тоа што одговара на различното доделување лабели на различните кластери.

Not only do we have to estimate θ but also z_n .

Since z_n are hidden, the parameters are no longer independent, which complicates the computation of MAP and ML estimates.

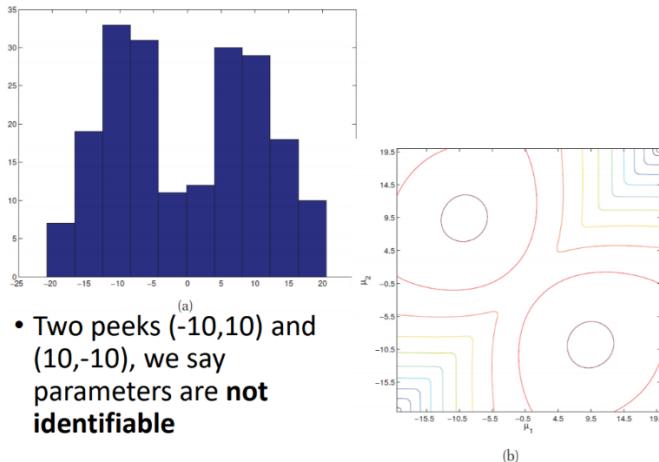
For each possible ways of “filling in” the z_i ’s, we get a different unimodal likelihood.

Thus when we marginalize out over the z_i ’s, we get a multi-modal posterior for $p(\theta | \mathcal{D})$

This modes correspond to different labeling of the clusters

Parameter estimation for mixture models

- $N = 200$ data points sampled from a mixture of 2 Gaussians in 1d, with $\pi_k = 0.5$,
- $\sigma_k = 5$, $\mu_1 = -10$ and $\mu_2 = 10$. Right: Likelihood surface $p(D|\mu_1, \mu_2)$, with all other parameters set to their true values. We see the two symmetric modes, reflecting the unidentifiability of the parameters.



There is not a unique MLE

Therefore there cannot be a unique MAP estimate and hence the posterior must be multimodal

Computing the MAP estimate is non-convex

Hard to find global optimum, most algorithm will find only local optimum. Which one they find, depends on where they start

In practice, we will run a local optimizer, perhaps using **multiple random restarts** to increase the chance of finding “good” local optimum

Careful initialization can help a lot, too, which is case-by-case sensitive

Сега следува алгоритмот кој се користи најчесто во ваквите ситуации, до сега во моделите пресметите на лајклихуд и постериори е лесно ако земемд дека ги надгледуваме сите променливи и го примаме цело податочно можество. Ако имаме податоци кои недостасуваат или имаме скриени податоци тогаш веќе се усложнуваат работите и тогаш имаме можност да пресметување, со оптимизатор кој се базира на градиент, и да се движиме кон оптималното решение, но во овој случај има многу пооптимални решение односно Use Expectation Maximization • Often much simpler in these cases • Simple iterative algorithm, often with closed-form updates at each step. Тоа упдејтирање не се базира на градиентот туку се базира на нешто друго, значи ако ги знаевме вредностите за z_i тогаш е лесно да се максимизира, главната идеја е да земеме некои проценки за параметрите т.е почетни вредности, и врз основа на тие да ги добие z_i променливите, врз основа на тие z_i да ги добиеме оптималните вредности за параметрите, и овој процес се повторува. Во суштина овој комплетен лајклихуд се заменува со очекуван целосен лајклихуд, тука воведуваме помошна функција која што во суштина ни го дава очекуваниот лајклихуд и се зема врз основа на добиените параметри и набљудуваните.

- Complete loglikelihood

$$l_c(\theta) = \sum_i \log p(x_i, z_i) = \sum_i \log p(z_i)p(x_i|z_i)$$

- Problem: z_i not known

- Possible solution: Replace with conditional expectation

- Expected complete loglikelihood

$$Q(\theta, \theta_{old}) = E[\sum_i \log p(x_i, z_i)]$$

Where Q is **auxiliary function**, and expectation is taken with respect to the old parameters and the observed data

Expectation Maximization Algorithm

1. Initialize θ .

2. Repeat until $\ell(\theta)$ stops changing

- (a) E step: compute $p(z_n|x_n, \theta^{old})$ for each case n .

- (b) M step: compute

$$\theta^{new} = \arg \max_{\theta} Q(\theta, \theta^{old})$$

where **auxiliary function** Q is the expected complete data log likelihood:

$$\begin{aligned} Q(\theta, \theta^{old}) &= \sum_z p(z|x, \theta^{old}) \log p(x, z|\theta) \\ &= \sum_{z_{1:N}} p(z_{1:N}|x_{1:N}, \theta^{old}) [\sum_n \log p(x_n, z_n|\theta)] \\ &= \sum_n \sum_{z_n} p(z_n|x_n, \theta^{old}) \log p(x_n, z_n|\theta) \end{aligned}$$

- (c) Compute the log likelihood

$$\ell(\theta) = \log \sum_n \sum_{z_n} p(z_n, x_n|\theta)$$

Се повторува постапката се до конвергенција т.е се додека нема промени, имаме два чекори во алгоритамот е expectation and the other is maximization, expectation пресметува веројатноста на податокот x_n со дадени тетат параметри θ^{old} т.е оние кои се иницијализирани вредности, веројатноста дека x_n припаѓа на кластерот z_n , за секој податок n , и потоа во максимизацискиот чекор пресметуваме нови тета параметри кои што сега имајќи во предвид распределбата на податоци во кластери ги пресметуваме најоптималните параметри за секој од тие кластери, значи барам тета вредностит кои ја максимизираат помошната функција, е сега во оваа равенка го земаме во предвид веројатноста z_n, x_n , ако тета θ^{old} , она што сме го пресметале во последниот чекор и сега врз основа на тоа ги наоѓаме новите параметри на тета, се додека нема промена во тета се повторува, т.е се додека не се менува логлајкихуд а тоа кое ќе престане да се менува значи сме ги добиле оптималните податоци по кластери

EM for GMM

Ова го користиме кога имаме мешавина на гаусови распределби, тута под претпоставка бројот на компоненти K се знае, во овие алгоритми K треба да се претпостави, првично треба да го зададеме, значи на почетокот како што кажавме ги иницијализираме средната вредност на компонентата K и ја додаваме на една од податоците, по случаен избор, а \sum_k да биде мала фракција од глобалната коваријанса, значи ги иницијализираме на почеток параметрите μ и Σ за секое од компонентите K и понатаму пресметуваме во е-чекорот односно expectation и врз основа на овие параметрите пресметуваме која е веројатноста да имаме x_n ако се дадени параметрите. Всушност ние пресметуваме responsibility која е веројатноста од секое од податоците да пропада на компонентата K т.е да припада на одреден кластер користејќи бајесова формула.

- The **E step** is to compute

$$p(z_n = k|x_n, \theta^{old}) = r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

where r_{nk} is called the **responsibility** of cluster k for data point n . (the probability that it belongs to cluster k)

normalized to sum to one (over clusters)

Следниот чекор е максимизација:

- The **M step**

- Maximizing the expected complete data log likelihood (Q) with respect to π and μ_k, Σ_k separately.

- For each cluster (Gaussian) update the parameters

- For π we have $\pi_k = \frac{1}{N} \sum_i r_{ik} = \frac{r_k}{N}$ Total responsibility allocated to cluster k

Fraction of total assigned to cluster k

where $r_k \triangleq \sum_i r_{ik}$ is the weighted number of points assigned to cluster k

- The estimates of the parameters μ_k and Σ_k are given by

$$\mu_k = \frac{\sum_i r_{ik} x_i}{r_k} \quad \text{Weighted mean of assigned data}$$

$$\Sigma_k = \frac{\sum_i r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{r_k} = \frac{\sum_i r_{ik} x_i x_i^T}{r_k} - \mu_k \mu_k^T \quad \begin{matrix} \text{Weighted covariance of assigned data} \\ (\text{use new weighted means here}) \end{matrix}$$

- After computing the new estimates, we set $\theta^t = (\pi_k, \mu_k, \Sigma_k)$ for $k = 1 : K$, and go to the next E step.

- Example

- We start with $\mu_1 = (-1, 1)$, $\Sigma_1 = I$, $\mu_2 = (1, -1)$, $\Sigma_2 = I$. We color code points such that blue points come from cluster 1 and red points from cluster 2. More precisely, we set the color to

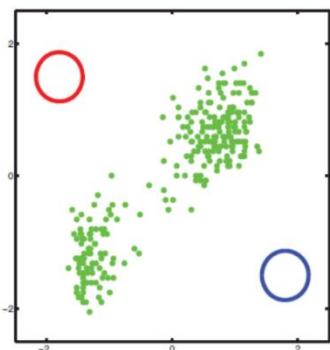
$$\text{color}(i) = r_{i1}\text{blue} + r_{i2}\text{red}$$

so ambiguous points appear purple.

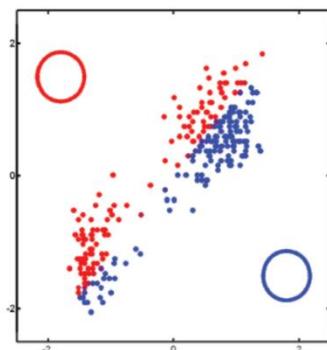
- After 20 iterations, the algorithm has converged on a good clustering. (The data was standardized, by removing the mean and dividing by the standard deviation, before processing. This often helps convergence.)

Во суштина да ги пресметаме новте μ , Σ , треба да ги најдеме оптапалните параметри.

Примерот на претходниот слайд, значи почуваме со некои податоци почнуваме со $\mu_1 = (-1, 1)$ и $\sum_1 = I$, $\mu_2 = (-1, 1)$. Примерот т.е податоците во него се работи за боја, таа боја ќе ја добиеме со равенката на слайдот, значи веројатноста да припаѓа на одредена класа по соодветната боја плус да припаѓа на друга класа по црвената боја (може било која така е на примерот). Значи првично имам претпоставка за параметрите е сега вторито чекор е да ги доделам сите овие податоци да се распределат во некоја од k -те компоненти, односно да пресметам која е веројатноста за секој од овие податоци да припаѓа на једниот или на другиот кластер.

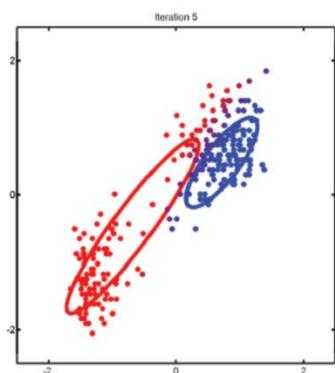


(a)

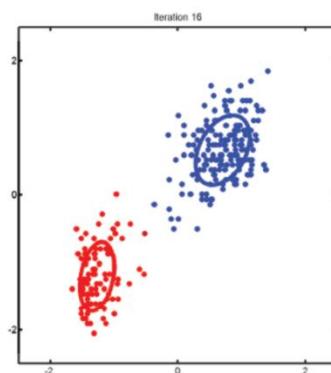


(b)

И врз основа на таа пресметка, гледаме дали може да припаѓа на калстер еден или на кластер 2 и кога ќе ги помножам тие две вредности со соодветната боја добивам на кој кластер ќе припаѓа секој од тие податоци. Првично горната половина припаѓа на горниот црвен кластер а долната половина на долните кластери оставијат, некои што се измеѓу имаат вилетови бои, т.е некои измеѓу бои од двете бои соодветно на калстерите.



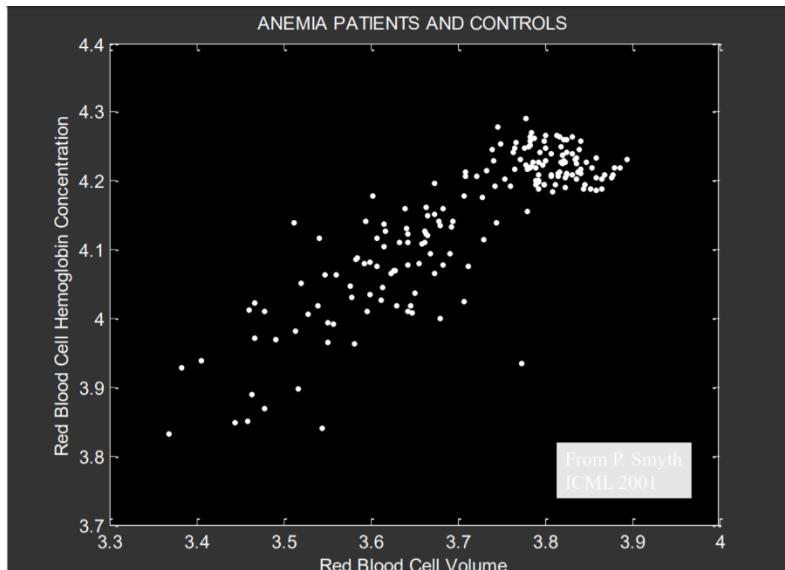
(e)



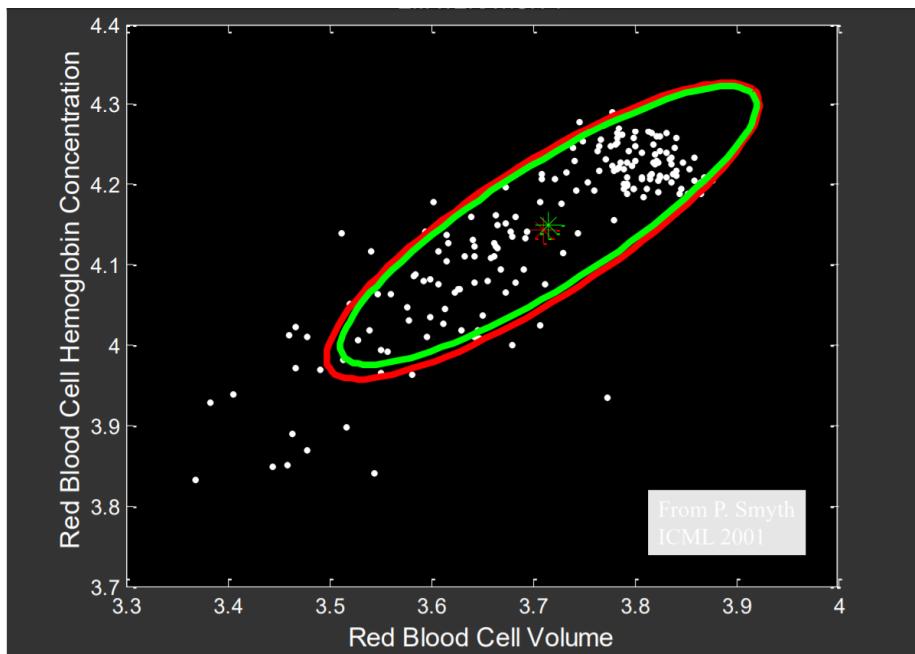
(f)

Во вториот чекор сега, т.е втората итерација, вообичаено ги земаме почетните вредности да бидат некој од примероците, и ќе почниме од е) ги пресметрувавме оптималните вредности за Σ , μ за секој од двата кластери кој што најдобро во суштина ги фитуваат податоците од двата кластери, значи сега еве како средната вредност се забележува на сликата каде што има повеќе примероци и коваријансата и сега имаќи ги во предвид овие два податоци за Σ ,

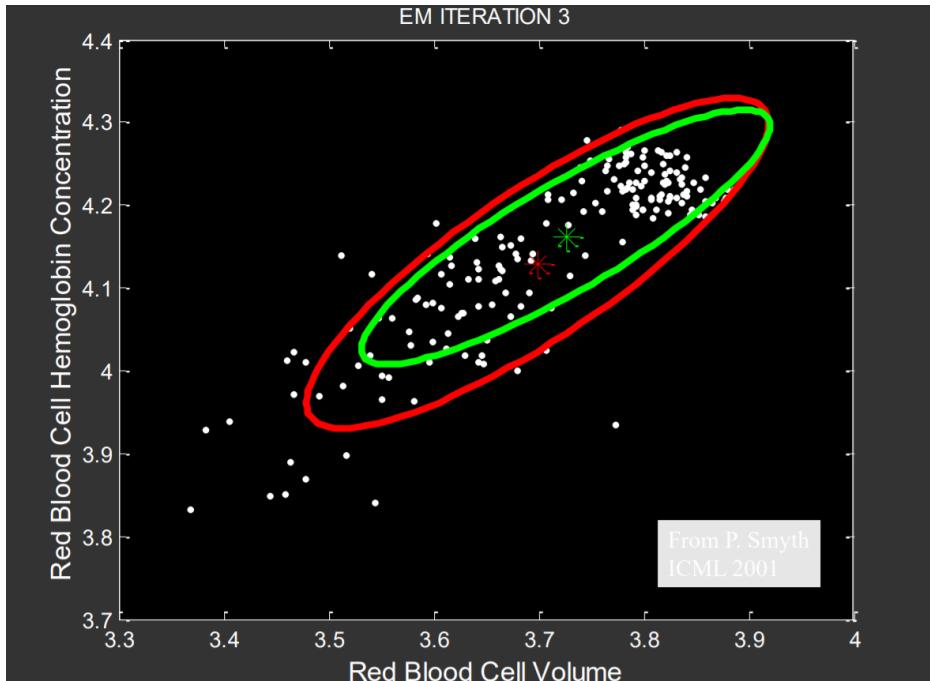
и одново ги земам, значи од ново стануваат сите зелени и се некластериирани и сега одново пресметувам веројатност за секој од податоците да припаѓа на црвениот или на плавиот кластер, имајќи ги во предвид овие распределби пресметуваме веројатност за секој од можните податоци заборавќи како изгледаше претходнот кластерирање, пресметувам како и гледа да припаѓаат податоците на едниот или на другиот кластер f) е после 15 итерации. Овој алгоритам конвергира кога нема повеќе промени.



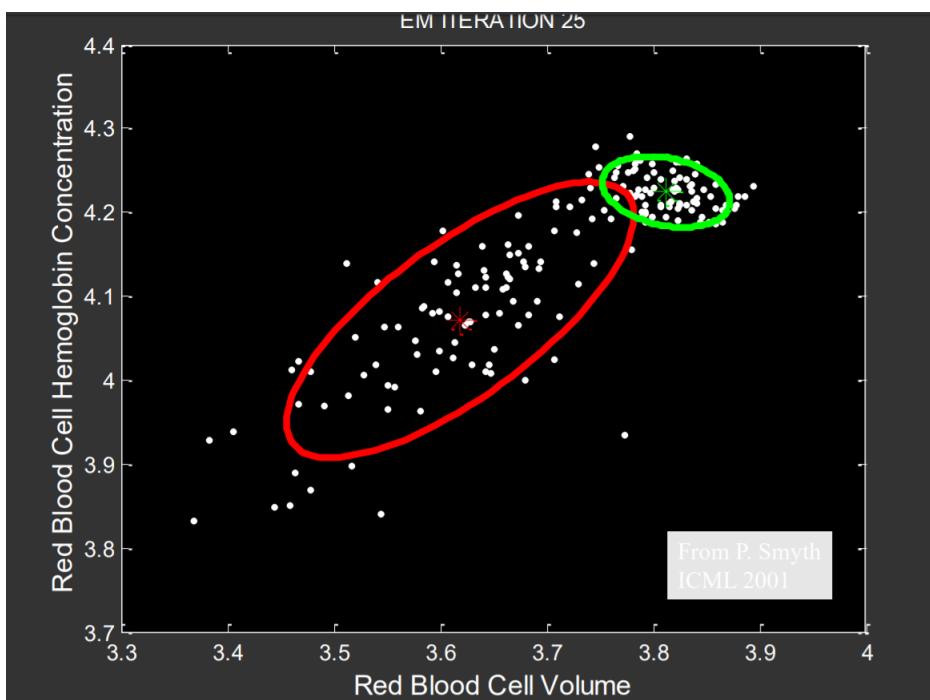
Овде имаме податоци за пациенти со анемија. И сега ако се вакви податоците значи имам вакви податоци значи барам некако овие податоци да се кластерираат односно да се групираат, пациентите врз основа на овие два податоци/карактеристики.



