

Week 1

1. Bayesian inference, estimators and posterior summaries

- Traditionelt når man ønsker eksempelvis at **fitte en model** til noget data vælger man en model og en loss funktion of finder det/et set parametre der minimerer loss funktionen og lader det være ens endelig model.
- **Nødvendigvis ikke kun et sæt parametre der moddelerer dataen** godt men en række af dem, men de leder alle til forskellige predictions når vi bruger modellen. Så hvorfor modellerer vi ikke bare gode sæt af parametre.
- **Det er præcis hvad bayesian forsøger.** Istedet for kun et sæt modellerer man dem alle ved en sandsynlighedsfordeling over hver parameter vi ønsker at bestemme.
- **Marginalization. Når vi ønsker at lave predictions tager vi et weighted average:**

$$y^* = \int f(\mathbf{x}^* | \mathbf{w}) p(\mathbf{w} | \mathbf{y}) d\mathbf{w} \quad y^* = \sum_{i=1}^M f(\mathbf{x}^* | \mathbf{w}_i) p(\mathbf{w}_i | \mathbf{y})$$

over alle vores parametre, herved tager vi højde for usikkerhederne.

- **Hvordan udfører vi baysian inference:** Vi modelerer alle parameter og data som sandsynligheder og gør brug af bays rule:

$$p(\mathbf{w} | \mathbf{x}) = \frac{p(\mathbf{x} | \mathbf{w}) \cdot p(\mathbf{w})}{p(\mathbf{x})}$$

- **p(x|w) er vores likelihood:** givet et sæt parametre w hvad er sandsynligheden for at observere dataen. Det er et udtryk for hvad daten kan fortælle om os hvad vores parametre skal være.
- **p(w) er vores prior:** den er et udtryk for hvad vi tror om vores parametre før vi observerer dataen x.
- **p(x) er marginal fordeling over vores data x:** Den kan vi ikke rigtig ændre den er givet er egentlig bare en normalisering konstant der sikrer det er en gyldig sandsynlighed. Derfor skriver vi også ofte bayes some en proportionalitet hvor vi ser bort fra p(x)
- **p(w|x) er vores posterior** og udtrykket alt vi ved om vores parametre efter at have observeret dataen x.
- **Hvordan vælger vi priors?** Viden om parametre før vi går i gang med modelleringen af vores nye data. Ofte er de bare svage og ret uninformative, eller også er de valgt pga matematisk ulejlighed. EKS conjugate priors.
- **(Hvorfor priors så:** Virke regulariserende så man ungar overfitting og at man bliver for naiv. Gør modellen mere robust mod outliers)

Beta binomialmodel hvad bruge den til: Er en meget oplagt model til at modelerer og sammenligne proportioner. Eks. bias coinflip eller reklamevisning/reklameklik.

Motivation for beta binomial: hvis vi observerer 10 visninger og ingen klik og vi bare brugte en binomialfordeling ville vi med MLE estimere parametren mu til at være 0. Chancen for at reklamen giver et klik simpelthen 0. Det er meget naivt og tydeligvis ikke korrekt.

Bayesian treatment: Hvis vi nu også benytter os af en prior, så vil den få os til at holde igen sørge for vi ikke var for naive. Dette kunne eksempelvis være betafordelingen, da den netop er en fordeling over sandsynligheder mellem 0 og 1, som jo er den proportion vi forsøger at modellere.

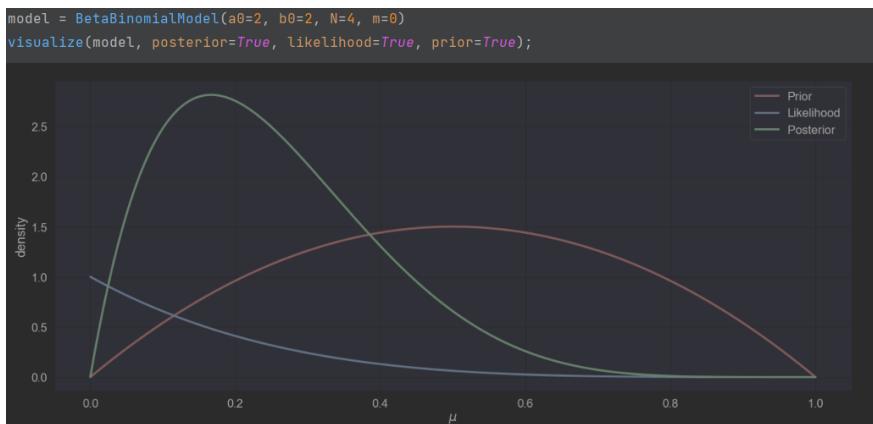
Beta fordelingen af en conjugate prior: Når vi ganger vores binomialfordeling/likelihood sammen med vores betaprior får vores posterior samme funktionelle form som vores prior, den bliver også en beta fordeling. Skulle vi observere nyt data kan vi blot bruge vores gamle posterior som vores nye prior.

$$p(\mu) = \text{Beta}(\mu|a_0, b_0) \quad (\text{Prior})$$

$$p(m|\mu) = \binom{N}{m} \mu^m (1-\mu)^{N-m} \quad (\text{Likelihood})$$

$$p(\mu|m) = \text{Beta}(\mu|a_0 + m, b_0 + N - m) \quad (\text{Posterior})$$

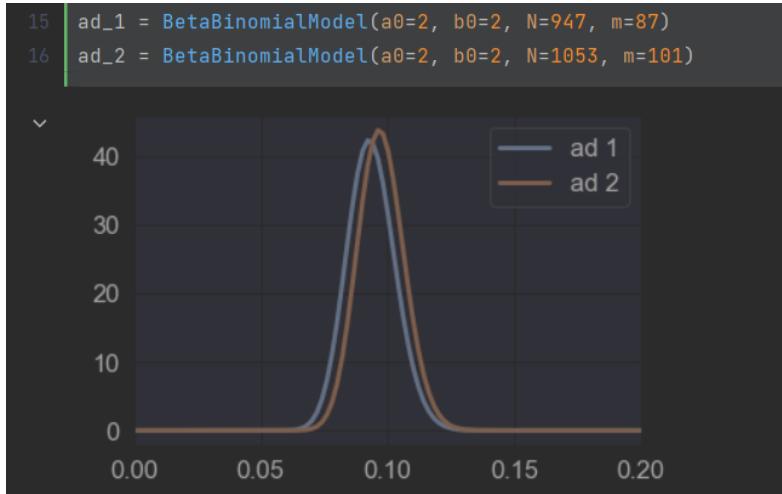
Det der er rigtig smart er at a_0 og b_0 svarer til pseudo observationer.
a=succes, beta=failures. mean= a_0 / (a_0+b_0)



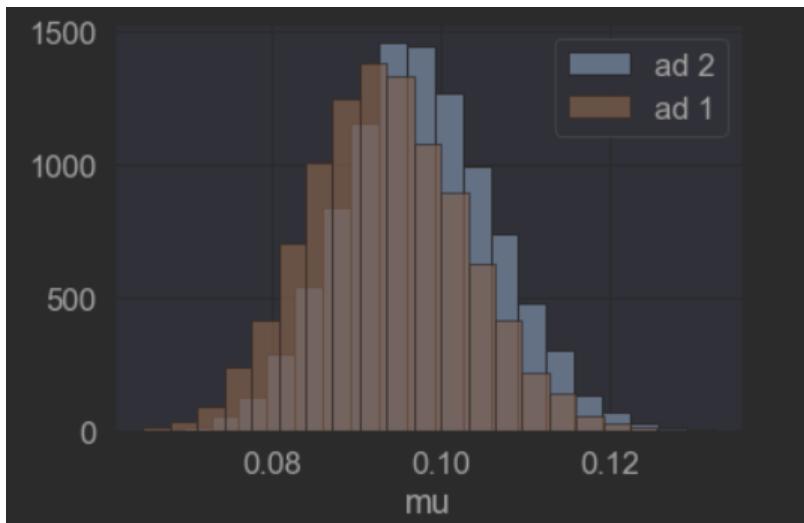
- **Observerer 4 trials med 0 succeser.** Vores MLE siger at at succesraten er 0 nul men vores prior siger vi har 4 seudo observationer, 2 af var succes så den holder igen.
- **Usikkerheder på mu eksakt.** Fordi at vi analytisk kan beregne vores posterior kan vi eksakt beregne conf-int på vores posterior ved at bruge PPF af betafordelinge. point percent function = inverse cummulative fordeling. jeg vil vide hvilken mu værdi der giver mig 2.5% af den cummulative sandsynlighed.

- Måske har vi ikke en ppf men vi kan sample fra vores posterior. Da kan vi istedet bare sample fra den og lave montecarlo estimator. Mean bliver bare sample avg. og conf ints bliver blot 2.5 og 97.5 quantiler.

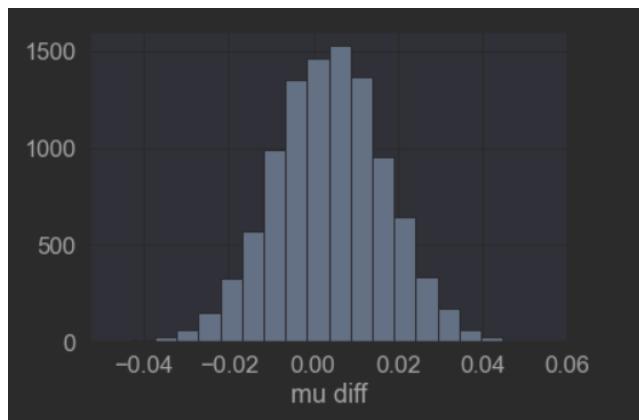
- A/B testing er en reklame bedre end en anden:



- Vi observerer det ovenstående. vi modellerer det og sampler:



- Vi modelerer forskellen mellem de 2 (mellem deres samples):



- **Hvad er sandsynligheden for at ad 2 har en højere click rate end ad 1.**
Det ville være at sige hvor meget af sandsynlighedsmassen ligger til højre for 0. Det kan vi igen blot beregne med samples. og det viser sig at være 61%
Så meget lav, da det svarer til at der er 39% chance for at add1 er bedre end 2.