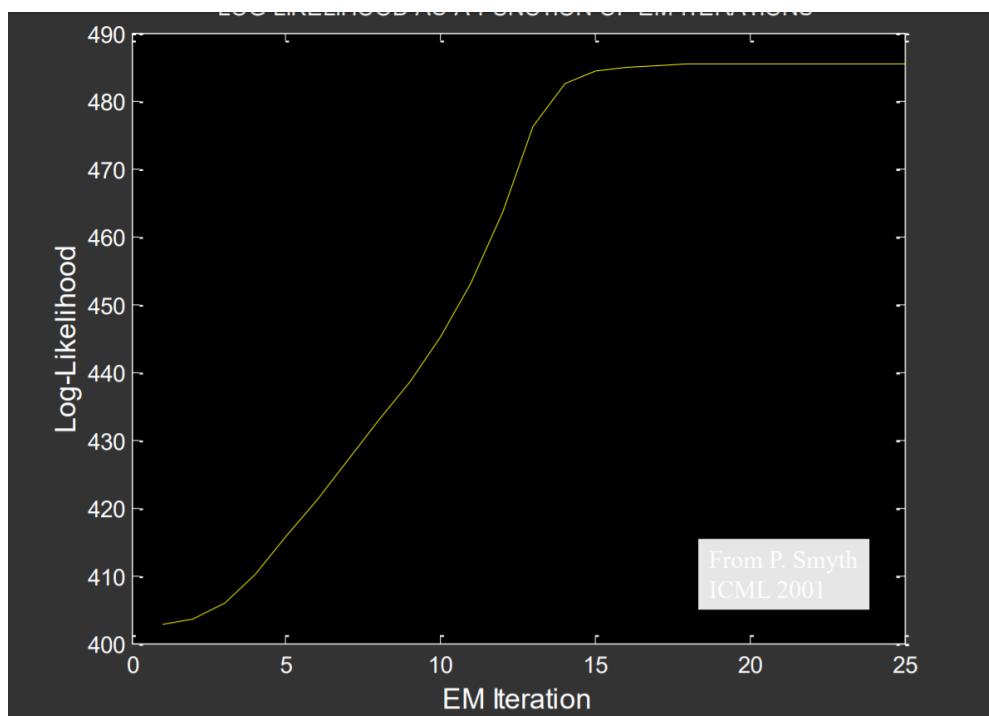
Во првата итерација пресметуваме средни вредности и гледаме дека се близки еден до друг и одредени коваријансни матрици, првично ги земам овие две претпоставки и ги кластерираме податоците во соодветните кластери.



Напомена во овој пример нагласено е како се менуваат параметрите на двета кластери со итерациите, ова сега е 3-та итерација и забележуваме како кластерите се одалечуваат односно забележуваме некакво почетно кластерирање.



Во 25-тата итерација сме конвергирале и сме ги добиле оптималните вредности за соодветната средна вреснот и коваријансната матрица



Ова е
репрезентација
на log likelihood
во однос на
итерациите,
гледаме дека
после 15
итерација
немаме многу
згоменост, и се
намалува
примената
накај 20-тата
итерација значи
алгоритамот
конвергира.

The K means algorithm

Многу често користен алгоритам за кластерирање е овој, овие претходно што ги зборувавме се soft алгоритми, е сега ако имаме **hard assignment**, односно да земиме како што и претходно кажавме да ги бараме класетрот К кој што ќе ја максимизира веројатноста податокот во тој кластер тогаш добиваме К минус алгоритамот, тука го земаме само едниот кластер кој што има максимална веројатност. И велиме податокот припаѓа ТОЧНО на тој еден кластер, додека претходно викавме дека припаѓа на сите кластери со одредена веројатност. И сега овде во овој случај ги менуваме само μ и вредностите односно само средната вредност и се претпоставува дека кластерите имаат иста фиксна коваријансна матрица, овде го апдејтираме само μ , и всушност сега тоа се сведува на доделување на секој податок на соодветниот кластер до чија што средна вредност е најблизок, на тој кластер ќе припаѓа податокот. Значи како што кажавме бидејќи претпоставуваме еднинечна коваријансна матрица за секој кластер тогаш најголема веројатност за кластерот x_i да припаѓа на кластер, се пресметува така што го бараме минималното растојание помеѓу точката x_i до сите средни вредности К, и ова се сведува во суштина на евклидово растојанија, ова треба на сите податоци бараме евклидово растојание до центрите на сите кластери, ако се два кластери на пример, за сите податоци пресметувам растојанието од едниот до другата средна вредност, до таа што е поблиску, тогаш на таа на крајот ќе припаѓа соодветниот податок, и тоа во суштина е чекорот Е, додека М чекорот се состои од аудејтирање сега на средните вредности, сега пак за центар земам средна вредност, од сите податоци кои што припаѓаат на одреден кластер, тој е новиот од кластерот, средна вредност од сите податоци кои што припаѓаат на соодветниот кластер.

Consider the following delta-function approximation to the posterior computed during the **E step**: $p(z_i = k | \mathbf{x}_i, \theta) \approx \mathbb{I}(k = z_i^*)$

where $z_i^* = \operatorname{argmax}_k p(z_i = k | \mathbf{x}_i, \theta)$

Given the hard cluster assignments, the **M step** updates each cluster center by computing the mean of all points assigned to it:

$$\mu_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$$

K-means vs EM for GMM

The K-means algorithm is **faster** than EM for GMMs with full covariance matrices,

But its **results** tend to be **not as good**, since

- it cannot model the **shape of the clusters** (hence it does not learn a distance metric).
- it is not robust to **ambiguous points** that are equidistant between clusters, because it uses hard assignment. (Probabalistic inference in the E step would softly assign such ambiguous points to multiple clusters.)

The K means algorithm

Algorithm 11.1: K-means algorithm

- 1 initialize \mathbf{m}_k ;
- 2 repeat
 - 3 Assign each data point to its closest cluster center;
 - 4 Update each cluster center by computing the mean:

$$\mu_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i;$$
- 5 until converged;

```
function mu = kmeans(data, K, maxIter, thresh)
[N D] = size(data);
% initialization by picking random pixels
% mu(k,:) = k'th center
perm = randperm(N);
mu = data(perm(1:K), :);

converged = 0;
iter = 1;
while ~converged & (iter < maxIter)
    newmu = zeros(K,D);
    % dist(i,k) = squared distance from pixel i to center k
    dist = sqdist(data', mu');
    [junk, assign] = min(dist,[],2);
    for k=1:K
        newmu(k,:) = mean(data(assign==k,:));
    end
    delta = abs(newmu(:) - mu(:));
    if max(delta./abs(mu(:))) < thresh
        converged = 1;
    end
    mu = newmu;
    iter = iter + 1
end

if ~converged
    error(sprintf('did not converge within %d iterations', maxIter))
end
```

Значи иницијализираме \mathbf{m}_k – тоа се средините т.е средните вредности, во овој случај за к-та кластери. Потоа го доделуваме секој податок од податочното множество на најблискиот кластер, со тоа што пресметувам растојание за секој податок и го земаме минималнот, и така одредувакој ќе припаѓа кој на кој класа. Врз основа на тие параметри пресметуваме нов μ т.е ново \mathbf{m} , односно средината ќе биде средина од сите податоци кои што припаѓаат на кластерот. После ги земам сите пак да се зелени, односно ако сме имале две класи плава и црвена кога ќе почнеме со новата итерација ги маркираме еднобојни пример зелени и правиме нови калстери, се додека не конвергира алгоритмот, т.е кога не се менуваат параметрите.

Vector quantization

Сега имајќи ги во предвид овие центри, па ако имаме 8 кластери ќе имаме 8 центри, тогаш може да кажеме дека сите податоци кои што припаѓаат на тој кластер да се заменат со средните вредности од тој кластер, ние центрите ги нарекуваме **codebook** или на некој начин ги користиме за кодирање и може да се користат за дискретизација на податоците, значи да ги дискретизирам, на пример ако имаме 8 кластери да ги ставам да припаѓаат секој од податоците во еден од 8те кластери, и ова се нарекува vector quantization. Поентата е дека секој нов податок се заменува со најблискиот ценар, односно со центарот во кластерот во кој што тој припаѓа, и тогаш имаме множество од т дискретни симболи, и ова може да се користи за компресија на податоци и еве да видиме пример:

Vector quantization

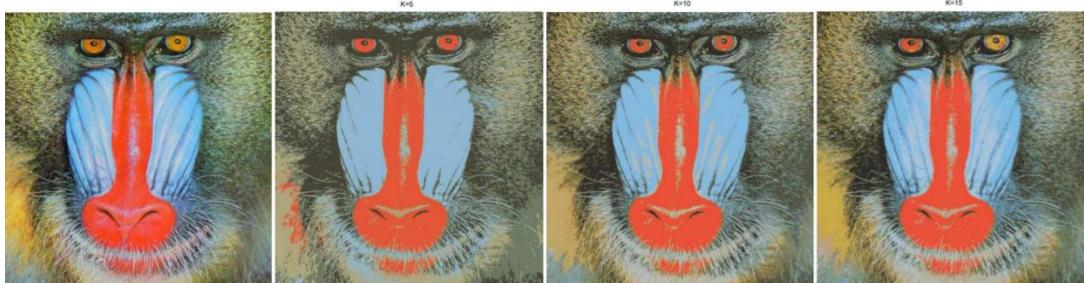
- For example, consider $N = 512 \times 512 = 262,144$ pixel image, and each pixel is represented by three 8-bit numbers (each ranging from 0 to 255) that represent the green, red and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $24N = 6,291,456$ bits. We applied vector quantization to this using $K = 5, 10, 15$ codewords using the code below.

```
% vqDemo
% Learn code book on small image (for speed)
A = double(imread('mandrill-small.tiff'));
figure; imshow(uint8(round(A)));
[nrows ncols ncolors] = size(A);
% data(i,:) = rgb value for pixel i
data = reshape(A, [nrows*ncols ncolors]);
maxIter = 100; % usually converges long before 100 iterations!
thresh = 1e-2;
K = 15;
mu = kmeansKPM(data, K, maxIter, thresh);

% Apply codebook to quantize large image
B = double(imread('mandrill-large.tiff'));
imshow(uint8(round(B)));
[nrows ncols ncolors] = size(B);
data = reshape(B, [nrows*ncols ncolors]);
[N D] = size(data);
dist = sqdist(data', mu');
[junk, assign] = min(dist,[],2);
qdata = zeros(size(data));
for k=1:K
    ndx = find(assign==k);
    Nassign = length(ndx);
    qdata(ndx, :) = repmat(mu(k,:), Nassign, 1);
end
Qimg = reshape(qdata, [nrows ncols ncolors]);
figure; imshow(uint8(round(Qimg)));
title(sprintf('K=%d',K))
```

Е сега ако сакаме да ја намалиме оваа слика тогаш може да направиме да примениме vector quantization, така што ќе користиме $K=5, 10, 15$. Пример ако земам 5, ќе имаме 5 кластери секој кластер соодветна боја па сликата ќе се претстави во 5 бои во зависност од тоа бојата до кој кластер што претставува боја е најблизок.

Vector quantization

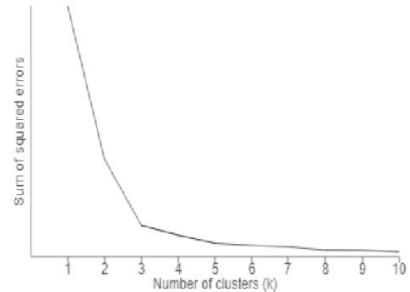


K-medoids

Постојат ситуации каде што правење на средна вредност нема логика, на пример ако разгледуваме кластерирање на графови, тогаш средна вредност ќе биде некој јазел што не постои, нема логика во такви случаи, тогаш ова евклидово растојание може да се замени со било која мерка за сличност на растојание, тогаш го бараме К кој што го минимизира растојанието помеѓу X_n и K, е сега во зависност ако станува збор за анализа на графови може да станува збор за графови кои што треба да се скокнат до центарот итн. Значи евклидово растојание нема смисла и треба да се замени со друго, е тука M чекорот е кога бараме значи новите центри, кога брам нов центар кај графови може да најма центар кој што не постои, т.е се бара еден јазел кој е најблиску до кластерот. И тој се нарекува K-medoids тоа се нарекува затоа што и центарот се смета како податок. Овде во секој чекор центрите се во суштина дел од податочното множество. Е сега како што кажавме проблем е пресметување на K, бројот на мешани компоненти, тоа зависи од апликацијата, може да се исцртаат податоците, е сега овде проблем е всушност затоа што за тренинг множеството колку повеќе го зголемувам K тоа повеќе се фитуваат податоците, меѓуатоа доаѓам до проблемот на overfitting, најдноставно е да се направи **cross validation** со посебно валидациско множество. Исто така друг алгоритам е да се додаваат K, се додека не добијеме "elbow" или намалување на подобрувањето на грешката, после ова додавањето на кластери помалце придонесува за подобрување на резултатите, другиот начин е рачно да се проверат начините на кластерите.

Estimating the number of mixture components

- Defined by the application
- Plot data (after PCA) and check for clusters
- Maximum likelihood will always favor the largest possible value of K, since that gives the greatest number of parameters and maximizes the ability to fit the data. But this may result in overfitting. A simple alternative is to use **cross validation**, i.e., to find the value of K that maximizes the log likelihood of a **validation set**.
- Incremental (leader-cluster) algorithm: Add one at a time until "**elbow**"
- Manual check for meaning

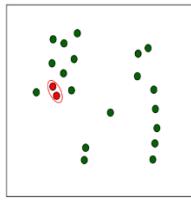


Hierarchical clustering

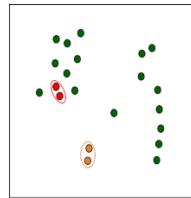
- GMMs can be used to perform clustering, but the clustering is “flat”.
- Often, we want to learn hierarchical clusters.
- The most popular approach is to use **agglomerative clustering**, which we will describe below.
- Agglomerative clustering starts with N groups, **each initially containing one training instance**, merging similar groups to form larger groups, until there is a single group.
- At each iteration of an agglomerative algorithm, we choose the two closest groups to merge.

Претходно беше модел веројатносен, ове сега зборуваме за алгоритам кој што всушност користи хиерархиско класерирање.

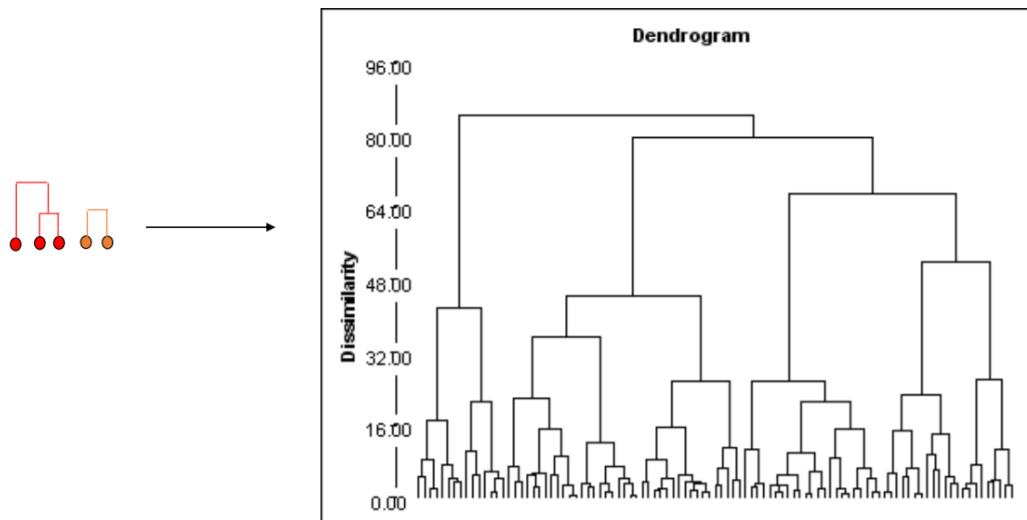
Iteration 1



Iteration 2



Dendrogram



Hierarchical clustering

- Distance between two groups

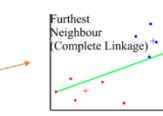
• **single link clustering** - the distance between two groups is defined as the distance between the two closest members of each group:

$$D_{SL}(G_i, G_j) = \min_{x^r \in G_i, x^s \in G_j} D(x^r, x^s)$$



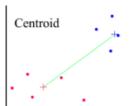
• **complete link clustering** - the distance between two groups is defined as the distance between the two most distant pairs:

$$D_{CL}(G_i, G_j) = \max_{x^r \in G_i, x^s \in G_j} D(x^r, x^s)$$

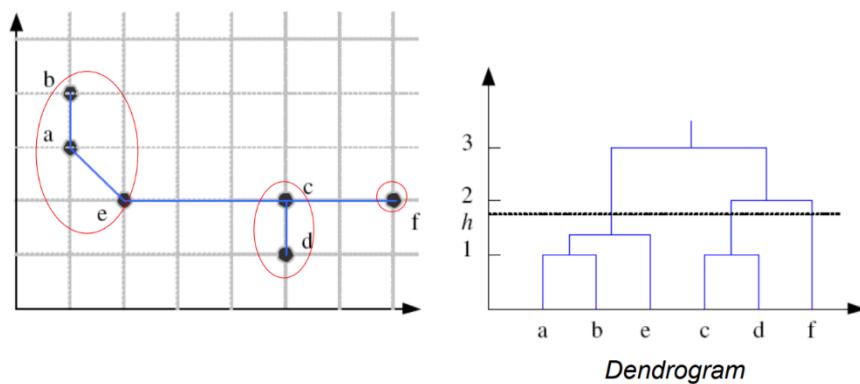


• **average link clustering**, which measures the average distance between all pairs

• **centroid clustering**



Example: Single-Link Clustering



Hierarchical clustering

- Distance measure between instances \mathbf{x}^r and \mathbf{x}^s

Minkowski (L_p) (Euclidean for $p = 2$)

$$d_m(\mathbf{x}^r, \mathbf{x}^s) = \left[\sum_{j=1}^d (\mathbf{x}_j^r - \mathbf{x}_j^s)^p \right]^{1/p}$$

City-block distance

$$d_{cb}(\mathbf{x}^r, \mathbf{x}^s) = \sum_{j=1}^d |\mathbf{x}_j^r - \mathbf{x}_j^s|$$

LECTURE 9 Latent linear models

Претходната лекција гледавме мешавина од гаусеви модели и гледавме во суштина, скриени променливи кои што во суштина беа дискретни, значи за тие кластери во суштина викаме дека податоците може да припаѓаат на некој од тие кластери, меѓутоа има модели каде што овие скриени променливи може да бидат континуирани, сега ќе видиме такви модели и вушност не интересира дали може ако имаме податоци кои што се во повеќе димензии на некој начин да ги компресираме во помалку димензии односно ако помеѓу себе ако тие податоци се корелирани на некој начин да ги сведиме на податоци кои ќе бидат со помала димензија и некорелирани помеѓу себе, значи на некој начин да ги издвоиме тие повеќе димензии и ќе ги компреисраме во помалку димензии што ќе ни кажат нешто, и ќе ни откријат нешто тие помалце димензии за податоците. Значи најдноставни модели со скриени континуирани променливи се ако претпоставиме дека имаме гаусова распределба и на скриените и на надгледуваните променливи, значи тоа е најдноставната верзија, и вушност од тоа произлегуваат и техниките кои ќе ги разгледаме.

Factor Analysis

Значи ако имаме вектор од скриени променливи \mathbf{z}_i така што приорот на тие променливи да биде гаусова распределба, тоа е најдноставната распределба.

$$p(\mathbf{z}_i) = \mathcal{N}(\mathbf{z}_i | \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

Ако надгледуваните променливи се исто така континуирани, па $x_i \in \mathbb{R}^D$, тогаш може да претпоставиме да користиме и гаусова распределба и за лајклихуд, и најдноставниот начин е да претпоставиме дека средината е линеарна функција на скриените влезови, ова е слично како кај линеарна регресија.

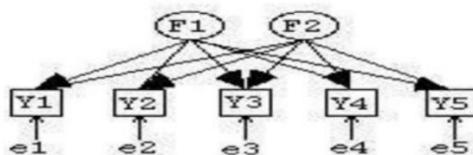
$$p(\mathbf{x}_i | \mathbf{z}_i, \theta) = \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

- where
 - \mathbf{W} is a $D \times L$ matrix – **factor loading matrix**
 - $\boldsymbol{\Psi}$ is a $D \times D$ covariance matrix (diagonal – to force z_i explain the correlation)

\mathbf{W} - е она што нас генерално не интересира, тоа е матрица со димензии $D \times L$, каде што D е вушност димензијата на надгледуваните променливи а L е димензијата на скриените променливи, значи на некој начин ова \mathbf{W} ни кажува како ќе се мапира од видливиот простор т.е од просторот на надгледуваните променливи во просторот на скриените проенливи, и кси ни е коваријансна матрица, која ќе ни открие која е корелацијата на променливите. Ова $\Psi = \sigma^2 I$, вушност тутка имаме изотропен модел значи не само што кси ќе биде дијагонално туку ќе има исти вредности со по дијагоналата, тоа е специјален случај кој се нарекува probabilistic principal components analysis or PPCA. Кога ова σ^2 тежнее кон нула, тогаш го добиваме обилниот PCA. Вушност претпоставуваме дека постои простор за компресија ако информациите во \mathbf{x} не се конкретно независни, па таа независност сакаме да ја пресликаме со тоа што ќе користиме помалце димензии. Значи \mathbf{x} е во суштина линеарна трансформација на скриените променливи кои што се со некоја помала димензија. Уште

еднаш Factor analysis е класа од процедури кои се користат за редукција и сумаризација на податоци. Е сега зошто го користиме, зошто вакво нешто би нас требало? Пример имаме некои податоци кои што се со димензија 15, може и да се 1000, и може на некој начин да пробам тие да ги компресирам и една од предностите е ако ги компреисрам или ги средам на една или две димензии тоа може и да го визуелизирме, значи таа визуелизација може и да ни открие нешто за нашите податоци, нешто што ние од прв поглед може нема да го приметиме ако го визуелизирме во простор со помалце димензии може да откриеме нешто за податоците, таа графичка репрезентација може да ни открие некоја зависност која стои зад тие податоци. Исто така може ако добиеме помало множество со променливи со помала димензија тие да ги користиме наместо оригиналното множество на корелирани податоци со што го поедноставуваме моделот кој го користиме за податоците. Еве како изгледа едноставен дијаграм:

F1 and F2 are two common factors, Y1-Y5 are observed variables, e1-e5 represent residuals or unique factors, which are assumed to be uncorrelated with each other



Е сега математиката тука треба да е позната, тука имаме линеарен гаусов систем

- FA can be thought of as a way of specifying a joint density model on x using a small number of parameters.
 - To express MVN, originally, $O(D * D)$ parameter is needed to express covariance. Assuming latent variable, only $O(D * L)$ parameter is needed.

Linear Gaussian system

$$\begin{aligned}
 p(x) &= \mathcal{N}(x|\mu_x, \Sigma_x) \\
 p(y|x) &= \mathcal{N}(y|Ax + b, \Sigma_y) \\
 p(y) &= \mathcal{N}(y|A\mu_x + b, \Sigma_y + A\Sigma_x A^T)
 \end{aligned}$$

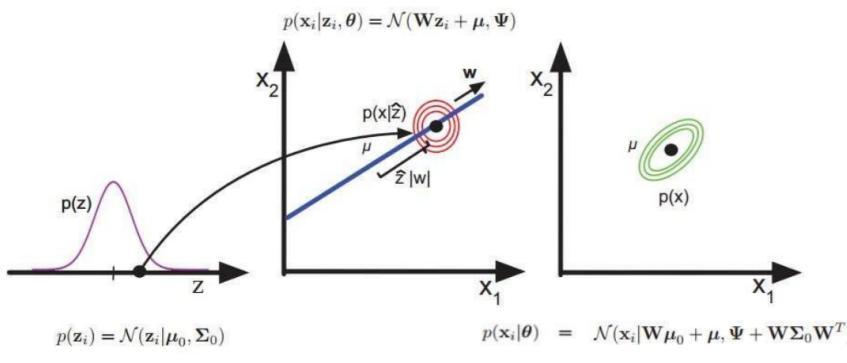
(4.126)

$$\begin{aligned}
 p(x_i|\theta) &= \int \mathcal{N}(x_i|Wz_i + \mu, \Psi) \mathcal{N}(z_i|\mu_0, \Sigma_0) dz_i \\
 &= \mathcal{N}(x_i|W\mu_0 + \mu, \Psi + W\Sigma_0 W^T)
 \end{aligned}$$

$$p(x_i|\theta) = \mathcal{N}(x_i|W\mu_0 + \mu, \Psi + W\Sigma_0 W^T)$$

Factor Analysis, може да се смета како параметризација т.е пресметување на помалце параметри во однос на првобиниот модел. Значи равенките горе е линеарен гаусов систем, за разлика од гаус линеарен систем во овој случај x е Z т.е скриените променливи, y е во суштина x . И сега врз основа на овој систем имаме $p(x_i|\theta)$. Следува пример

- The generative process, where $L=1$, $D=2$ and Ψ is diagonal, is illustrated in the figure
- We take an isotropic Gaussian “spray can” and slide it along the 1d line defined by $wz_i + \mu$.
- This induces an elongated (and hence correlated) Gaussian in 2d



A sampled value of the observed variable is obtained by first choosing a value for the latent variable and then sampling the observed variable conditioned on this latent value. Specifically, the 2-dimensional observed variable x is defined by a linear transformation of the 1-dimensional latent variable z plus, additive Gaussian ‘noise’

This framework is based on a mapping from latent space to data space

Имаме скриена променлива со една димензија која има нормална распределба, и сега тута, земајќето примерот на надгледуваната променлива тоа е x се добива со оа што прво се пресметува некоја вредност од скриената промелва па се прави условна веројатност врз основа на таа z , всушност овој дводимензионален простор X_1, X_2 простор е дефиниран со линеарна трансформација со скриениот простор и тута плус се додава одреден шум кој е дефиниран со кси, гаусовата распределба на X_1, X_2 е развлечена која кажува дека постои корелација помеѓу X_1 и X_2 . Е сега нас не интересираат Z променливите, т.е тие скриените променливи, кои што треба да ни откријат нешто за самите податоци, нас не интересираат во суштина Z , исто користеќи го овој линеарен гаусов систем и со користење на баесовото правило може да добиеме веројатноста за z ако x ,

$$\begin{aligned} p(\mathbf{x}) &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_x, \Sigma_x) \\ p(\mathbf{y} | \mathbf{x}) &= \mathcal{N}(\mathbf{y} | \mathbf{A}\mathbf{x} + \mathbf{b}, \Sigma_y) \end{aligned} \quad (4.124)$$

$$\begin{aligned} p(\mathbf{x} | \mathbf{y}) &= \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_{x|y}, \Sigma_{x|y}) \\ \Sigma_{x|y}^{-1} &= \Sigma_x^{-1} + \mathbf{A}^T \Sigma_y^{-1} \mathbf{A} \\ \boldsymbol{\mu}_{x|y} &= \Sigma_{x|y} [\mathbf{A}^T \Sigma_y^{-1} (\mathbf{y} - \mathbf{b}) + \Sigma_x^{-1} \boldsymbol{\mu}_x] \end{aligned} \quad (4.125)$$

Linear Gaussian system,
Bayes rule

$$\begin{aligned} p(\mathbf{z}_i | \mathbf{x}_i, \theta) &= \mathcal{N}(\mathbf{z}_i | \mathbf{m}_i, \Sigma_i) \\ \Sigma_i &\triangleq (\Sigma_0^{-1} + \mathbf{W}^T \Psi^{-1} \mathbf{W})^{-1} \\ \mathbf{m}_i &\triangleq \Sigma_i (\mathbf{W}^T \Psi^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) + \Sigma_0^{-1} \boldsymbol{\mu}_0) \end{aligned}$$

\mathbf{m}_i are sometimes called the latent **scores**, or latent **factors**.

Тута нас не интересираат m_i кои се наречени скрени фактори. Всушност дименизацијата на ова m_i таа е $L \times 1$, L – е димензијата на скриентире променливи односно го редуцираме од d димензии во 1 димензии. Следува пример

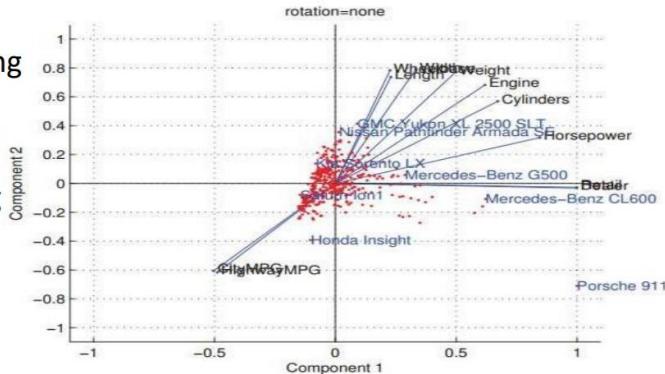
- $D = 11$ dimensions (engine size, number of cylinders, price, ...), $N = 328$ examples (types of cars), $L = 2$
- Project unit vectors for each feature (engine size, number of cylinders, .. 11) into the low dimensional space - blue lines

- Also, each data point is projected using

$$L^* L^* (L^* D^* D^* D^* 1 + L^* L^* L^* 1) = L^* 1$$

$$m_i \triangleq \Sigma_i (W^T \Psi^{-1} (x_i - \mu) + \Sigma_0^{-1} \mu_0)$$

- Dimensional reduction of the training set (red dot) from D dimensions to L dimensions



Значи имаме податочно множество за автомобили, и за тие автомобили имаме 11 карактеристики значи 11 димензии, големина на мотор, цилиндри итн итн, со 328 примероци е сега оца од 11 димензии сакаме да го сведиме на 2 димензии, ако ги кориситме претходните равенки каде што во суштина ја имаме димензијата на таа равенка, ова се кога ќе се пресмета се добива $L^* 1$ во суштина секоја податок се мапира во две димензии тука на цртежот и со црвените точки, запувани од 11 димензии во 2 димензии. И плус проектириани се единечни мерки на самите карактеристики, овие две димензии што ги добиваме не често може да кажеме што значат, е сега во овој проимер тука може да видиме, дека некој автомобили се и напишани, на нашиот цртеж може да видиме дека долж компонентата 2 вушност е променливата кој а што означува кој е продавач и имаме две такви променливи и на некој начин таа се поврзува со цената, така тие со повисока цена се повеќе на десната страна. Од друга страна на компонентата 1 генерално ги имаме променливите за тежина, литри на километар итн, ова на некој начин оваа компонента еден би одговарала на ефикасносната што е потешка ќе има помала ефикасност, понеefикасна се наѓа погоре што поefикасна подолу, овде едната компонента би била цената по што се разликуваат сите автомобили едната е цената а другата ефикасноста. На некој начин може да ги споиме овие 11 димензии во 2. Исто како Гаусови мешани модели може да имаме мешавина од Factor analysis.

Mixture of factor analysis

Model - each data x_i comes from k FAs (similar to GMM)

Let the k'th linear subspace of dimensionality L_k be represented by W_k , for $k = 1: K$.

Suppose we have a latent indicator $q_i \in \{1, \dots, K\}$ specifying which subspace we should use to generate the data.

We then sample z_i from a Gaussian prior and pass it through the W_k matrix (where $k = q_i$) and add noise.

$$\begin{aligned} p(x_i|z_i, q_i = k, \theta) &= \mathcal{N}(x_i|\mu_k + W_k z_i, \Psi) \\ p(z_i|\theta) &= \mathcal{N}(z_i|0, I) \\ p(q_i|\theta) &= \text{Cat}(q_i|\pi) \end{aligned}$$

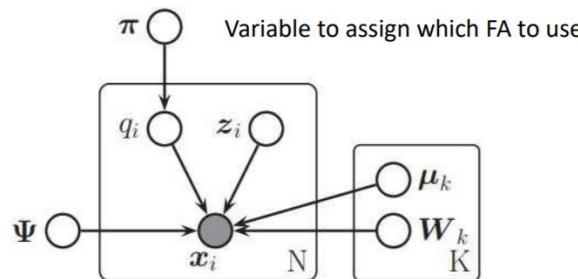
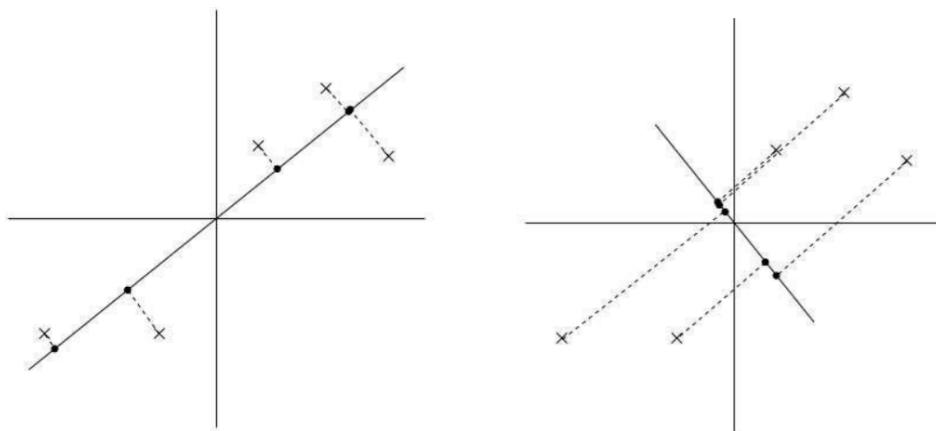


Figure 12.3 Mixture of factor analysers as a DGM.

Сега имаме повеќе имаме повеќе простори и треба да одредиме кој подпростор треба да го користиме за да ги генерираме податоците, така што сега ако имаме к-тиот линеарен подпростор со диманзиија L_k и е претставен со W_k сега сите се претставени со к знак што означува к-тиот подпростор. Значи потоа се семплира во суштина z_i , и тоа се поминува на W_k матрицата и од тоа се одредува x_i , к се одредува врз q_i кој ни одредува кој подпростор треба да го користиме, сега имаме, $z_i | q_i$. q_i има категорична дистрибуција односно може да припаѓа на една од к-те различни подпростори. Она што најчесто тута се користи е PCA, тоа е кога овој шумот ние го изоставуваме, значи коваријансата кси=сигма на квадрат по и, кога сигма тежи кон 0.



Тука имаме 5 податоци пример тука дводемензионални сакаме да ги редуцираме во една димензија, тогаш најдобро е да ги проектираме така што ќе се максимизира коваријансата на проектираниите податоци, значи податоците најмногу што може да се развлечени на таа една димензија. Интуитивно подобро е да се земе примерок каде имаме поголема варијанса

отколку на вториот каде многу помалце се прикажуваат разликите во податоците, ова е од интуитивна гледна точка, тоа е PCA.

Principal component analysis

Consider the FA model where we constrain $\psi = \sigma^2 I$, and W to be orthonormal.

It can be shown (Tipping and Bishop 1999) that, as $\sigma^2 \rightarrow 0$, this model reduces to classical (nonprobabilistic) principal components analysis (PCA)

$$p(\mathbf{x}_i | \mathbf{z}_i, \theta) = \mathcal{N}(\mathbf{W}\mathbf{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})$$

The version where $\sigma^2 > 0$ is known as probabilistic PCA(PPCA)

PCA-theorem

Претпоставуваме дека сакаме да најдеме множество односно матрица W која што е составена од вектори и соодветното scores, т.е проекциите на податоците во новата проекција, така што сакаме да ја минимизираме просечната грешка на реконструкција, значи имаме оригиналните податоци X и нивната реконструкција, ги ставаме во новата реконструкција ги ставаме назад и ги гледаме која е грешката. Имаме

Theorem 12.2.1. Suppose we want to find an orthogonal set of L linear basis vectors $\mathbf{w}_j \in \mathbb{R}^D$, and the corresponding scores $\mathbf{z}_i \in \mathbb{R}^L$, such that we minimize the average **reconstruction error**

$$J(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \quad (12.26)$$

where $\hat{\mathbf{x}}_i = \mathbf{W}\mathbf{z}_i$, subject to the constraint that \mathbf{W} is orthonormal. Equivalently, we can write this objective as follows:

$$J(\mathbf{W}, \mathbf{Z}) = \|\mathbf{X} - \mathbf{W}\mathbf{Z}^T\|_F^2 \quad (12.27)$$

where \mathbf{Z} is an $N \times L$ matrix with the \mathbf{z}_i in its rows, and $\|\mathbf{A}\|_F$ is the **Frobenius norm** of matrix \mathbf{A} , defined by

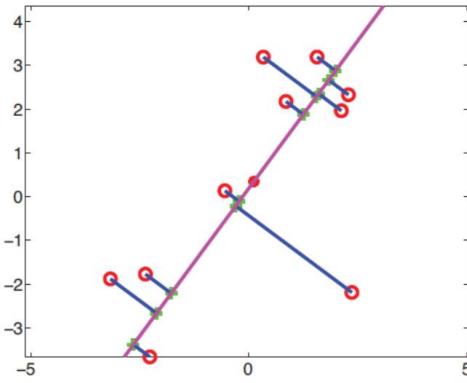
$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2} = \sqrt{\text{tr}(\mathbf{A}^T \mathbf{A})} = \|\mathbf{A}(:, :)\|_2 \quad (12.28)$$

The optimal solution is obtained by setting $\hat{\mathbf{W}} = \mathbf{V}_L$, where \mathbf{V}_L contains the L eigenvectors with largest eigenvalues of the empirical covariance matrix, $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T$. (We assume the \mathbf{x}_i have zero mean, for notational simplicity.) Furthermore, the optimal low-dimensional encoding of the data is given by $\hat{\mathbf{z}}_i = \mathbf{W}^T \mathbf{x}_i$, which is an orthogonal projection of the data onto the column space spanned by the eigenvectors.