Week 2

1. Bayesian linear regression
2. Model selection using the marginal likelihood

- **Normal linear regression:** Sædvanligvis har vi givet et datasæt bestående af parvise observationer af x og t : Vi vil gerne bruge vores x (som meget vel kunne være en vektor af features) til at modelere t . Modellen for signalet y kan udtrykkes ved følgende lineære sammenhæng:

$$y(\mathbf{x}|\mathbf{w}) = \sum_{j=0}^M w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

target t har også et støj komponent uover signalet y :

$$t_i = y(\mathbf{x}_i|\mathbf{w}) + \epsilon_i$$

- **Hvor theta er designmatricen.** For lineær regression vil det blot svare til at append et 1 til x vektoren hvorved vores bias/intercept fremkommer. Men theta muliggør også ikke linear modelling af x . Det kunne eksempelvis være x i anden eller cos/sinus/exp af x .
- **Sædvanligvis løses problemet ved MLE.** Dette svarer til at minimere the sum of squares error. Her ved finder vi et sæt vægte. (Og kan gøres meget effektivt ved at opstille normal equations som blot formuleres som matrix vektor produkt)
- **MLE har en tilbøjelig til at overfitte** og hvis man introducerer regularization kan det være svært at afgøre hvor stærkt man skal gøre det eller hvor (ridge regression/ lambda) og det kan være svært at afgøre hvor kompleks modellen skal være. Hvor mange ikke linære termer/polynomier man skal benytte i designmatricen.
- **Bayesian Linear regression forsøger at løse de her problemer:**

$$p(\mathbf{w}|t) = \frac{p(t|\mathbf{w})p(\mathbf{w})}{p(t)} = \frac{\mathcal{N}(t|\Phi\mathbf{w}, \beta^{-1}\mathbf{I})\mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})}{p(t)}$$

- **Vi ser at likelihood termet er det samme som før.**
- **Men nu har vi også introduceret en zero mean isotropic prior.** Denne prior er en conjugate prior og det betyder at vores posterior ligeledes bliver en normalfordeling.
- **Den marginale likelihood $p(t)$ afhænger ikke vægtene.** Vi kan som udgangspunkt se bort fra den og ofte skrives Bayes rule blot som en proportionalitet.
- **Alpha er prior precision of the regression weights.** Forøgning af alpha svarer til at præcisionen vægte forøges. Vi stoler altså mere på vægtene og den normalfordeling de følger må da være smallere/mere spiked.

- **Beta er prior pressure of the data/measurements.** Hvis beta er stor stoler vi meget på vores data og regner med at det primært er signal og meget lidt støj, herved må vores likelihood have en meget lille spredning.

- **Maximum a posteriori (MAP)** svarer blot til ridge regression med lambda = alpha/beta

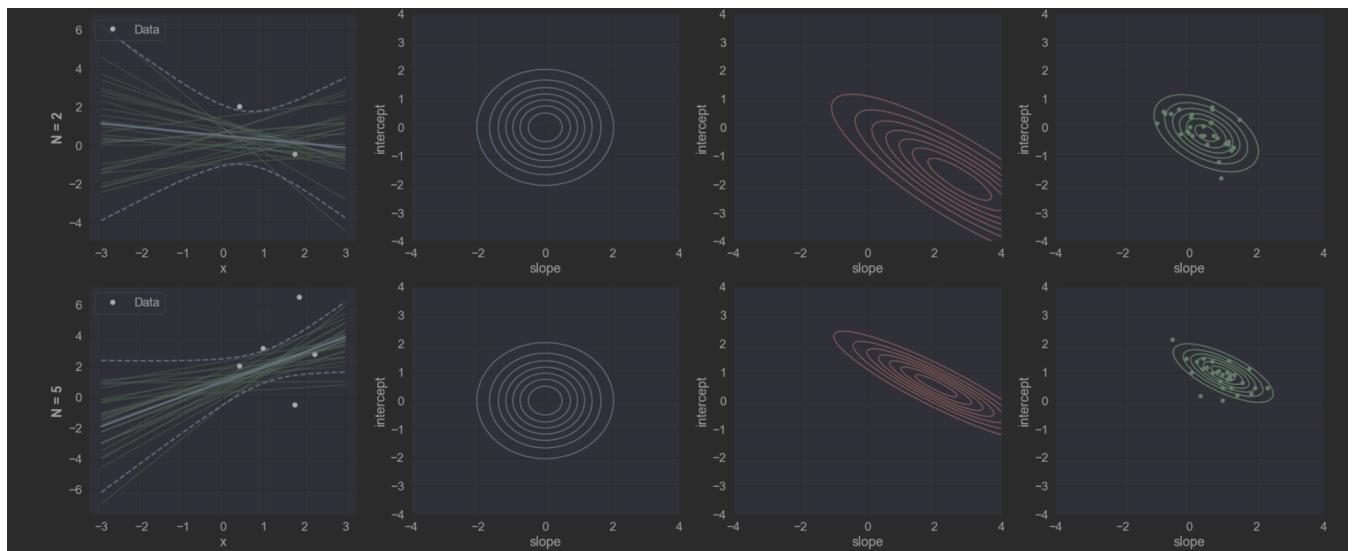
$$\hat{\mathbf{w}}_{\text{MAP}} = \arg \max_{\mathbf{w}} p(\mathbf{w} | \mathbf{t})$$

- **Analytisk løsning til the posterior.** Normalfordelinger er meget belejlige at regne med så vores eksakte posterier kan bestemmes analytisk. Hvordan og hvorledes vil jeg ikke snakke om.

$$\mathbf{m} = \beta \mathbf{S} \Phi^T \mathbf{t}$$

$$p(\mathbf{w} | \mathbf{t}) = \mathcal{N}(\mathbf{w} | \mathbf{m}, \mathbf{S}) \quad \mathbf{S} = (\alpha \mathbf{I} + \beta \Phi^T \Phi)^{-1}$$

- Alle parametrene og intuitioner forklares nok bedst med et plot:

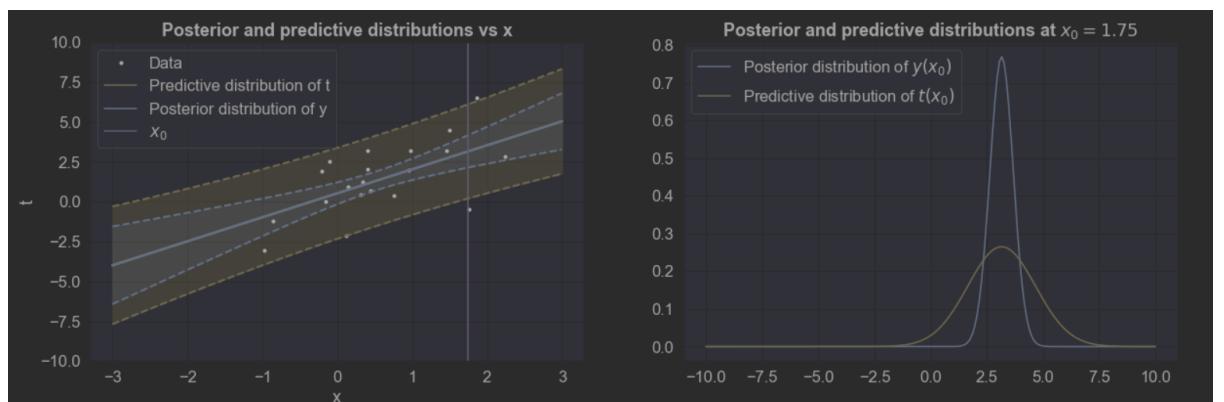


- store alpha/weight precision vil gøre vores prior mere kompakt og omvendt.
- store beta/data precision vil gøre vores likelihood mere kompakt og omvendt.
- 1: Når vi får mere data bliver vi mere sikre på vores parametre. de tynde streger er eksempler på vægte samplet fra vores posterior (svarer til en grøn prik i vores posterior), og de stiplede linjer er 95% konfidensintervaller.
- Processen ovenpå kunne være beregnet et datapunkt af gangen hvor man efter hver iteration brugte posterioren fra den foregående iteration som prior i den nuværende iteration. Dette er ikke tilfældet her dog.

- Ønsker vi at lave predictions for et nyt punkt x^* , svarer det til at middle/over alle parametre vægtet med posteriorene. Dette gøres ved først at beregne likelyhooden af det nye punkt, og derefter tage den forventede værdig med hensyn til posteriorene hvilket udtrykkes ved følgende integral.

$$p(t^* | \mathbf{t}) = \mathbb{E}_{p(\mathbf{w} | \mathbf{t})} [p(t^* | \mathbf{w})] = \int p(t^* | x^*, \mathbf{w}) p(\mathbf{w} | \mathbf{t}) d\mathbf{w}$$

- I dette tilfælde kan dette integral løses eksakt men det er ofte ikke tilfældet og man må i stedet løse det ved at sampling.



- Det her plot illustrerer meget godt forskellen i det 2. Usikkerheden i vores posterior er vores usikkerhed i vores parametre (epistemic/reducerbare) mens den ekstra usikkerhed vi ser i spredning i posterior predictive distribution er usikkerheden fra dateen (aleatoric/ ikke reducerbar). Den ekstra usikkerhed er dikteret af beta.
- Vi har 2 hyper parametre som dikterer vores usikkerhed i vores parametre og data: Hvad vi sætter dem til har en stor indflydelse på hvad hvordan vores endelige model ser ud. Kan vi optimere over dem. Ideelt set skulle vi inkludere dem direkte i vores model og inkludere en prior over dem men det ville resulterer i at vi skulle senere skulle integrere dem ud når vi skal lave posterior predictive hvilket ikke kan lade sig gøre analytisk.

$$p(t|\mathbf{t}) = \iiint p(t|\mathbf{w}, \beta) p(\mathbf{w}|\mathbf{t}, \alpha, \beta) p(\alpha, \beta|\mathbf{t}) d\mathbf{w} d\alpha d\beta$$