Важно е дека оптимално решение се добива кога $W=Vl$ – во суштина ова е составено од вектори на емпириска коваријансна матрица, ги бараме нејзините сопствени вектори и тие ќе ни го претставуваат новиот координатен систем и од него може да земем L колку што сакаме, чија што сопствена вредност е најголема, сега одиме на доказо ->

proof sketch

- Finding W that reduces reconstruction error = Finding W that maximizes the variance of data z projected by W
- Find the W that maximizes the variance of the data projected on W by lagrange multiplier optimization.
- The W of which the variance of the data projected on W is maximized, contains the eigenvectors with largest eigenvalues of the empirical covariance matrix



Ќе покажеме дека W кој што ја намалува грешката на конструкција всушност е еднакво на наоѓање на w што ја максимизира варијансата, т.е она што го заклучивме дека податоците е подобро да се пораширени во новите димензии. Потоа

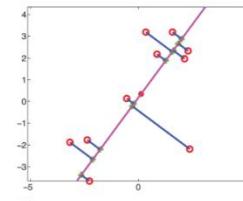
$$\begin{aligned} w_j &\in \mathbb{R}^D \text{ to denote the } j\text{'th principal direction} \\ x_i &\in \mathbb{R}^D \text{ to denote the } i\text{'th high-dimensional observation,} \\ z_i &\in \mathbb{R}^L \text{ to denote the } i\text{'th low-dimensional representation} \\ \tilde{z}_j &\in \mathbb{R}^N \text{ to denote the } [z_{1j}, \dots, z_{Nj}], \text{ which is the } j\text{'th component of all the low-dimensional vectors} \end{aligned}$$

- Start by estimating the best 1d solution, i.e. to find the w_1 , and the corresponding projected points \tilde{z}_1
- The reconstruction error is given by:

$$\begin{aligned} J(w_1, z_1) &= \frac{1}{N} \sum_{i=1}^N \|x_i - z_{i1}w_1\|^2 = \frac{1}{N} \sum_{i=1}^N (x_i - z_{i1}w_1)^T (x_i - z_{i1}w_1) \\ &= \frac{1}{N} \sum_{i=1}^N [x_i^T x_i - 2z_{i1}w_1^T x_i + z_{i1}^2 w_1^T w_1] \quad \text{=1 (by the orthonormality assumption)} \\ &= \frac{1}{N} \sum_{i=1}^N [x_i^T x_i - 2z_{i1}w_1^T x_i + z_{i1}^2] \end{aligned}$$

Proof of PCA

$$J(\mathbf{w}_1, \mathbf{z}_1) = \frac{1}{N} \sum_{i=1}^N [\mathbf{x}_i^T \mathbf{x}_i - 2z_{i1} \mathbf{w}_1^T \mathbf{x}_i + z_{i1}^2]$$



- Taking derivatives wrt z_{i1} and equating to zero gives

$$\frac{\partial}{\partial z_{i1}} J(\mathbf{w}_1, \mathbf{z}_1) = \frac{1}{N} [-2\mathbf{w}_1^T \mathbf{x}_i + 2z_{i1}] = 0 \Rightarrow z_{i1} = \mathbf{w}_1^T \mathbf{x}_i$$

- So the optimal reconstruction weights are obtained by orthogonally projecting the data onto the first principal direction \mathbf{w}_1

- Plugging back it gives:

$$J(\mathbf{w}_1) = \frac{1}{N} \sum_{i=1}^N [\mathbf{x}_i^T \mathbf{x}_i - z_{i1}^2] = \text{const} - \frac{1}{N} \sum_{i=1}^N z_{i1}^2$$

- The variance of the projected coordinates is given by:

$$\text{var}[\tilde{\mathbf{z}}_1] = \mathbb{E}[\tilde{\mathbf{z}}_1^2] - (\mathbb{E}[\tilde{\mathbf{z}}_1])^2 = \frac{1}{N} \sum_{i=1}^N z_{i1}^2 - 0$$

- Because the data has been centered: $\mathbb{E}[z_{i1}] = \mathbb{E}[\mathbf{x}_i^T \mathbf{w}_1] = \mathbb{E}[\mathbf{x}_i]^T \mathbf{w}_1 = 0$

Proof of PCA

- We see that, *minimizing the reconstruction error* is equivalent to *maximizing the variance* of the projected data (the variance is with a sign “-” in the reconstruction error):

$$\arg \min_{\mathbf{w}_1} J(\mathbf{w}_1) = \arg \max_{\mathbf{w}_1} \text{var}[\tilde{\mathbf{z}}_1]$$

- It is often said that PCA finds the direction of maximal variance

- The variance of the projected data can be written as:

$$\frac{1}{N} \sum_{i=1}^N z_{i1}^2 = \frac{1}{N} \sum_{i=1}^N \mathbf{w}_1^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{w}_1 = \mathbf{w}_1^T \hat{\Sigma} \mathbf{w}_1$$

- Where $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^N \sum_i \mathbf{x}_i \mathbf{x}_i^T$ is the empirical covariance matrix

- We can trivially maximize the variance of the projection (and hence minimize the reconstruction error) by letting $\|\mathbf{w}_1\| \rightarrow \infty$, so we impose the constraint $\|\mathbf{w}_1\| = 1$ and instead maximize:

$$\tilde{J}(\mathbf{w}_1) = \mathbf{w}_1^T \hat{\Sigma} \mathbf{w}_1 + \lambda_1 (\mathbf{w}_1^T \mathbf{w}_1 - 1)$$

- Where λ_1 is Lagrange multiplier. Taking derivatives and equating to 0, we have

$$\frac{\partial}{\partial \mathbf{w}_1} \tilde{J}(\mathbf{w}_1) = 2\hat{\Sigma} \mathbf{w}_1 - 2\lambda_1 \mathbf{w}_1 = 0$$

the direction that maximizes the variance is an eigenvector of the covariance matrix

$$\hat{\Sigma} \mathbf{w}_1 = \lambda_1 \mathbf{w}_1$$

- Multiplying by \mathbf{w}_1 (and using $\mathbf{w}_1^T \mathbf{w}_1 = 1$) for λ_1 we obtain $\mathbf{w}_1^T \hat{\Sigma} \mathbf{w}_1 = \lambda_1$ which is equal to the variance. So, if we want to maximize the variance, we pick the eigenvector that corresponds to the largest eigenvalue

- Now let us find another direction \mathbf{w}_2 to further minimize the reconstruction error, subject to $\mathbf{w}_1^T \mathbf{w}_2 = 0$ and $\mathbf{w}_2^T \mathbf{w}_2 = 1$. The error is

$$J(\mathbf{w}_1, \mathbf{z}_1, \mathbf{w}_2, \mathbf{z}_2) = \frac{1}{N} \sum_{i=1}^N \|\mathbf{x}_i - z_{i1}\mathbf{w}_1 - z_{i2}\mathbf{w}_2\|^2$$

- the solution is given by the eigenvector with the second largest eigenvalue:

$$\hat{\Sigma} \mathbf{w}_2 = \lambda_2 \mathbf{w}_2$$

- The proof continues in this way. (Formally one can use induction.)

Минимизирањето на конструкциската грешка е еквивалентно со максимизирање на варијансата на проектирани податоци. Ова е првиот дел од доказот, очигледно не треба да се знај само да се сфати која е поентата на теоремта.

Singular value decomposition (SVD)

SVD вели дека секоја матрица може да се претстави со три матрици.

Singular value decomposition (SVD)

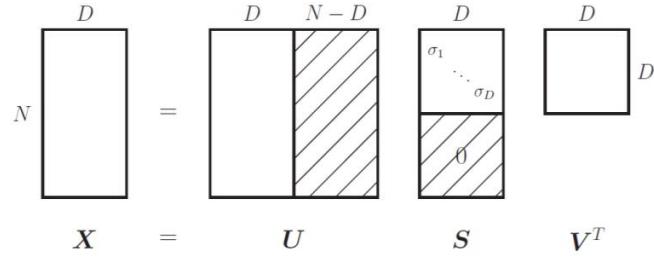
- SVD is closely related to PCA**
 - Running the SVD gives the PCA solution for W (in another way)
- The truncated SVD writes only as many singular values as it wants.**
 - The identity of singular values will be explained later.
- SVD is a matrix decomposition algorithm** that splits a large matrix into three. Any (real) $N \times D$ matrix \mathbf{X} can be decomposed as follows:

$$\underbrace{\mathbf{X}}_{N \times D} = \underbrace{\mathbf{U}}_{N \times N} \underbrace{\mathbf{S}}_{N \times D} \underbrace{\mathbf{V}^T}_{D \times D}$$

U is an $N \times N$ matrix whose columns are orthonormal (so $\mathbf{U}^T \mathbf{U} = \mathbf{I}_N$).
V is $D \times D$ matrix whose rows and columns are orthonormal (so $\mathbf{V}^T \mathbf{V} = \mathbf{V} \mathbf{V}^T = \mathbf{I}_D$).
S is a $N \times D$ matrix containing the $r = \min(N, D)$ singular values $\sigma_i \geq 0$ on the main diagonal,

Од S матрицата освен тие вредности по дијагонала, другите сите вредности се 0. Секоја матрица може да запише во овој формат. Кога множиме $\mathbf{U}^* \mathbf{S}$ тогаш сите тие че с епомножат по нула ќе се нула така да што може да добиеме економична верзија на SVD така што нив мнема да ги земаме во пресметките, тогаш земаме ако $N > D$ тогаш постојат D сингуларни вредности, како во овој случај и може да скратам ова U да биде со димензија $N \times D$, а S да е $D \times D$ и V – $N \times D$.

Singular value decomposition (SVD)



- **Economy sized SVD or thin SVD**

- Assuming ($N>D$), there are at most D singular values
- The last $N-D$ columns of U are irrelevant since they will be multiplied by 0

$$\underbrace{\mathbf{X}}_{N \times D} = \underbrace{\hat{\mathbf{U}}}_{N \times D} \underbrace{\hat{\mathbf{S}}}_{D \times D} \underbrace{\hat{\mathbf{V}}^T}_{D \times D}$$

- Similarly, if ($D>N$)

$$\underbrace{\mathbf{X}}_{N \times D} = \underbrace{\hat{\mathbf{U}}}_{N \times N} \underbrace{\hat{\mathbf{S}}}_{N \times N} \underbrace{\hat{\mathbf{V}}^T}_{N \times D}$$

- Computing the economy-sized SVD takes $O(ND\min(N, D))$

Следува пример:

SVD: example

```
julia> X = randn(5,3)
5x3 Array{Float64,2}:
 0.154315  0.0536085  1.4482
 0.854679  -0.44341   1.15055
-1.68437   -0.538634  -0.707005
 1.36306   -0.328046  -0.0691797
 0.972401   0.428038  0.163025

julia> U,S,V = svd(X)

(
5x3 Array{Float64,2}:
-0.316711  0.728368  -0.179291
-0.461348  0.380697  0.530302
 0.650839  0.214743  0.450356
-0.390511  -0.455366  0.587217
-0.332809  -0.266601  -0.372815,
[2.819612693991145, 1.5674002569268684, 0.8489036938627892],
3x3 Array{Float64,2}:
-0.84953  -0.512867  0.123556
-0.06289  -0.134082  -0.988973
-0.523778  0.847932  -0.0816528)
```

Овде едноставно земаме случајна матрица со големина 5,3 ги пресметуваме со СВД алгоритамот, ова е доакз дека може секоја матрица да ја разделиме на 3 различни матрици.

Singular value decomposition (SVD)

- The connection between eigenvectors and singular vector

- For an arbitrary matrix $\mathbf{X} = \mathbf{USV}^T$, we have

$$\mathbf{X}^T \mathbf{X} = \mathbf{VS}^T \mathbf{U}^T \mathbf{USV}^T = \mathbf{V}(\mathbf{S}^T \mathbf{S}) \mathbf{V}^T = \mathbf{V} \mathbf{D} \mathbf{V}^T$$

$$(\mathbf{X}^T \mathbf{X}) \mathbf{V} = \mathbf{V} \mathbf{D}$$

$\mathbf{D} = \mathbf{S}^2$ is diagonal matrix containing the squares of the singular values

- So the eigenvectors of $\mathbf{X}^T \mathbf{X}$ are equal to \mathbf{V} , and eigenvalues to \mathbf{D}

- Similarly, $\mathbf{XX}^T = \mathbf{USV}^T \mathbf{VS}^T \mathbf{U}^T = \mathbf{U}(\mathbf{S} \mathbf{S}^T) \mathbf{U}^T$

$$(\mathbf{XX}^T) \mathbf{U} = \mathbf{U}(\mathbf{S} \mathbf{S}^T) = \mathbf{UD}$$

- In summary:

$$\mathbf{U} = \text{evec}(\mathbf{XX}^T), \quad \mathbf{V} = \text{evec}(\mathbf{X}^T \mathbf{X}), \quad \mathbf{S}^2 = \text{eval}(\mathbf{XX}^T) = \text{eval}(\mathbf{X}^T \mathbf{X})$$

Заклучок тута, дека сите три матрици можат да се претстават од сопствените вредности и сопствените вектори на матрицата \mathbf{X} , in summary

Eigenvectors are unaffected by linear scaling of a matrix => \mathbf{V} is equal to the eigenvectors of the empirical covariance $\hat{\Sigma}$

The eigenvalues of $\hat{\Sigma}$ are a scaled version of the squared singular values

Perform PCA using few lines of code

example

```
julia> X = randn(5,3)
5x3 Array{Float64,2}:
 -0.117381  1.1506  -0.948362
 0.578899  -0.22324  -0.320394
 -0.13306   -2.2004  -0.125136
 0.312237  -0.723595  2.30425
 -1.54173   0.0966524  0.586023

julia> eig(X'*X)
([2.733352021168894, 4.461496649383056, 9.06623121211592],
 3x3 Array{Float64,2}:
 0.960559  -0.248476  0.0123959
 0.182649  0.676368  -0.713558
 0.168918  0.693387  0.700486 )

julia> U,S,V = svd(X)
(
 3x3 Array{Float64,2}:
 -0.493783  -0.0709266  -0.038547
 -0.0192495  0.244762  0.281743
 0.491797  0.730029  -0.33383
 0.708828  -0.487987  0.338409
 0.107267  -0.404952  -0.832569,
 3x3 Array{Float64,2}:
 0.011018301524573, 2.1122255204838, -0.6532852207556006,
 3x3 Array{Float64,2}:
 0.0123959  0.248476  0.968559
 -0.713558  -0.676368  0.182649
 0.700486  -0.693387  0.168918)

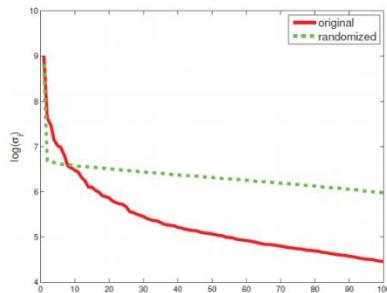
julia> X = randn(5,3)
5x3 Array{Float64,2}:
 0.154315  0.0536085  1.4482
 0.854679  -0.44341  1.15055
 -1.68437  -0.538634  -0.707005
 1.36306  -0.328046  -0.0691797
 0.972401  0.428038  0.163025

julia> U[:,1]*diagm(S[1:1])*V'[1:1,:]
5x3 Array{Float64,2}:
 0.758632  0.0561609  0.467735
 1.10509  0.0818088  0.681342
 -1.55898  -0.11541  -0.961192
 0.93541  0.0692476  0.576727
 0.797193  0.0590155  0.491509
```

Truncated SVD

- If the singular values die off quickly, we can produce rank L approximation to the matrix as follows:

$$\mathbf{X} \approx \mathbf{U}_{:,1:L} \mathbf{S}_{1:L,1:L} \mathbf{V}_{:,1:L}^T$$



```
julia> U[:,1:2]*diagm(S[1:2])*V'[1:2,:]
5x3 Array{Float64,2}:
 0.17312  -0.0969136  1.43577
 0.799057  0.00180106  1.18731
 -1.73161  -0.160541  -0.675788
 1.30146  0.164948  -0.0284765
 1.0115  0.115045  0.137183

julia> U[:,1:3]*diagm(S[1:3])*V'[1:3,:]
5x3 Array{Float64,2}:
 0.154315  0.0536085  1.4482
 0.854679  -0.44341  1.15055
 -1.68437  -0.538634  -0.707005
 1.36306  -0.328046  -0.0691797
 0.972401  0.428038  0.163025
```

Truncated SVD



Е сега наместо тоа, земам слики со ранг 20, веќе се гледа дека сликата може да се реконструира во помал ранг, тогаш ја напалувам димензионалноста на сликите така што многу може многу побрзо да манипулирам со податоците.

Ова мора да се знај долу!

Connection between SVD and PCA

- Let $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ be a truncated SVD of \mathbf{X}
- We know $\hat{\mathbf{W}} = \mathbf{V}$ and $\hat{\mathbf{Z}} = \mathbf{X}\hat{\mathbf{W}}$
- So, $\hat{\mathbf{Z}} = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V} = \mathbf{U}\mathbf{S}$
- The optimal reconstruction is given by $\hat{\mathbf{X}} = \mathbf{Z}\hat{\mathbf{W}}^T$
- So we find that, $\hat{\mathbf{X}} = \mathbf{U}\mathbf{S}\mathbf{V}^T$
- This is precisely the same as a truncated SVD approximation!
- This is another illustration of the fact that PCA is the best low rank approximation to the data.

Всушност се сведува дека ова SVD е на некој анчин ги добива истите резултати како PCA, тоа го потврдува фактот дека PCA е најдобрата апроксимација од низок ранг на податоците.

Да заклучиме ако сакаме да најдеме линеарна апроксимација каде што скриените z_i променливи се најблиску до x_i ги бараме оние чии проекции кои имаат најголема варијанса и тие со најголема варијанса во суштина се димензиите кои ги претставуваат сопствените вектори на коваријансната матрица, и SVD може да се сведе дека е еднакво на PCA каде:

V is equal to the \hat{W} of the PCA

US is equal to point \hat{Z} where PCA's X is projected on W

Probabilistic PCA

Претходно шумот го отстранувавме, но сега постои шум значи $\mathbf{x} = \mu + \sigma^2 \mathbf{z}$ и \mathbf{W} е ортогонална.

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) \triangleq \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right]$$

Probabilistic PCA

$$p(\mathbf{x}_i|\theta) = \mathcal{N}(\mathbf{x}_i|\mathbf{W}\mathbf{z}_0 + \mu, \Psi + \mathbf{W}\mathbf{z}_0\mathbf{z}_0^T)$$

- Consider a factor analysis model in which $\Psi = \sigma^2 \mathbf{I}$ and \mathbf{W} is orthogonal. The observed data log likelihood is given by:

$$\log p(\mathbf{X}|\mathbf{W}, \sigma^2) = -\frac{N}{2} \ln |\mathbf{C}| - \frac{1}{2} \sum_{i=1}^N \mathbf{x}_i^T \mathbf{C}^{-1} \mathbf{x}_i = -\frac{N}{2} \ln |\mathbf{C}| + \text{tr}(\mathbf{C}^{-1} \hat{\Sigma})$$

- Where $\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}$ and $\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T = (1/N) \mathbf{X}^T \mathbf{X}$.
(the data are centered)

- The maxima of the log likelihood are given by: $\hat{\mathbf{W}} = \mathbf{V}(\Lambda - \sigma^2 \mathbf{I})^{\frac{1}{2}} \mathbf{R}$

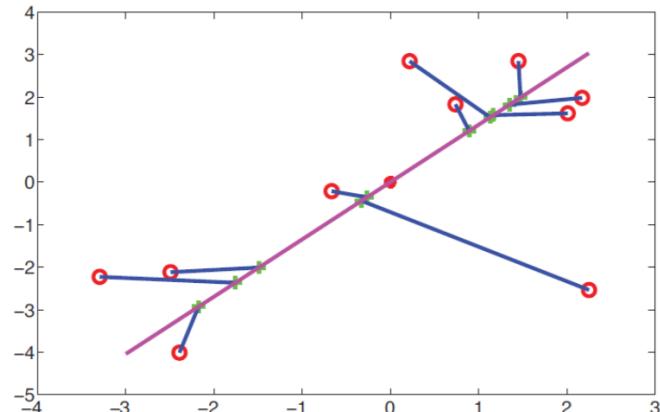
\mathbf{R} is an arbitrary $L \times L$ orthogonal matrix,

\mathbf{V} is the $D \times L$ matrix whose columns are the first L eigenvectors of $\hat{\Sigma}$

Λ is the corresponding diagonal matrix of eigenvalues.