- The evidence approximation:** Hvis vi antager at vores posterior er sharply peaked omkring de optimale værdier af alpha og beta så kan posterior

predictive fordelingen fås ved blot at integrere over vægte w mens alpha og beta fastholdes.

$$p(t|\mathbf{t}) \simeq p(t|\mathbf{t}, \hat{\alpha}, \hat{\beta}) = \int p(t|\mathbf{w}, \hat{\beta})p(\mathbf{w}|\mathbf{t}, \hat{\alpha}, \hat{\beta}) d\mathbf{w}$$

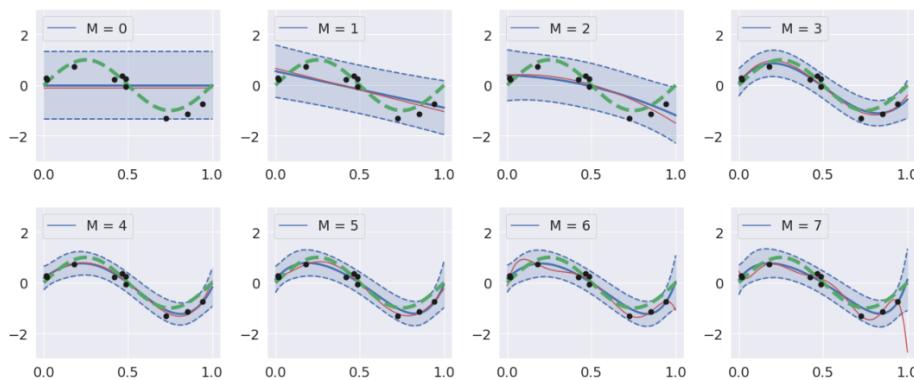
- **Vi kan bestemme de optimale værdier af alpha og beta** ved at tildele dem en flad prior og så optimere den marginale likelihood, hvor den marginale likelihood er givet ved:

$$p(\mathbf{t}|\alpha, \beta) = \int p(\mathbf{t}|\mathbf{w}, \beta)p(\mathbf{w}|\alpha) d\mathbf{w}$$

$$\hat{\alpha}, \hat{\beta} = \arg \max_{\alpha, \beta} \log p(\mathbf{t}|\alpha, \beta)$$

(i praksis Først fastholder vi alpha og beta og beregner vægtene w analytisk og så fastholder vi vægtene og optimere over alpga og beta. til sidst beregner vi vægtene analytisk igen, men kunne principielt køre flere iterationer)

- Ved at optimere over alpha og beta laver vi principielt automatisk model regularisering da alpha over beta svarer til lambda parameter i ridge regression. Vi kunne eksempelvis beregne den marginale likelihood for en række forskellige sammensætninger af basisfunktioner og den med den maksimale ville være den med det bedste trade off mellem kompleksitet og fit.



Week 3

1. Generative and discriminative classification
2. Logistic regression
3. Laplace approximations

- **Givet et dataset D med parvise forklarende variabel x og tilhørende labels t** ønsker vi at lære en mapping fra x til t. Lad os sige at t er binær enten 0 eller 1.
- **Nu ønsker vi at bruge vores mapping til at prædiktere et nyt datapunkt vi endnu ikke har set.**

$$p(t^* = 1 | \mathcal{D}, \mathbf{x}^*)$$

- Her er der 2 approaches til probabilistisk klassifikation:

Generative metoder

Diskriminative metoder

Generative metoder: Her modellerer vi join fordelingen af targets og attributes og benytter bays:

$$p(t_n = 1 | \mathbf{x}_n) = \frac{p(\mathbf{x}_n | t_n = 1)p(t_n = 1)}{p(\mathbf{x}_n)} \quad p(t_n = 0 | \mathbf{x}_n) = \frac{\pi_0 p(\mathbf{x}_n | t_n = 0)}{\pi_0 p(\mathbf{x}_n | t_n = 0) + \pi_1 p(\mathbf{x}_n | t_n = 1)}$$

- classconditionals: givet t=1 så følger x en fordeling
- prioclass probabilities=sandsynligheden for at t tilhører enten 0 eller 1
- den marginale data fordeling: som er en mixture af de 2 classconditionals.

Ud fra class conditionals kan vi modellerer den funktionelle form a(x)

$$a(\mathbf{x}_n) = \ln \frac{\pi_1 p(\mathbf{x}_n | t_n = 1)}{\pi_0 p(\mathbf{x}_n | t_n = 0)} = \ln \frac{p(t_n = 1 | \mathbf{x}_n)}{p(t_n = 0 | \mathbf{x}_n)}$$

$$p(t_n = 1 | \mathbf{x}_n) = \frac{1}{1 + \exp(-a)} = \sigma(a(\mathbf{x}_n))$$

(sigma er sigmoid og defineret som et ovenfor. kan generaliseres til multiclass med softmax)

Vi tager sigmoid for at "klemme" vores sandsynlighed mellem 0 og 1 så det reelt er en sandsynlighed.