Thus, as $\sigma^2 \rightarrow 0$, we have $\hat{\mathbf{W}} \rightarrow \mathbf{V}$, as in classical PCA.

- Note, however, that if $\sigma^2 > 0$, the posterior mean is not an orthogonal projection, since it is shrunk somewhat towards the prior mean.
- This sounds like an undesirable property, but it means that the reconstructions will be closer to the overall data mean

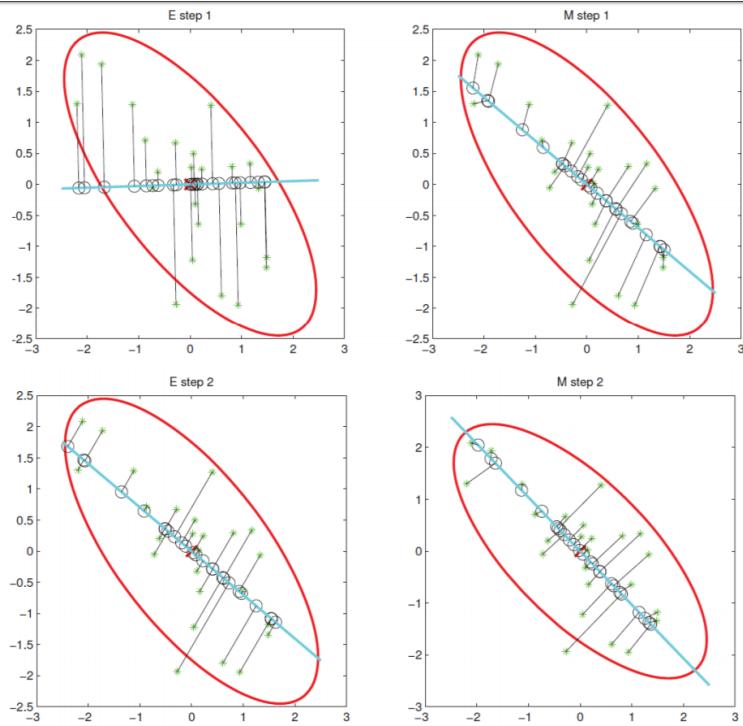


EM algorithm for PCA

$$\text{E step } \tilde{\mathbf{Z}} = (\mathbf{W}^T \mathbf{W})^{-1} \mathbf{W}^T \tilde{\mathbf{X}}$$

$$\text{M step } \mathbf{W} = \tilde{\mathbf{X}} \tilde{\mathbf{Z}}^T (\tilde{\mathbf{Z}} \tilde{\mathbf{Z}}^T)^{-1}$$

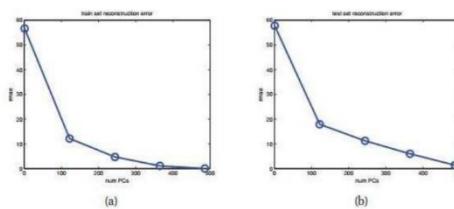
- $\tilde{\mathbf{Z}}$ is a $L \times N$ matrix storing the posterior means (low-dimensional representations) along its columns
- $\tilde{\mathbf{X}}$ stores the original data along its columns
- EM can be faster
- EM can be implemented in an online fashion, i.e., we can update our estimate of \mathbf{W} as the data streams in.



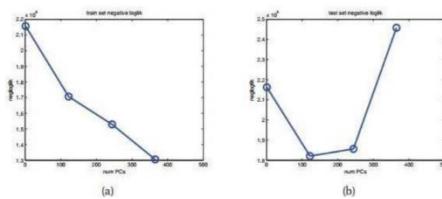
Овде битно е дека може да се употреби Expectation maximization. Овој алгоритам би го употребиле за да биди побрзо и може да додаваме податоци во Online режим, како работи интуитивно овој модел, значи прво имаме претпоставка за \mathbf{W} и ги проектираме самите податоци под прав агол, тоа е во Е чекорот, но во М чекорот ги имам проекциите, ја бараме насоката на \mathbf{W} која ќе ми ја намали грешката на растојанието помеѓу точките и реконструкциите. Понатаму Е ја имаме \mathbf{W} , од претходниот чекор добиена и сега пак ги проектираме податоците под прав агол на таа права и сега имајки ги проекциите пак ја бараме \mathbf{W} која ќе ги намали растојанијата. Ова е интуитивно EM for PCA. Бројот колку диманции ќе се тоа ќе биде бројот на скриени димензии за PCA & PPCA, така што за PCA да ја исцртаме грешката на реконструкција на тренинг множеството и множеството за тестирање.

Choosing the number of latent dimensions

- PCA
 - Plot reconstruction error
 - No U shape on test set, since it is only compression technique



- PPCA
 - Plot negative log likelihood
 - It gets “punished” if it wastes probability mass on parts of space where there is little data (Bayesian Occam’s razor effect)



Lecture 10 - Kernels

Kernels денес најмногу се користи во машинско учење заедно со мрежи, тоа е се поради тоа што се користи КЕРНЕЛ ТРИКОТ кој што користи кернел. Досега во суштина сите објекти ги претставуваме како низа од карактеристики, т.е фиксен вектор од карактеристики x_i може да не ни е толку лесно да го претставиме, односно да го најдеме тој вектор на карактеристики со фиксна големина, значи може тоа да биде проблематично на пример ако имаме текст документи со различна големина, молекуларна структура која што има некоја комплексна 3Д структура, е сега прашањето е како во суштина тие да ги претставиме со фиксен вектор на карактеристики т.е fixed-size feature vector. Наместо тоа може да користиме мерка сличност помеѓу објектите, не секој објект да го специфицирам со точен број на карактеристики туку да ме интересира сличноста помеѓу два објекти, значи да ја барам сличноста или разликата помеѓу два објекти на пример edit distance between two strings значи тоа се операции кои што се треба да се направат за од еден стринг да добијам друг стринг, значи тоа е пример каде што споредувам два различни објекти, и ако со K ја означиме мерката за сличност помеѓу x, x' објектите $\kappa(x, x') \geq 0$, за x, x' да припаѓаат на некое апстрактен простор. Сега за сега K (кернел функција) ни е мерка за сличност или различност помеѓу два објекти најчесто таа функција е симетричната што подеднакво е сличноста помеѓу x и x'' , и е ненегативна но меѓутоа тоа не е обавезно.

Kernel functions

Е сега почнуваме од кернел функции, функции кои ќе ги користиме за добивање на сличност или различност помеѓу два објекти, значи ќе ги поминиме следните кернели:

- RBF kernels – за споредување на документи
- Kernels for comparing documents
- Mercer kernels
- Linear kernels
- Matern kernels
- String kernels
- Kernels derived from probabilistic generative models

Radial basis function (RBF) kernels

The **squared exponential kernel** (SE kernel) or **Gaussian kernel** is defined by

$$\kappa(x, x') = \exp\left(-\frac{1}{2}(x - x')^T \Sigma^{-1} (x - x')\right)$$

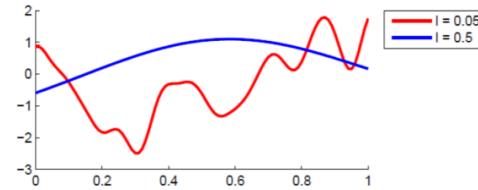
If Σ is diagonal, this can be written as

$$\kappa(x, x') = \exp\left(-\frac{1}{2} \sum_{j=1}^D \frac{1}{\sigma_j^2} (x_j - x'_j)^2\right)$$

σ_j defines the **characteristic length scale** of dimension j .

If Σ is spherical, we get the isotropic kernel, where σ^2 is known as the **bandwidth** – shared length scale (metric of the period over which local fluctuations are allowed)

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$



Ако сигма е дијагонална, види горе запис, и означива скала за карактеристична должина, и ова може да е различна се секој објект може да е заедничка или споделена најчесто се користи, таа споделена должина, и во суштина ова е рабенката RBF кернерл, каде што Сигма е споделено помеѓу сите карактеристики, и тоа ни е на некој начин мерка за колку ќе ни биде, да речиме „испеглана функцијата“ на пример ако сигма е помало ќе ни биде со многу различни пикови како на цртежот, ако пак е поголемо е поизмазента, тоа е параметар кој треба да се подеси и ни дава метрика над период кои овие локални фруктации се дозволени. Овој кернерл е default кернел кој прв би го користеле, и сега кернели за споредување на документи.

Kernels for comparing documents

Е сега ако имаме bag of words репрезентација на документите каде во суштина x_{ij} е бројот на пати зборот j да се појавил во документот i , тогаш сличноста се дефинира, како косинус на сличност:

$$\kappa(\mathbf{x}_i, \mathbf{x}_{i'}) = \frac{\mathbf{x}_i^T \mathbf{x}_{i'}}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_{i'}\|_2}$$

Оваа равенка всушност го мери косинусот, од аголот помеѓу аголот x_i and $x_{i'}$, ако двета ги претставам во оваа мерка ќе ни се даде косинус од аголот односно колку се слични или различни, x_i е број на пати, што значи не може да е негативен, косинусот оди од 0-1, каде 0 значи дека е под прав агол т.е немаат ништо заедничко, тука проблемот е тоа што вака ако претствам документ па (stop words) како ‘the’, ‘and’, може да ни создадат лажна сличност помеѓу документите поради тоа што тие се користат со сите документи, и плус имаме burstiness of word usage т.е ако еден збор се користи се користи еднаш, многу е варејатно да се користи повеќе пати во документите, така што оваа мерка сама по себе не се користи, ваквата форма не се користи многу, наместо тоа се претпоставуваат во тие **TF-IDF** “term frequency inverse document frequency” значи сега овој бројот на пати бројот да се појави во документот i , од него земаме логаритам така што ќе се намали влијанието на оние зборови кои што многу често се појавуваат во документот,

- The term frequency is defined as a log-transform of the count:

$$\text{tf}(x_{ij}) \triangleq \log(1 + x_{ij})$$

this reduces the impact of words that occur many times within one document

а од друга страна другата мерка е дадена со друга равенка значи:

document

- The inverse document frequency is defined as $\text{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^N \mathbb{I}(x_{ij} > 0)}$
- Where N is the total number of documents and denominator counts how many documents contain the word j
- Finally, we define: $\text{tf-idf}(x_i) \triangleq [\text{tf}(x_{ij}) \times \text{idf}(j)]_{j=1}^V$

Значи овде во суштина гледаме колку некој збор е заеднички и вообичаен за сите документи, ако е многу вообичаен не треба многу да му даваме за значење затоа што нема да ни направи разлика во документите нас не интересира нешто што ќе направи разлика, ако е малце споменуван во другите документи тогаш оваа вредност ќе биде поголема, така што ќе има поголемо значење. Значи ова TF-IDF место таа косинусна сличност ако замениме со tf-idf се добива следното:

We then use this inside the cosine similarity measure. That is, our new kernel has the form

$$\kappa(x_i, x_{i'}) = \frac{\phi(x_i)^T \phi(x_{i'})}{\|\phi(x_i)\|_2 \|\phi(x_{i'})\|_2}$$

where $\phi(x) = \text{tf-idf}(x)$

Mercer (positive definite) kernels

Овој тип на кернели се специјален тип на кернели, чии што карактеристики се многу важни за да може после да го примениме оној кернел трик да го примениме што го кажувавме, кој што ги прави support vector machines многу помоќни, некои функции дефинираат дека кернел функциите мора да го задоволат следното барање односно **Грам матрицата**.

$$K = \begin{pmatrix} \kappa(x_1, x_1) & \cdots & \kappa(x_1, x_N) \\ \vdots & & \vdots \\ \kappa(x_N, x_1) & \cdots & \kappa(x_N, x_N) \end{pmatrix}$$

be positive definite (symmetric with positive eigenvalues) for any set of inputs $\{x_i\}_{i=1}^N$

Овој тип на кернел е Mercer, претходните се Mercer kernel.

Е сега тоа е важно затоа што тогаш оваа Грам матрица може да направиме декомпозиција по сопствени вредности така што ќе биде како следното:

If the Gram matrix is positive definite, the eigenvalue decomposition is the following: $K = U^T \Lambda U$, where Λ is a diagonal matrix of eigenvalues, $\lambda_i > 0$

Consider an element of K: $k_{ij} = (\Lambda^{\frac{1}{2}} U_{:,i})^T (\Lambda^{\frac{1}{2}} U_{:,j})$

Let us define $\phi(x_i) = \Lambda^{\frac{1}{2}} U_{:,i}$

Then we can write $k_{ij} = \phi(x_i)^T \phi(x_j)$

We see that the entries in the kernel matrix can be computed by performing an inner product of some feature vectors that are implicitly defined by the eigenvectors U

Она што е важно е, ако Грам матрицата е positive definite тогаш може да се направи декомпозиција по споствените вредности, и понатаму секој елемент може да се претстави со некоја функција како x_i транспонирано по истата функција од x_j . Зошто пак ова е важно, затоа што постои функција f_i која што го мапира x_i , така што оваа кернел функција е следната:

In general, if the kernel is Mercer, then there exists a function ϕ mapping x such that

$$\kappa(x, x') = \phi(x)^T \phi(x')$$

Where ϕ depends on the eigen functions of k

For example, **polynomial kernel** $\kappa(x, x') = (\gamma x^T x' + r)^M$ where $r > 0$.

If, for example, $M = 2$, $\gamma = r = 1$ and $x, x' \in \mathbb{R}^2$, we have

$$(1 + x^T x')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2 \\ = 1 + 2x_1 x'_1 + 2x_2 x'_2 + (x_1 x_1)^2 + (x_2 x_2)^2 + 2x_1 x'_1 x_2 x'_2$$

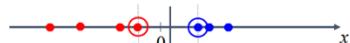
This can be written as $\phi(x)^T \phi(x')$ where $\phi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^T$

So using this kernel is equivalent to working in a 6-dimensional feature space.

Пр. Ако земам некој поминомен кернел, тој е дефиниран на овој начин, ги земам овие вредности го поедноставувам, ако го извршиме квадратот, се добива квадратната ϕ -ја.

За овој кернел може да се покаже дека може да се запиже во формата како на слайдот најдолу. $f_i(x)$ претставува некакво мапирање во нова димензија која што во овој случај е 6 деминезии пример, пример x - од 2 димензии го мапирам во простор од 6 димензии. Во тој нов простор кернел матрицата ни ја дава сличноста на двата објекти во новиот простор, поентата е дека $f_i(x)$ - мапирањето од 2-демензионален во 6-демензионален простор на секој објект воопшто не е ни потребно ако користам овој внатрешен продукт кој ни ја дава сличноста или разлика на објектите во новиот простор. Еве зошто би сакале да пресликаме во нов простор:

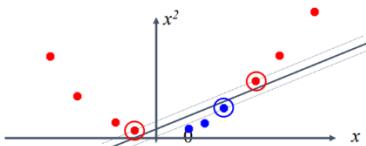
Datasets that are linearly separable with some noise:



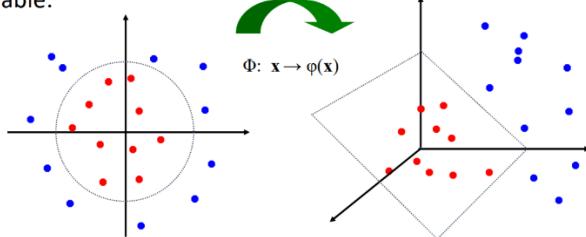
But what are we going to do if the dataset is just too hard?



How about... mapping data to a higher-dimensional space:



General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Ова би го сакале, ако имаме податочно множество кое е линеарно разделиво, како на сликата, прва, но што како сликата долу, таму не можеме со една линија да ги разелиме двете класи, е сега ако ова од една димензија го мапираме во 3 димензии може да повлечам права и да ги поделам податоците

Linear kernels

Изведувањето на векторот на карактеристики во новиот простор е тешко и е можно само кога е Mercer kernel, меѓутоа портата е што ни треба сличноста/разликата што ни треба во тој простор, е сега:

However, deriving a kernel from a feature vector is easy: we just use

$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle$$

If $\phi(x) = x$ we get the **linear kernel**, defined by

$$\kappa(x, x') = x^T x'$$

This is **useful if the original data is already high dimensional**, and if the original features are **individually informative**, e.g., a bag of words representation where the vocabulary size is large, or the expression level of many genes.

In such a case, the decision boundary is likely to be representable as a linear combination of the original features, so it is not necessary to work in some other feature space.

String kernels

Што ако имам сега некој објекти кои што се со различна должина, имаме x, x' кои имаат различни должини D, D' , и се дефинирани на некоја азбука A која има 20 букви, кога споредуваме вака стрингови тогаш кернел функцијата е дадена со оваа равенка, така што $\phi(x)$ го кажува бројот на пати дека одреден потстинг се појавува во стингот x

Let $\phi_s(x)$ denote the number of times that substring s appears in string x . We define the kernel between two strings x and x' as

$$\kappa(x, x') = \sum_{s \in A^*} w_s \phi_s(x) \phi_s(x')$$

where $w_s \geq 0$ and A^* is the set of all strings (of any length) from the alphabet A (this is known as the Kleene star operator)

This is a Mercer kernel

Variations, bag-of-characters, bag-of-words, strings of fixed length

Fisher kernel (derived from probabilistic generative models)

Suppose we have a probabilistic generative model of feature vectors, $p(x|\theta)$.

The Fisher kernel is defined as: $\kappa(x, x') = g(x)^T F^{-1} g(x')$

Where $g(x)$ is the gradient of the log likelihood, or score vector, evaluated at the MLE $\hat{\theta}$: $g(x) \triangleq \nabla_{\theta} \log p(x|\theta)|_{\hat{\theta}}$

And \mathbf{F} is the Fisher information matrix, which is essentially the Hessian

$$\mathbf{F} = \nabla \nabla \log p(\mathbf{x}|\theta)|_{\hat{\theta}}$$

The intuition behind the Fisher kernel is the following: let $g(\mathbf{x})$ be the direction in which \mathbf{x} would like the parameters to move (from $\hat{\theta}$) so as to maximize its own likelihood; call this the directional gradient. Then we say that two vectors \mathbf{x} and \mathbf{x}' are similar if their directional gradients are similar wrt the geometry encoded by the curvature of the likelihood function

Using kernels inside Generalized linear models - Kernel machines

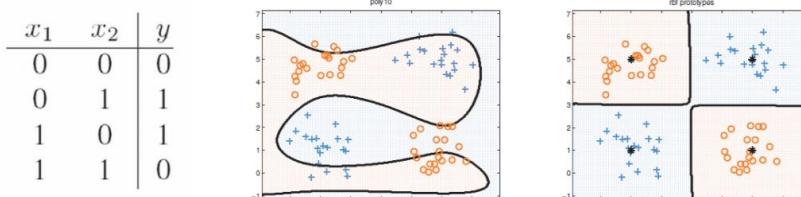
- We can define **kernel machine** with the input feature vector

$$\phi(\mathbf{x}) = [\kappa(\mathbf{x}, \mu_1), \dots, \kappa(\mathbf{x}, \mu_K)]$$

where $\mu_k \in \chi$ are a set of K **centroids**, and $\phi(\mathbf{x})$ is **kernelized feature vector**. If κ is RBF kernel, this is called RBF network.