- **Kaldes generativ fordi vi modellerer p(x),** således vi rent faktisk kan sample nye datapunkter hvis vi ønsker og resonerer om dataen.
Det kan dog være svært gøre sig korrekte/gode antagelser om classconditionals. Og hvis antagelserne er meget inkorrekt bliver det endelige resultat selvfølgelig påvirket derefter.

eks:

- I Diskriminativ klassifikation gør vi også ingen antagelser om the class conditionals:** Vi modellerer i stedet den funktionelle form af $a(x)$ direkte som en sigmoid af en linearkombination af vores inputs x . Ikke linearitet kan opnås afhængigt af hvordan designmatricen konstrueres. Den funktionelle form kunne også modelleres med et neutralt netværk for eks eller en anden model:

$$p(t_n = 1 | \mathbf{x}_n, \mathbf{w}) = \frac{1}{1 + \exp(-a)} = \sigma(\phi(\mathbf{x}_n)^T \mathbf{w})$$

Vi kan nu estimere w ved bayesian inference eller MLE. Dette er også kaldet logistic regression og har følgende likelihood.

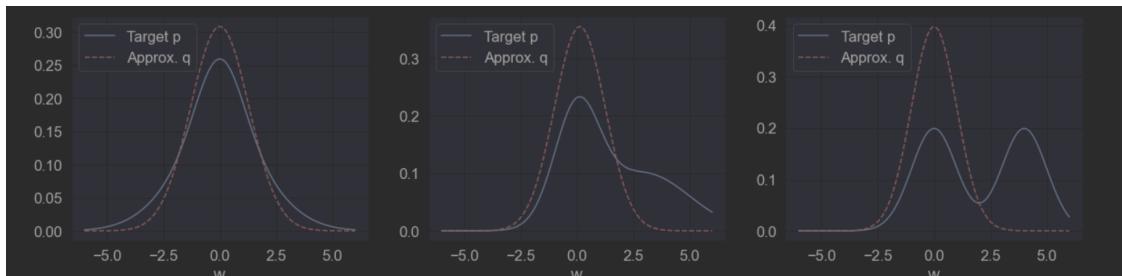
$$p(\mathbf{t} | \mathbf{w}, \mathbf{X}) = \prod_{n=1}^N \sigma(\phi(\mathbf{x}_n)^T \mathbf{w})^{t_n} (1 - \sigma(\phi(\mathbf{x}_n)^T \mathbf{w}))^{1-t_n}$$

- Det diskriminative approach er bedre når vores antagelser om class conditionals er dårlige.**
- Det er et mere simpelt approach har og har færre parametre:** derfor er det mere fleksibelt og nemmere at kalibrere
- Vi kan ikke rigtig resonerer om vores data eller generere data.**
- Det viser sig at vi ikke kan analytisk beregne the posterior mean i vores logistic regression model eksakt.** Vi bliver derfor nødt til at approksimere den. Den ligner ret meget en gaussian så hvorfor approksimerer vi den ikke med en.

Laplace approximationen: approksimerer en hvilken som helst funktion med en gaussian. Det gør den ved:

$$p(\mathbf{w} | \mathbf{t}) \approx q(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{w}_{\text{MAP}}, \mathbf{A}^{-1})$$

Her meanen af vores approksimation givet med moden/MAP solution af vores posterior, mens at covariance matricen er givet ved den inverse hessian beregnet i mode/MAP



Når vores posterior ligner en gaussian får vi en god approksimation.

Når vores posterior er ret symmetrisk får vi en god approksimation, men den kan ikke rigtig tage højder for tails.

Hvis den er multimodal kommer vores MAPsolution til at afhænge initialiseringen af vores optimiser.

- **Hvad så med prædiktioner.** Sædvanligvis beregner vi likelihood i x^* vægter den med vores posterior og integrerer vægtene ud. Dette integral er analytisk intractable.
 - Monte carlo methods: Hvor vi blot sampler vægte fra vores posterior og herved kan konstruere en fordeling over outputs t^* , og eventuelt bestemme usikkerheder eller en prædiktion som blot vil være mean af den samlede fordeling over outputs.

$$p(t^* = 1 | \mathbf{t}, \mathbf{x}^*) = \int \sigma(y) \mathcal{N}(y | \mu, \sigma^2) dy \approx \frac{1}{S} \sum_{i=1}^S \sigma\left(y^{(i)}\right) \quad \text{for} \quad y^{(i)} \sim \mathcal{N}(y | \mu, \sigma^2)$$

$$\mu = \phi(\mathbf{x}^*)^T \mathbf{m} \quad \sigma^2 = \phi(\mathbf{x}^*)^T \mathbf{S} \phi(\mathbf{x}^*)$$

- Probit approximation: hvor vi approksimerer sigmoid funktionen med en probit. Og herved kan løse integralet analytisk da:

$$\int \Phi(\lambda a) \mathcal{N}(a | \mu, \sigma^2) da = \Phi\left(\frac{\mu}{(\lambda^{-2} + \sigma^2)^{1/2}}\right) \quad \text{a=y}$$

- Sidste mulighed er numerisk integration. eks gauss-hermite.

Week 4

1. Covariance functions and the squared exponential kernel
2. Gaussian processes for regression

Givet et datasæt bestående af par forklarende variable x og afhængige y , har vi med **bayesian linear regression** bestemt en **fordeling over vægte w** der giver en **mapping** fra x over i y :

$$t_n = y_n + e_n = \phi(x_n)^T w + e_n \quad y_n = \phi(x_n)^T w$$

Her har fokuseret kun være at modererer joint fordelingen af dataen og vægtene. og så snart vi har bestemt vægtene har vi smidt dataen væk. Nu kunne vi istedet betragte funktionsværdierne $y(x)$ for alle x :

$$[y(\phi(x_1)) \quad y(\phi(x_2)) \quad \dots \quad y(\phi(x_N))]$$

og lade disse ingå i vores joint fordeling

$$p(t, y, w) = p(t|y)p(y|w)p(w)$$

Det her udtryk er det helt samme som, idet vi blot kan marginalisere over y hvorved den sædvanlige joint fordeling over parametre og data fremkommer.

Men hvad nu hvis vi istedet integrerer/marginaliserer over parametre/vægtene.

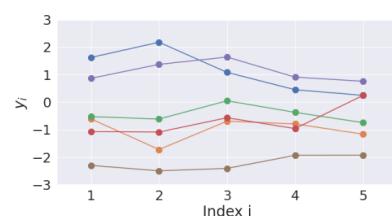
$$p(y) = \int p(y|w)p(w)dw = \int p(y|w)\mathcal{N}(w|0, \alpha^{-1}I)dw = \mathcal{N}\left(y|0, \alpha^{-1}\Phi\Phi^T\right)$$

herved kan vi konstruere en prior over lineære funktioner i stedet for vægte/parametre, der er parametreret ved ovenstående normal fordeling, der har 0-mean og en en sjovt størrelse af en covariance.

Men hvordan kan en multivariate normalfordeling repræsentere funktioner?

Hvis vi tænker på funktioner som en vektor med et (uendeligt) antal dimensioner.

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & 0.8^1 & 0.8^2 & 0.8^3 & 0.8^4 \\ 0.8^1 & 1 & 0.8^1 & 0.8^2 & 0.8^3 \\ 0.8^2 & 0.8^1 & 1 & 0.8^1 & 0.8^2 \\ 0.8^3 & 0.8^2 & 0.8^1 & 1 & 0.8^1 \\ 0.8^4 & 0.8^3 & 0.8^2 & 0.8^1 & 1 \end{bmatrix}$$



VI kan forbinde hvert tilstøde element/index i vektoren med en streg. Da vi vil se at givet kovariansmatricen dikterer at punkter der kommer efter hinanden også ligner hinanden. mens punkter der ligger langt fra hinanden kan varierer meget.

Hvad nu hvis vi ændrer covariance matrices/funktionen til noget andet?

Det må vi gerne. Det hedder kernelfunktioner og der er en række betingelser de skal opfylde. Herved kan vi modlere interessant ikke lineær funktioner.

Kriterierne for en kernel funktion er at de skal være symmetriske og. similariteten mellem a og b skal være den samme som mellem b og a. De skal være positive define idet vi skal kunne invertere den endelige covariance matrix. Vi kan konstruere kernel funktioner ud fra andre gyldige kernalfunktioner på alle mulige leder og kanter.

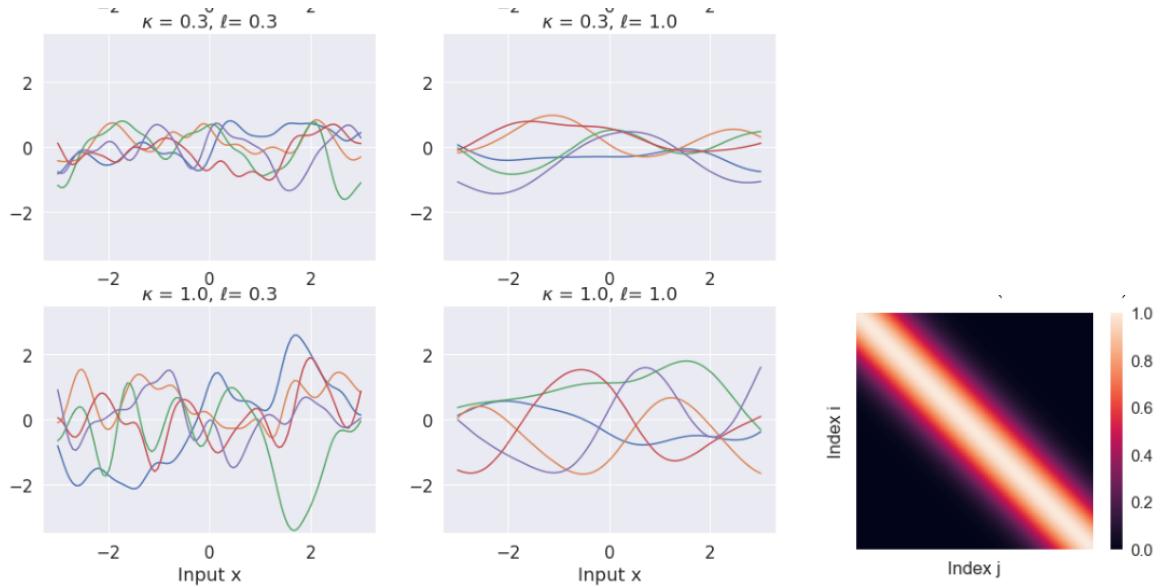
En væsentlig intuition bag GP er at hvis inputs ligner hinanden så gør deres outputs the også. Hvilket afpejles i at covariance mellem outputs er givet in terms of in puts.