- Kernelized feature vector for **logistic regression** $p(y|\mathbf{x}, \theta) = \text{Ber}(\mathbf{w}^T \phi(\mathbf{x}))$

- Simple way to define non-linear decision boundary

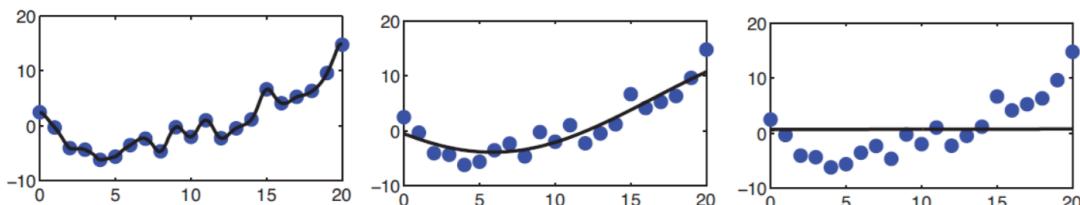


a) xor truth table. (b) Fitting a logistic regression classifier using degree 10 polynomial expansion. (c) Same model, but using an RBF kernel with centroids specified by the 4 black crosses.

Using kernels inside Generalized linear models - Kernel machines

- Kernelized feature vector inside a **linear regression** model –

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(\mathbf{w}^T \phi(\mathbf{x}), \sigma^2)$$



- Data set fit with $K = 10$ uniformly spaced RBF prototypes, but with the bandwidth ranging from small to large.

- Small values of bandwidth lead to very wiggly functions,
- If the bandwidth is very large the corresponding function is just a straight line.

За линеарна регресија, со нормална распределба како на цртежот имаме пример со 10 пртотипови, па bandwidth е различен, она што е во првиот слайд, мерка за пикови ако е премала може да е преискривена може да имаме overfitting ако е многу голема ќе биде права, тоа е мерка за overfitting.

The kernel trick – најважно во цела лекција

Кажува дека некои модели се базираат само на продуктот помеѓу векторите, $K(x_i, x_j) = x_i^T x_j$. Не ме интересира нивниот внатрешен продукт, не морам да знам секој поединечно колку е туку само вредноста на ова цело заедно, е сега ако ги мапирам, овие точки во нова димензија со фи функцијата тогаш тој продукт станува $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, сега ова се соодветствува на внатрешниот продукт на проширените простор на карактеристики, во новиот простор го имаме продуктот на двете карактеристики. Кернерл трикот е во тоа што нас не ни треба да го пресметаме $\phi(x)$ туку ми треба само овој продукт и секаде каде што продукт x_i, x_j јас го заменувам со кернел функцијата, не ги пресметувам во нова димензија туку нивниот производ го заменувам со кернел функцијата, ова бара кернелот да е Мерцер.

Rather than defining our feature vector in terms of $\phi(x)$ we can instead work with the original feature vectors x , but modify the algorithm so that it replaces all inner products of the form x, x' with a call to the kernel function, $k(x, x')$. This is called the **kernel trick**.

It turns out that many algorithms can be kernelized in this way.

Note that we require that the kernel be a Mercer kernel for this trick to work.

Пример на кернелизирани алгоритми има многу некој од нив се следните:

Kernelized nearest neighbor classification – compute the Euclidean distance of a test vector to all the training points, find the closest one and look up its label. This can be kernelized by observing that

$$\|x_i - x_{i'}\|_2^2 = \langle x_i, x_i \rangle + \langle x_{i'}, x_{i'} \rangle - 2\langle x_i, x_{i'} \rangle$$

- features produced by these feature maps can bring the points from the same class closer to each other and push points from different classes away if correct kernel is selected for the data

Kernelized K-medoids clustering – k-medoids clustering also uses Euclidean distance to measure dissimilarity, which can also be kernelized by using the same equation as in kernelized NN classification

Support vector machines

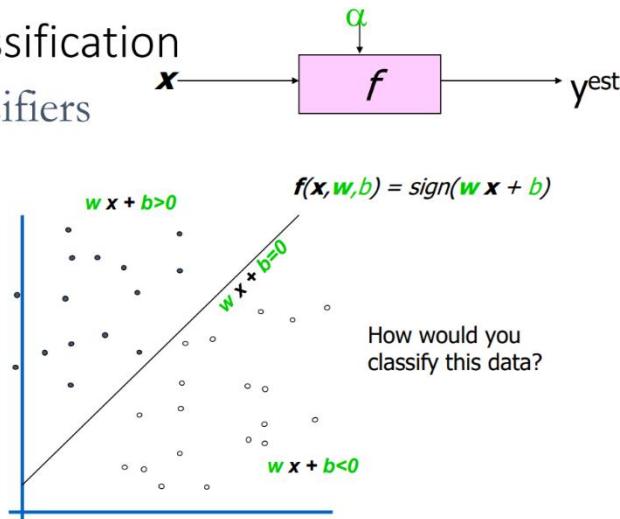
Во денепно време многу често се користат овие вектори, тоа е поради главно поради кернел трикот, прво се развиени за бинарна класификација подоцна се прошириени за регресија и мултикласна класификација, е сега се овие двете работи кои се многу важни е сега предвидувањата зависат од подмножество од податоците за тренирање, смао од некои точки од податоците на тренирање и тие точки се наречени support vectors. А другото е кернел трикот и заедно двете ги даваат support vector machines.

SVM for classification

Сега имаме податоци и треба да ги разделиме, имаме повеќе начини две различни класи од податочното множество, меѓутоа прашањето е која да се избере.

SVM for classification Linear Classifiers

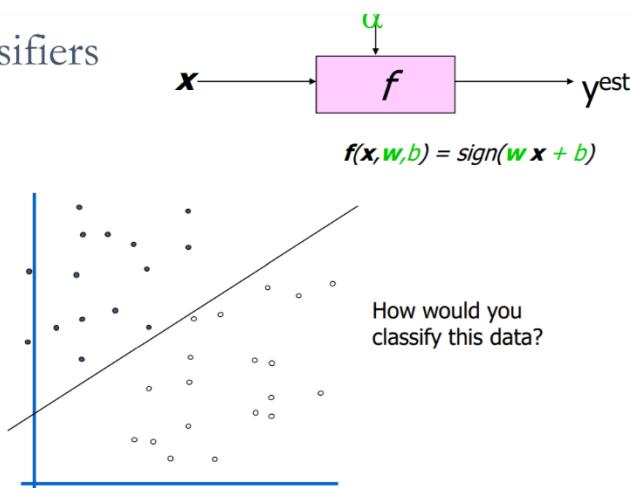
- denotes +1
- denotes -1



Ако избереме интуитивно оваа права оваа ги дели двете класи, сеа иамме нов примерок, кој не припаѓа во соодветната класа со првата права повлечена.

Linear Classifiers

- denotes +1
- denotes -1

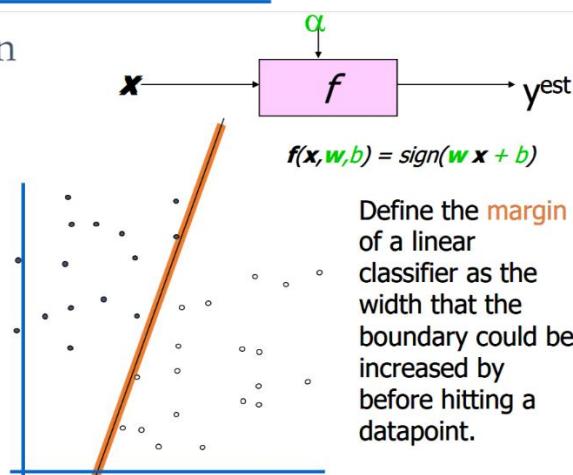


И сега како да се одбери која од овие прави е најдобра за разделување на овие две класи?

Е сега за тоа се дефинира маргина, тоа е ширината на границата на одлука

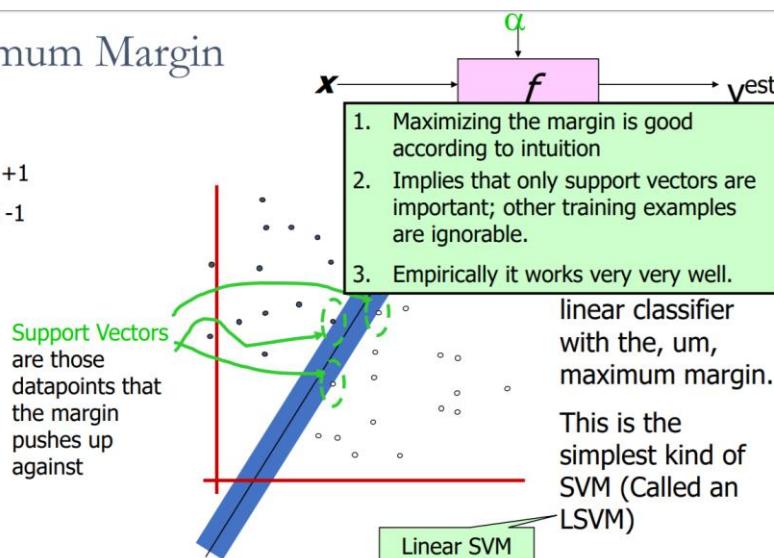
Classifier Margin

- denotes +1
- denotes -1



Maximum Margin

- denotes +1
- denotes -1



The large margin problem

- The goal is to derive a discriminant function $f(x)$ which will be linear in the feature space implied by the choice of kernel.

- Consider a point x in the induced space:

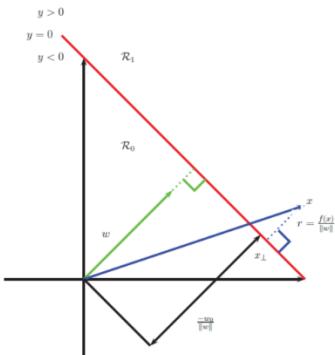
$$x = x_{\perp} + r \frac{w}{\|w\|}$$

- Where r is the distance of x from the decision boundary, whose normal vector is w and x_{\perp} is the orthogonal projection of x onto this boundary. So,

$$f(x) = w^T x + w_0 = (w^T x_{\perp} + w_0) + r \frac{w^T w}{\|w\|}$$

- Now, $f(x_{\perp}) = 0$ so $0 = w^T x_{\perp} + w_0$.

- Hence: $f(x) = r \frac{w^T w}{\sqrt{w^T w}}$, and $r = \frac{f(x)}{\|w\|}$



We would like to make the distance $r = f(x)/\|w\|$ as large as possible

In addition, we want to ensure each point is on the correct side of the boundary, hence we want $f(x_i)y_i > 0$. So the objective is:

$$\max_{w, w_0} \min_{i=1}^N \frac{y_i(w^T x_i + w_0)}{\|w\|}$$

Let us define the scale factor such that $y_i f_i = 1$ for the point that is closest to the boundary. Therefore, we want to optimize:

$$\min_{w, w_0} \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y_i(w^T x_i + w_0) \geq 1, i = 1 : N$$

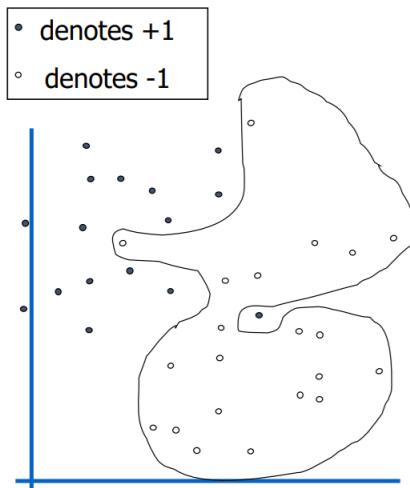
We would like to make the distance $r = f(\mathbf{x})/\|\mathbf{w}\|$ as large as possible
In addition, we want to ensure each point is on the correct side of the boundary, hence we want $f(\mathbf{x}_i)y_i > 0$. So the objective is:

$$\max_{\mathbf{w}, w_0} \min_{i=1}^N \frac{y_i(\mathbf{w}^T \mathbf{x}_i + w_0)}{\|\mathbf{w}\|}$$

Let us define the scale factor such that $y_i f_i = 1$ for the point that is closest to the boundary. Therefore, we want to optimize:

$$\min_{\mathbf{w}, w_0} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, i = 1 : N$$

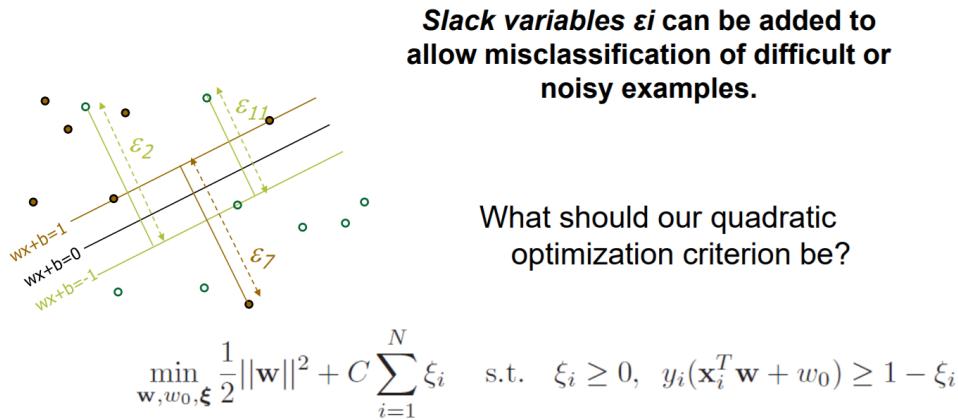
Dataset with noise



- **Hard Margin:** So far we require all data points be classified correctly
 - No training error
- **What if the training set is noisy?**
- **Solution 1:** use very powerful kernels

OVERFITTING!

Soft Margin Classification



SVMs for classification

- The optimal solution is: $\hat{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}_i$
- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector.
- At test time, the prediction is done using

$$\hat{y}(\mathbf{x}) = \text{sgn}(f(\mathbf{x})) = \text{sgn}(\hat{w}_0 + \hat{\mathbf{w}}^T \mathbf{x})$$

- If we also include the kernel trick, we get:

$$\hat{y}(\mathbf{x}) = \text{sgn}\left(\hat{w}_0 + \sum_{i=1}^N \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})\right)$$

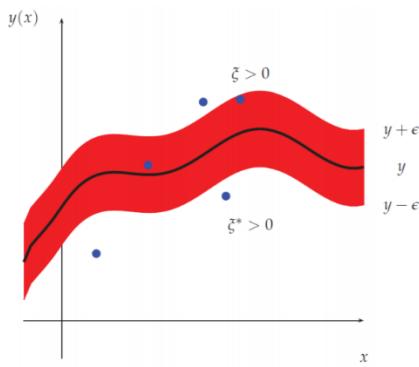
SVMs for regression

Во суштината имаме варијанта од функцијата на загуба или таа е наречена called epsilon intensive loss function

SVMs for regression

- Variant of the Huber loss function (which includes sparsity), called **epsilon intensive loss function**

$$L_\epsilon(y, \hat{y}) \triangleq \begin{cases} 0 & \text{if } |y - \hat{y}| < \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$



- Any point lying inside an ϵ -tube around the prediction is not penalized
- The corresponding objective function is usually written as:

$$J = C \sum_{i=1}^N L_\epsilon(y_i, \hat{y}_i) + \frac{1}{2} \|\mathbf{w}\|^2$$

where $\hat{y}_i = f(\mathbf{x}_i) = \mathbf{w}^T \mathbf{x}_i + w_0$ and $C = 1/\lambda$ is regularization constant

Оваа функција на загуба вели сите точки кои лежат во епсилон-цевка околу предвидувањето не се пенализирани, цената ќе биде нула ако цената е помала од епсилон

Formulate the problem as a constrained optimization problem.

Introduce **slack variables** that represent the degree to which each point lies outside the tube:

$$y_i \leq f(\mathbf{x}_i) + \epsilon + \xi_i^+ \quad \xi_i^+ \geq 0$$

$$y_i \geq f(\mathbf{x}_i) - \epsilon - \xi_i^- \quad \xi_i^- \geq 0$$

The objective function now is:

$$J = C \sum_{i=1}^N (\xi_i^+ + \xi_i^-) + \frac{1}{2} \|\mathbf{w}\|^2$$

The optimal solution has the form $\hat{\mathbf{w}} = \sum_i \alpha_i \mathbf{x}_i$

Where $\alpha_i \geq 0$. It turns that α vector is sparse, because we don't care about errors which are smaller than ϵ . The x_i for which $\alpha_i > 0$ are called **support vectors**

Once the model is trained, we can then make predictions using

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \hat{\mathbf{w}}^T \mathbf{x}$$

Plugging in the definition of $\hat{\mathbf{w}}$ we get

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \sum_i \alpha_i \mathbf{x}_i^T \mathbf{x}$$

Finally, we can represent a kernelized solution

$$\hat{y}(\mathbf{x}) = \hat{w}_0 + \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$$

Секоја точка може да лежи од таа цевка но меѓутоа треба да го минимизираме

LECTURE 11 Adaptive basis function models

Овие модели всушност пробуваат да ги научат корисните карактеристики $\phi(x)$ влезните податоци. Генералниот модел на Adaptive basis function model (ABM) изгледа така:

$$f(\mathbf{x}) = w_0 + \sum_{m=1}^M w_m \phi_m(\mathbf{x})$$

Всушност имаме M различни основни функции, кои што се учат од податоците и резултатот го добиваме како сум од множењето w и ϕ , најчесто овие функции се параметарски така што зависат од некој параметри:

Typically the basis functions are parametric – $\phi_m(\mathbf{x}) = \phi(\mathbf{x}; \mathbf{v}_m)$
where \mathbf{v}_m are the parameters of the basis function itself

The entire parameter set is $\theta = (w_0, \mathbf{w}_{1:M}, \{\mathbf{v}_m\}_{m=1}^M)$

На целиот параметарски сет е тета. Проблемот тука е што резултантниот модел не е веќе линеарен во параметрите, значи сега тука треба да најдеме само локално оптимално MLE или MAP од тета. Најчесто ваквите модели се многу подобри од линеарните модели.

Classification and regression trees

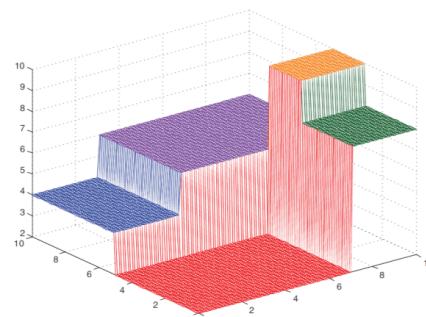
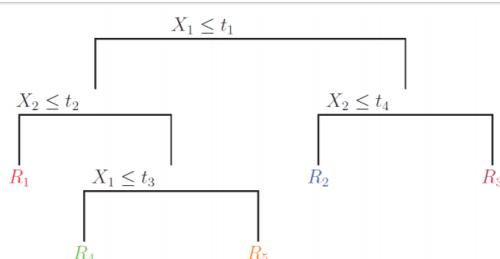
Овие дрва се дефинираат така што рекурзивно се дели просторот на влезни карактеристики и потоа се дефинира локален модел за секој од резултантните региони на влезните карактеристики, значи рекурзивно го делиме множеството и понатаму добиваме различни региони и за тие региони пресметуваме локален модел, се претставува како дрво така што имаме по еден лист за регион.

Basics

- Axis parallel split – partition 2d space into 5 regions
- Associate mean response with each of these regions

$$f(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \sum_{m=1}^M w_m \phi(\mathbf{x}; \mathbf{v}_m)$$

where w_m is the **mean response in region m** and \mathbf{v}_m encodes the **choice variable** to split on, and the **threshold value** on the path



Овде имаме две карактеристики т.е 2Д простор со x_1 и x_2 карактеристики и викаме ако x_1 е помало од некоја си вредност t_1 тогаш оди во левата гранка, па прашуваме после ако $x_2 < t_2$ го добиваме регионот R_1 . Овде во суштина го делам 2Д просторот во 5 различни региони, по која променлива се дели и која е trashhold value ќе видиме понатаму, врз основа одредувам дека некоја од овие карактеристики во случајов x_1 е најважна и прво треба да поделам по нејзе, и понатамо на x_2 и понатаму, и после имаме 5 региони, мна тие региони треба да пресметаме средна вредност на у т.е на излезната карактеристика, така што добиваме со равенката горе до цртежот. Долоклу имаме регресија ќе пресметаме среден одговор т.е средно у за сите тие региони, ако имаме класификација наместо среден одговор ќе ја зачуваме дистрибуцијата на класните лабели на сите од листовите, на пример имам вакво дрво на кое прво делиме по боја:

Generalize to classification – instead of mean response, store the **distribution over class labels in each leaf**

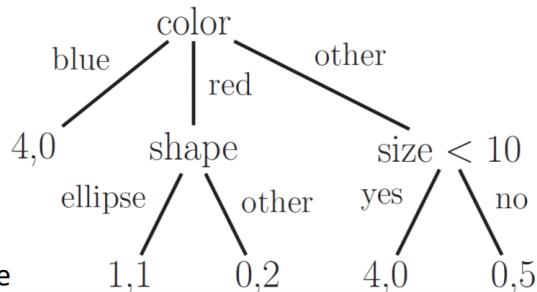
Example, color->blue, predict

$$p(y = 1|x) = 4/4$$

Color-blue, shape-other

$$p(y = 1|x) = 0/2$$

These probabilities – empirical fraction of positive examples that satisfy each conjunction of feature values



Лево кај blue имаме 4-позитивна класа 0-негативна класа, па после пресметувам ако бојата е пример плава која е веројатноста дека примерокот ќе припаѓа на позитивната класа, тука веројатноста у да е 1, ако сите припаѓаат на истата класа, или ако имаме црвена боја и облик некој си друг, тогаш која е веројатноста на припаѓа на позитивната класа, таа е 0/2. На овој начин го делам просторот го делам просторот, гледам колку од примероците, примаѓаат во секој од овие региони, ако е регресија земам средна вредност од нив, ако е класификација земам ваква распределба по класите и во суштина гледам која е веројатноста да припаѓа на едната или на другата класа, во зависност од дистрибуцијата по класи на листовите. Значи растењето на дрвото е рекурзивна процедура така што имаме функција `fitTree`, имаме почетен јазол D , и длабочина на дрвото значи овде прво е за регресија, следи алгоритамот

Greedy procedure to find the optimum partitioning of the data

Algorithm 16.1: Recursive procedure to grow a classification/ regression tree

```

1 function fitTree(node,  $\mathcal{D}$ , depth) ;
2 node.prediction = mean( $y_i : i \in \mathcal{D}$ ) // or class label distribution ;
3  $(j^*, t^*, \mathcal{D}_L, \mathcal{D}_R) = \text{split}(\mathcal{D})$ ;
4 if not worthSplitting(depth, cost,  $\mathcal{D}_L, \mathcal{D}_R$ ) then
5   return node
6 else
7   node.test =  $x_{j^*} < t^*$ ;
8   node.left = fitTree(node,  $\mathcal{D}_L$ , depth+1);
9   node.right = fitTree(node,  $\mathcal{D}_R$ , depth+1);
10  return node;

```

X_{j^*} -оптимална карактеристика

T^* - оптимален trashhold

- The split function – chooses best feature and best value for that feature

$$(j^*, t^*) = \arg \min_{j \in \{1, \dots, D\}} \min_{t \in \mathcal{T}_j} \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{ij} > t\})$$

- The set of possible thresholds \mathcal{T}_j for feature j is obtained by sorting the unique values of x_{ij} (e.g. feature 1 has values {4.5, -12, 72, -12}, then $\mathcal{T}_j = \{-12, 4.5, 72\}$)
- If inputs are categorical – the splits are of the form $x_{ij} = c_k$ and $x_{ij} \neq c_k$ for each possible class label c_k

• Stopping heuristics

- Is the reduction cost too small (normalized measure of the reduction cost):

$$\Delta \triangleq \text{cost}(\mathcal{D}) - \left(\frac{|\mathcal{D}_L|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_L) + \frac{|\mathcal{D}_R|}{|\mathcal{D}|} \text{cost}(\mathcal{D}_R) \right)$$

- Has the tree exceeded the maximum desired depth
- Is the distribution of the response in either \mathcal{D}_L or \mathcal{D}_R sufficiently homogeneous (e.g. all labels are the same, so the distribution is pure)
- Is the number of examples in either \mathcal{D}_L or \mathcal{D}_R too small

Го браме j – најдобрата карак. Т-нејзината вредност така што ќе имаме минимална цена на x_{ij} , т.е цена на разделување. Каде тие земаат само уникатните вредности, од тие вредности ќе барам која е минимална.

Тука се користат хеуристики кога да завршиме со дрвото, некои од нив се, дали намапувањето на цената е многу мала.

Stopping heuristics е важна!! Понатаму може да дадеме максимална длабочина на дрвото па кога ќе стигнеме до таа длабочина да престанеме, дали дистрибуцијата е D_L или D_R е доволно хомогена, ако е класификација ако сите примероци припаѓаат на една класа нема потреба да разделувам понатаму затоа што веќе тоа е доволно. Или бројот на примероци на D_L или D_R е прмногу мал, тогаш добинавме overfitting.

- Regression cost - $\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$

where $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$ is the mean of the response variable in the specified set of data

- Regression cost - $\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2$

where $\bar{y} = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} y_i$ is the mean of the response variable in the specified set of data

Ова е цената за регресија.

- Classification costs

- Probability a random entry in the leaf belongs to class c $\hat{\pi}_c = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i = c)$

- Misclassification rate $\frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}(y_i \neq \hat{y}) = 1 - \hat{\pi}_{\hat{y}}$

- Most probable class label $\hat{y}_c = \operatorname{argmax}_c \hat{\pi}_c$

- Entropy – amount of information

$$\mathbb{H}(\hat{\pi}) = - \sum_{c=1}^C \hat{\pi}_c \log \hat{\pi}_c$$

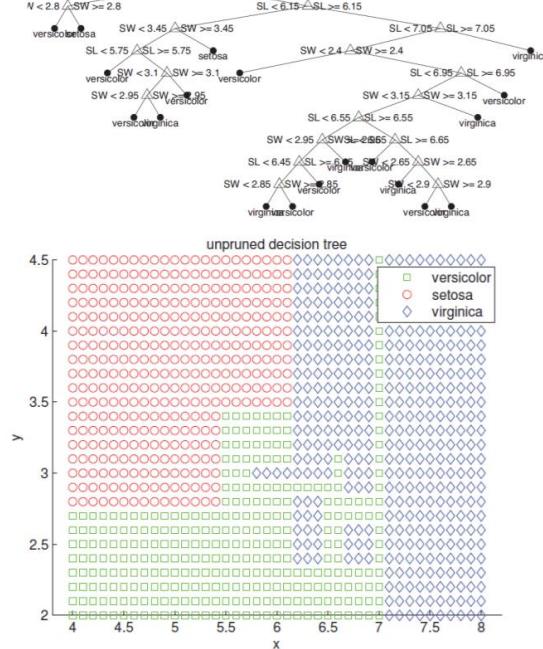
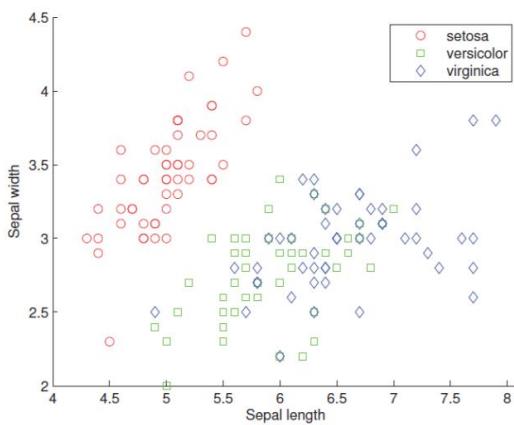
- Gini index – expected error rate

$$\sum_{c=1}^C \hat{\pi}_c (1 - \hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2$$

Ова цена е за класификација.

Example

- Iris dataset (using 2 features)

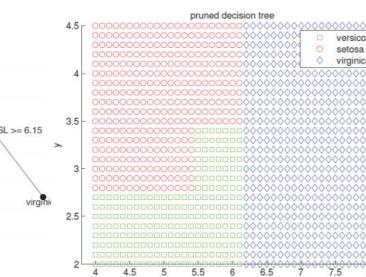
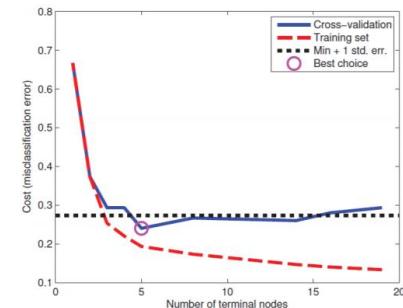
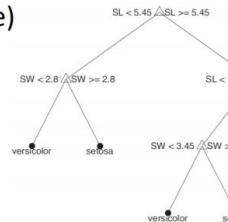


Овде ако земем 2 карактеристики, и да на правиме дрво на разделување, тоа е горе, и како ќе изгледаат границите на одлука, па може од цртежот да се забележи дека имаме overfitting. Затоа не треба да го правиме целото дрво туку треба некаде да се скрати, стандардна процедура е да се направи целото дрво

а потоа да се крати дрвото, за да се избегне overfitting, значи треба да се скратат оние гранки кои даваат најмалце придонесуваат за грешка во предвидувањето.

Pruning a tree

- Standard approach – grow a “full” tree and then perform **pruning** to avoid overfitting
 - Prune the branches giving the least increase in the error
 - How far to prune back? - evaluate the cross-validated error on each such subtree, and then pick the tree whose CV error is within one standard error of the minimum (heuristic based on sense)
 - Choose the simplest model whose accuracy is comparable with the best model



За да знаем кога треба да прекиниме со кратење на дрвото, треба да пресметаме cross validation на секоја од тие поддрва, и да се избере она кое има најмала су грешка. Значи треба да го избереме најдобриот модел кој е во рамките на најдобриот модел плус една стандардна грешка, бидејќи не мора секогаш да го избираме најминималното но доволно е да го избереме наједноставниот модел плус една стандардна грешка, на цртежот горе заокрижено е. Е сега ако ја земеме оваа вредност тогаш за бројот на крајни јазни го имаме дрвото кое го добиваме за ирис податочното множество, за 5 јазли добиваме 5 листови.

Pros and cons of trees

- Pros
 - Easy to interpret
 - Easily handle mixed discrete and continuous inputs
 - Perform variable selection
 - Scale well to large data sets
- Cons
 - Do not predict very accurately compared to other models
 - The trees are **unstable** – small changes in the input data can have large effect on the structure of the tree, (errors at the top affect the rest of the tree)
 - The trees are high variance estimators

Bootstrap Aggregation / Bagging

Сега тука го делиме податочното множество на m различни подмножества, кои што се бираат по сличен избор со displacement (еден примерок може да се избере двапати во истото податочно множество) пример имаме 10 примероци и го дела на две групи, може во едната група да го имам два пати или три пати истиот примерок. И секако за секој податочни m делови во множествата тренираме посебна копија на дрво и ќе имам m такви различни дрва, и понатаму правам предвидување по просекот од предвидувањата на секое од дрвата, значи го делам податочното множество со displacement за секое од нив креирај дрво и понатаму земам средна вредност од резултатите. Ако грешките се некорелирани, се намалува линеарно со M .

Average prediction over copies

$$f(x) = \frac{1}{M} \sum_i f_m(x)$$

Тука имаме проблем ако го тренираме ваквиот алгоритам на повеќе множества кои што ги имаат истите податоци, креира корелирани грешки т.е тие грешки што се прават се корелирани, за тоа се користи методата random forest, каде не само да бират рандом множества од тренинг податоците него исто и подмножество на променливи и сега го делат и податочното множество на различни дрва и променливите ги делиме на разлилните дрва, не секое дрво ќе ги има сите можни променливи, врз основа на тоа креирајме повеќе дрва а резултатот ќе биде средбата вредност од сите тие дрва, овој алгоритам има добра точност, и се губи таа интерпретабилност, и се широко користени.

Feedforward neural networks

Основната верзија на невронските мрежи е едно ставно низа на логистички регресиони модели, кои што се ставаат еден до друг, така што имаме логистички регресиони модели така што последниот слој ќе биде или логистичка регресија или линеарна регресија во зависност од проблемот што го разгледуваме ако е регресија последниот слој ќе е линеарна регресија ако е класификација и последниот слој ќе биде логистичка регресија. Земаме оној модел кој го знаеме за логистичка регресија и на него додаваме само повеќе вакви модели и повеќе поврзани меѓу себе.

- For example, if we have two layers, and have a regression problem, the models has the form

$$\begin{aligned} p(y|x, \theta) &= \mathcal{N}(y|\mathbf{w}^T \mathbf{z}(\mathbf{x}), \sigma^2) \\ \mathbf{z}(\mathbf{x}) &= g(\mathbf{V}\mathbf{x}) = [g(\mathbf{v}_1^T \mathbf{x}), \dots, g(\mathbf{v}_H^T \mathbf{x})] \end{aligned}$$

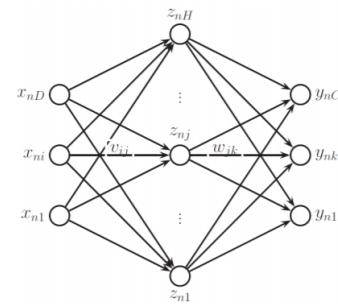
where g is a non-linear **activation** or **transfer** function (usually logistic), $\mathbf{z}(\mathbf{x})$ is hidden layer (deterministic function on the inputs), H is number of hidden units, \mathbf{V} – weight matrix from the inputs to the hidden nodes, and \mathbf{w} is weight vector from the hidden nodes to the output

Се што кажувавме за логистичка регресија важи за невронски мрежи.

g- функција на активација треба да биде нелинеарна. Ако е линеарна добиваме еден голем проблем од линеарна регресија, од оваа форма долу, значи затоа функциите на активација во скриените слоеви се нелинеарни.

- It is important that g be non-linear, otherwise the whole model collapses into a large linear regression model of the form $y = \mathbf{w}^T(\mathbf{V}\mathbf{x})$
- It can be shown that MLP is **universal approximator** meaning it can model any suitably smooth function, given enough hidden units, to any desired accuracy
- To handle binary classification, we pass the output through a sigmoid:

$$p(y|\mathbf{x}, \theta) = \text{Ber}(y|\text{sigm}(\mathbf{w}^T \mathbf{z}(\mathbf{x})))$$



Компјутерска визија: Детекција на автомобил



Автомобил

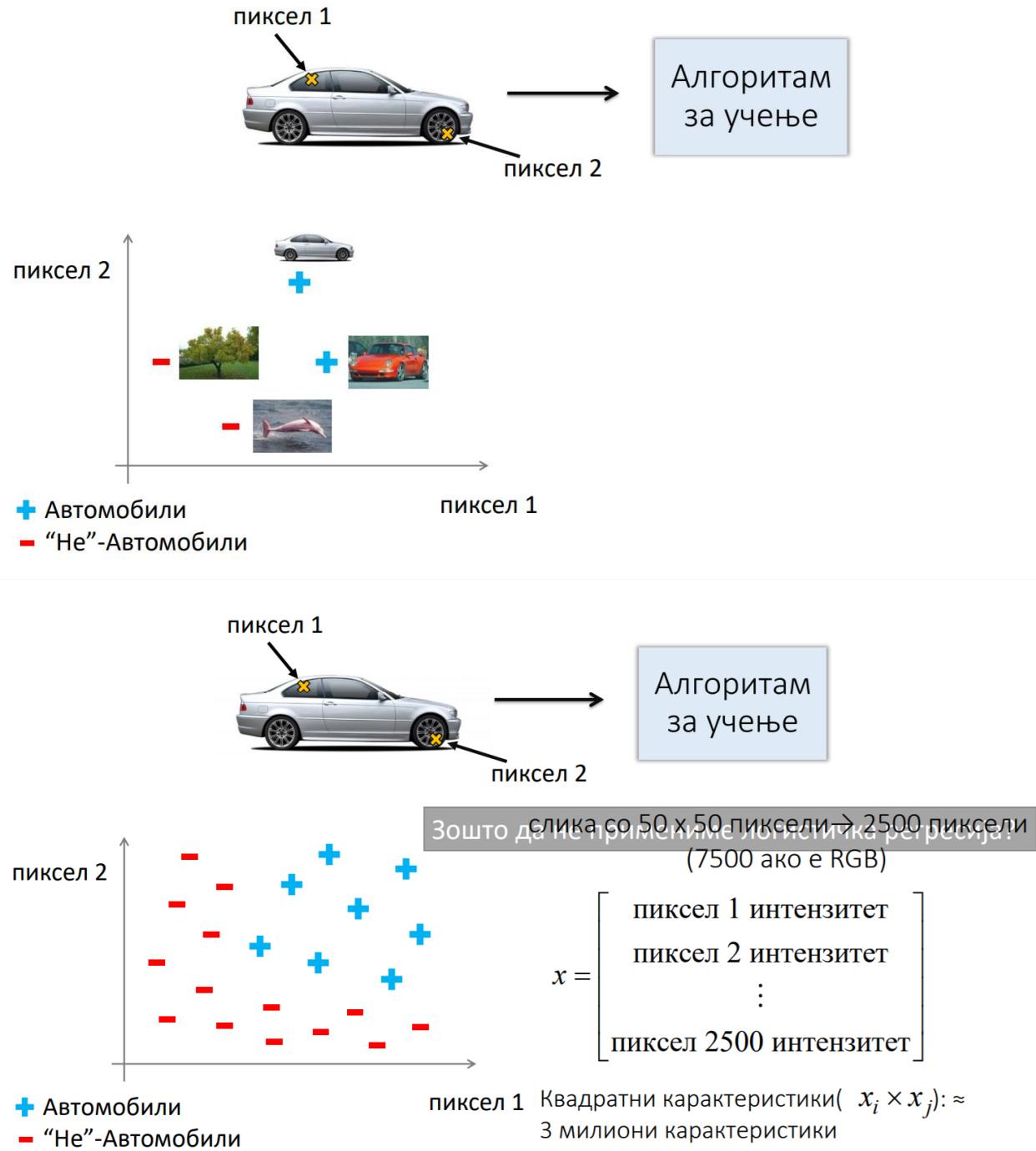


Не е автомобил



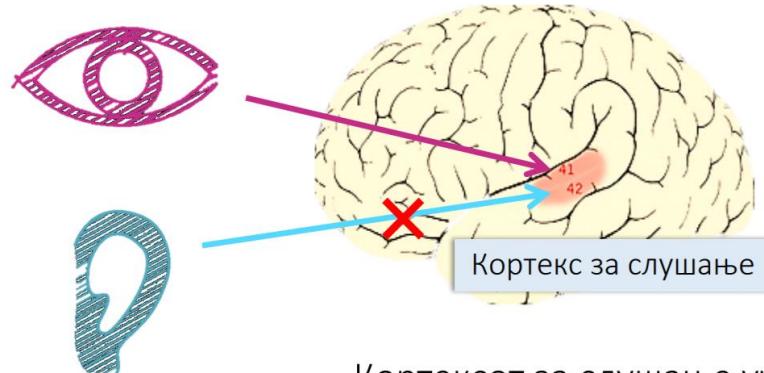
Тестирање:

Што е ова?



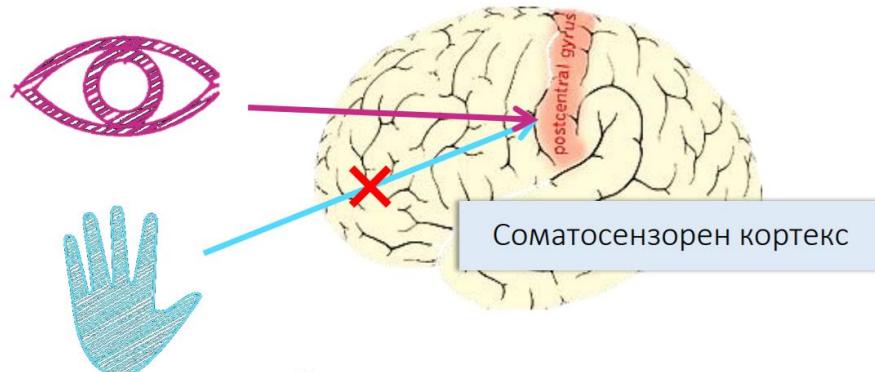
Не може логистичка бидејќи ќе добијеме премногу карактеристики, и заради тоа тука не користиме логистичка регресија. Идејата за невронски мрежи всушност била инспирирана од човечкиот мозок. И има хипотеза дека во нашиот мозок има единствен алгоритам за учење, само ги менуваме тежините и врз основа на тоа се менува резултатот.

Хипотезата на “единствен алгоритам за учење”



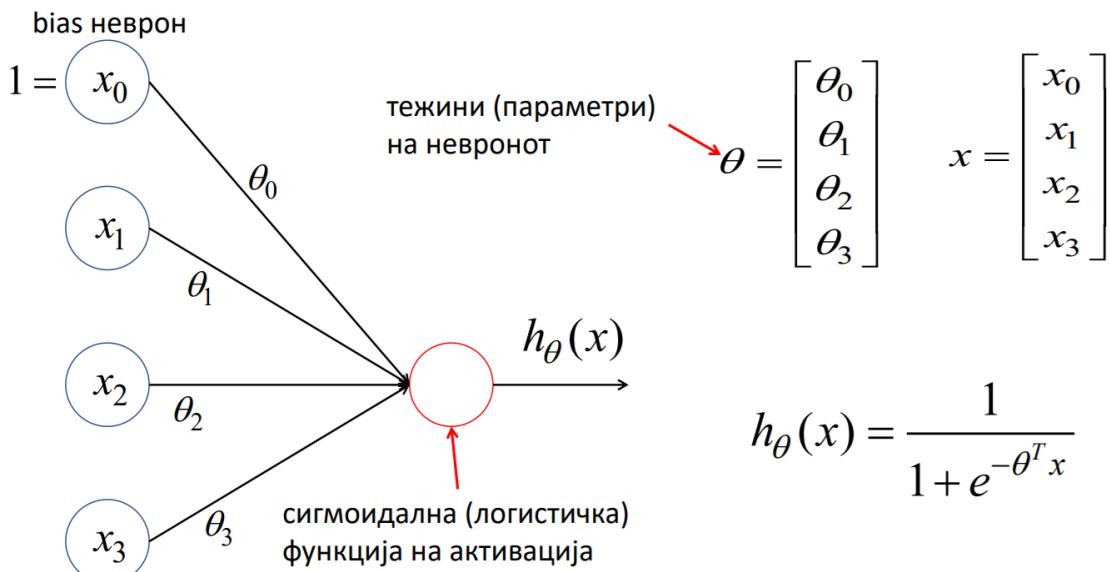
Кортексот за слушање учи да гледа

Хипотезата на “единствен алгоритам за учење”

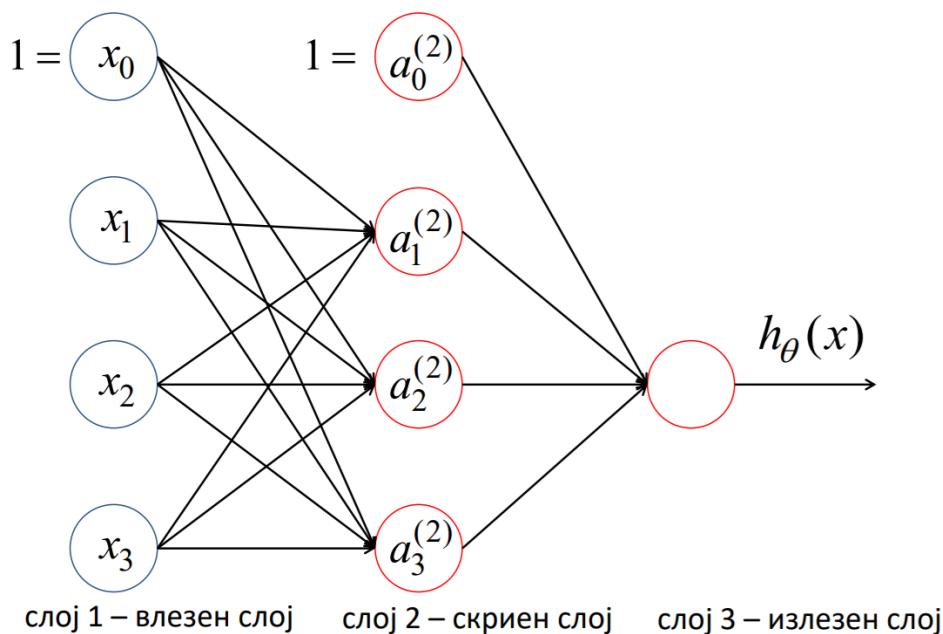


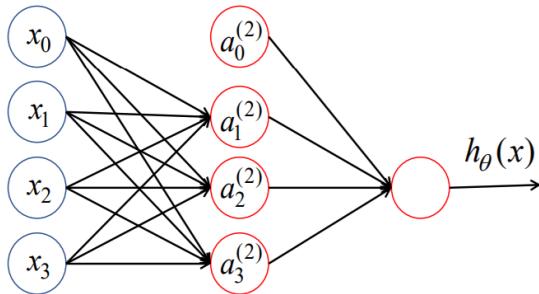
Соматосензорниот кортекс учи да гледа

Модел на неврон (логистичка единица)



Модел на невронска мрежа





$a_i^{(j)}$ - активација на невронот i

во слојот j

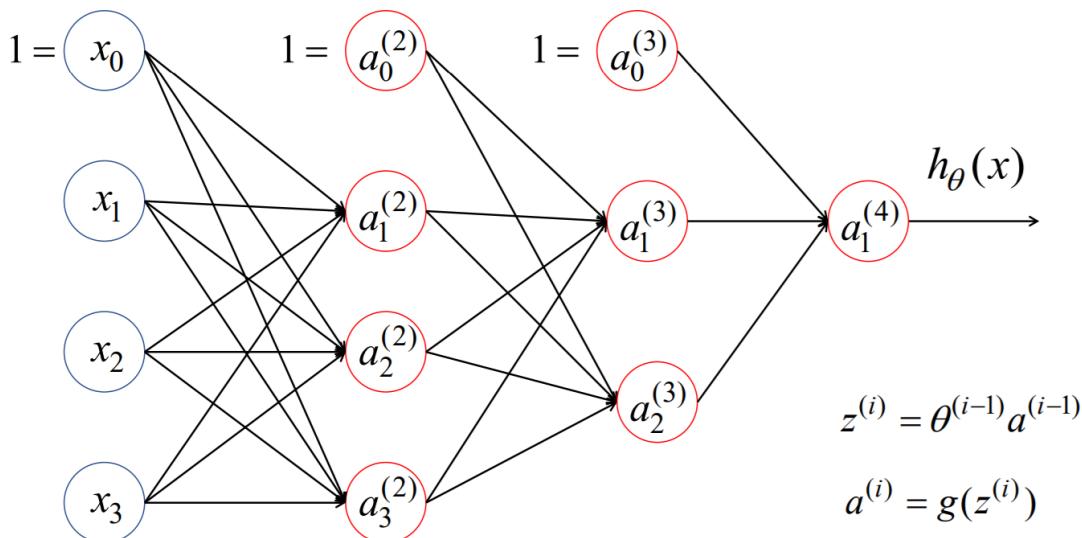
$\theta^{(k)}$ - матрица на тежини за врските помеѓу слојот k и слојот $(k+1)$

$$a_1^{(2)} = g(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3)$$

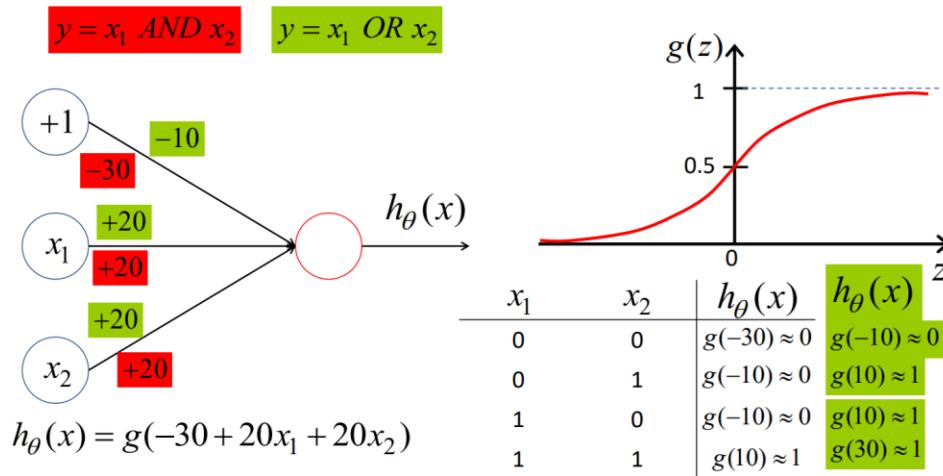
$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)})$$



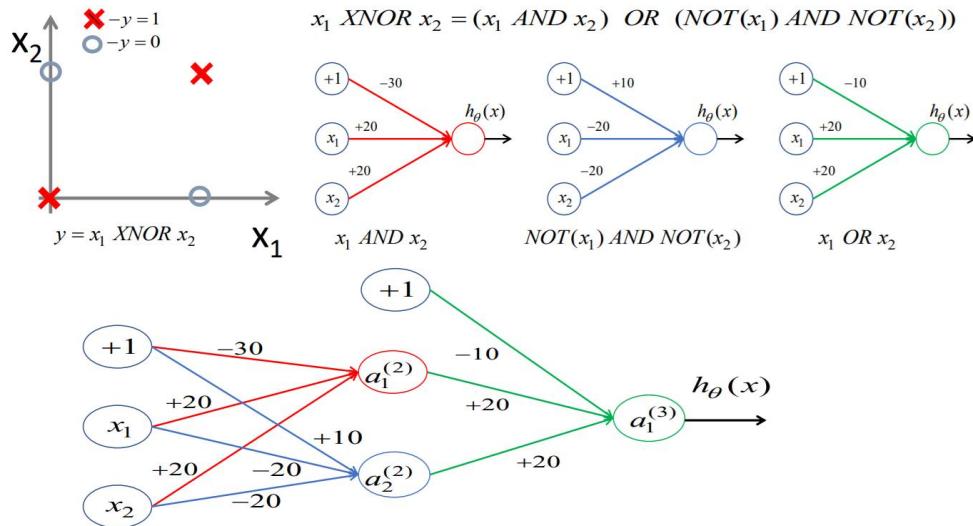
$$z^{(i)} = \theta^{(i-1)}a^{(i-1)}$$

$$a^{(i)} = g(z^{(i)})$$

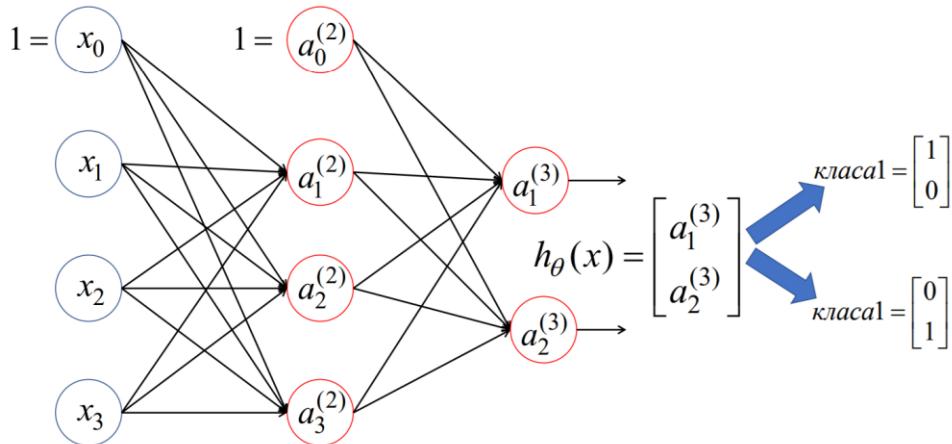
Едноставни примери на невронски мрежи: AND, OR



Пример на нелинеарна класификација: XOR/XNOR



Повеќекласна класификација



Backpropagation algorithm

The negative log likelihood of an MLP is a **non-convex function** of its parameters

But we can find a **locally optimal** ML or MAP estimate using standard gradient-based optimization methods

Since MLPs have lots of parameters, they are often **trained on very large data sets**

It is common to use **first-order online methods**, such as stochastic gradient descent

We will learn how to compute the gradient vector of the NLL by applying the chain rule – resulting algorithm is the backpropagation algorithm

For simplicity, we shall assume a model with just **one hidden layer**

Distinguish the pre- and post-synaptic values of a neuron, that is before and after we apply the nonlinearity

Input-to-hidden layer

- \mathbf{x}_n - n 'th input data
- $\mathbf{a}_n = \mathbf{V}\mathbf{x}_n$ - pre-synaptic hidden layer
- $\mathbf{z}_n = g(\mathbf{a}_n)$ – post-synaptic hidden layer (g -transfer function)

Hidden-to-output layer

- $\mathbf{b}_n = \mathbf{W}\mathbf{z}_n$ - pre-synaptic output layer
- $\hat{\mathbf{y}}_n = h(\mathbf{b}_n)$ – post-synaptic output layer

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

The parameters of the model are $\theta = (\mathbf{V}, \mathbf{W})$, first and second layer weight matrices

The bias terms are accommodated by adding an element of \mathbf{x}_n and \mathbf{z}_n to 1

In the regression case, with K outputs, the NLL is given by squared error:

$$J(\theta) = - \sum_n \sum_k (\hat{y}_{nk}(\theta) - y_{nk})^2$$

In the classification case, with K classes, the NLL is given by the cross entropy:

$$J(\theta) = - \sum_n \sum_k y_{nk} \log \hat{y}_{nk}(\theta)$$

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

Backpropagation algorithm

- Our task is to compute $\nabla_{\theta} J$.
- We will derive this for each n separately; the overall gradient is obtained by summing over n
- Start by considering the output layer weights:

$$\nabla_{\mathbf{w}_k} J_n = \frac{\partial J_n}{\partial b_{nk}} \nabla_{\mathbf{w}_k} b_{nk} = \frac{\partial J_n}{\partial b_{nk}} \mathbf{z}_n = \delta_{nk}^w \mathbf{z}_n$$

since $b_{nk} = \mathbf{w}_k^T \mathbf{z}_n$ and $\delta_{nk}^w = (\hat{y}_{nk} - y_{nk})$ is the error signal

- So the overall gradient is: $\nabla_{\mathbf{w}_k} J_n = \delta_{nk}^w \mathbf{z}_n$

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

Backpropagation algorithm

- For the input layer weights, we have $\nabla_{\mathbf{v}_j} J_n = \frac{\partial J_n}{\partial a_{nj}} \nabla_{\mathbf{v}_j} a_{nj} \triangleq \delta_{nj}^v \mathbf{x}_n$
where $a_{nj} = \mathbf{v}_j^T \mathbf{x}_n$
 - For the first level error signal we have:
- $$\delta_{nj}^v = \frac{\partial J_n}{\partial a_{nj}} = \sum_{k=1}^K \frac{\partial J_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial a_{nj}} = \sum_{k=1}^K \delta_{nk}^w \frac{\partial b_{nk}}{\partial a_{nj}}$$
- Now $b_{nk} = \sum_j w_{kj} g(a_{nj})$, so $\frac{\partial b_{nk}}{\partial a_{nj}} = w_{kj} g'(a_{nj})$
 - Hence $\delta_{nj}^v = \sum_{k=1}^K \delta_{nk}^w w_{kj} g'(a_{nj})$ $g'(a) = \frac{d}{da} \sigma(a) = \sigma(a)(1 - \sigma(a))$
 - The layer 1 error can be computed by passing the layer 2 error back

Putting it all together

- First perform a forward pass to compute $\mathbf{a}_n, \mathbf{z}_n, \mathbf{b}_n, \hat{\mathbf{y}}_n$
- Compute the error for the output layer $\delta_n^{(2)} = \hat{\mathbf{y}}_n - \mathbf{y}_n$
- Compute the error for the hidden layer $\delta_n^{(1)}$ by passing the error backwards through \mathbf{W} and using

$$\delta_{nj}^v = \sum_{k=1}^K \delta_{nk}^w w_{kj} g'(a_{nj})$$

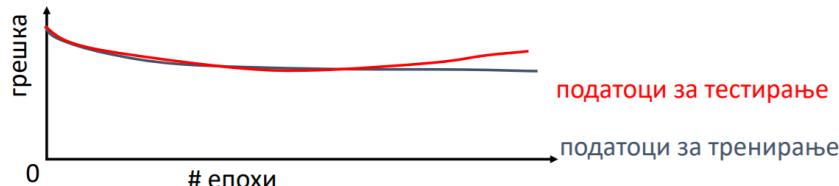
- Compute the overall gradient as follows $\nabla_{\theta} J(\theta) = \sum_n [\delta_n^v \mathbf{x}_n, \delta_n^w \mathbf{z}_n]$

Коментари за алгоритамот за тренирање

- Нема гаранции дека ќе конвергира кон нулева грешка, може да конвергира во локален оптимум или да влезе во бесконечни осцилации.
- Во пракса, конвергира кон многу мали грешки за големи мрежи врз реални податоци.
- За тренирање на големи мрежи можат да бидат потребни многу епохи (илјадници) што може да трае дури со денови.
- За да избегнете проблеми со локални минимуми испробајте неколку варијанти со различни случајни тежини (*random restarts*)
 - вредностите да бидат близки до нула $[-\epsilon, +\epsilon]$

Спречување на преголемо тренирање

- Многу голем број на епохи може да резултира во over-fitting.



- Дел од податоците поставете го како множество за валидација и тренирајте ја прецизноста после секоја епоха. Запрете со тренирањето кога дополнителните епохи ќе почнат да ја зголемуваат грешката на валидација.
- За да избегнете губење на податоци за тренирање поради валидација:
 - Користете внатрешна за множеството за тренирање (валидација) за да го пресметате просечниот број на епохи што ја максимизира прецизноста на генерализацијата.
 - Крајното тренирање направете го со целото множество за тренирање користејќи го овој број на епохи.

Определување на најдобриот број на скриени неврони

- Вообичаено само еден скриен слој
 - ако има повеќе слоеви во секој слој ист број неврони
- Премалку скриени неврони можат да го оневозможат соодветното прилагодување кон податоците.
- Премногу скриени неврони можат да доведат до over-fitting.



- Користете внатрешна CV за емпириски да го определите оптималниот број на скриени неврони.