$$x \approx x' \Rightarrow y(x) \approx y(x')$$

Et eksempel på en meget populær kernelfunktion af the **squared exponential kernel**

$$\text{cov}[y(x), y(x')] = k(x, x') = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right)$$

Den er parametriseret ved length scale l og magnitude sigma.



her dikterer lengthscale hvor hurtigt funktionen kan oscillerer/ ændre retning. en høj lengthscale medfører at x værdier langt fra hinanden stadig betragtes som kovariende: heatmappetdiagonalen vil være bredere, og de samplede funktioner er strukket mere ud/fladtrykte fordi de ikke må ændre sig særlig hurtigt. og omvendt en lille length scale medfører at diagonalen er meget smal i heat mappet og funktionværderne y kan ændrer sig meget hurtigt.

magnitude dikerer egentlig bare intensiteten på heatmappet eller skaleringen på y aksen. en stor magnitude vil gøre funktionstoppe/dalene meget høje/lave.

Hvis vi nu betragter vores GP som en prior fordeling over funktioner, så kunne vi jo eksempelvis konditionere på observeret data. herved kan vi lave ikke linear

regression: med andre ord gaussian process regression. Vi antager at vores data er gaussian distribution. Tilsvarende er vores prior, og herved kan man benytter nogle identiteter for konditionering på multivariate gaussiske fordelinger (som jeg ikke vil komme ind på til at) til at konstruere:

$$p(t^* | \mathbf{t}) = \mathcal{N} \left(t^* | \mu_{t^* | \mathbf{t}}, \sigma_{t^* | \mathbf{t}}^2 \right)$$

$$\mu_{t^* | \mathbf{t}} = \mathbf{k} (\mathbf{K} + \beta^{-1} \mathbf{I})^{-1} \mathbf{t}$$

$$\sigma_{t^* | \mathbf{t}}^2 = c - \mathbf{k} (\mathbf{K} + \beta^{-1} \mathbf{I})^{-1} \mathbf{k}^T$$

hvor

$$[\mathbf{K}]_{ij} = k(x_i, x_j)$$

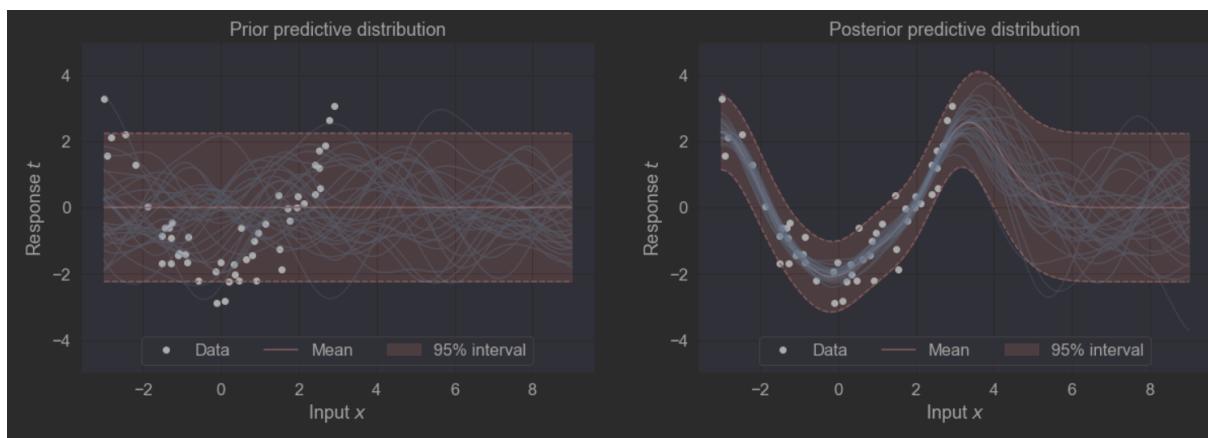
kovariansmatricen for vores træningsdata

$$[\mathbf{k}]_j = k(x_*, x_j)$$

kovariansen mellem de punkter vi ønsker og prædiktere og vores træningsdata

$$c = k(x_*, x_*) + \beta^{-1}$$

covariance af de punkter vi ønsker at prædiktere. dette udtryk indeholder også noise variance af vores data: inverse precision.



Her er et eksempel på en GP regression. Vi har 3 parametre. lengthscale, magnitude og noise variance.

det er meget tydeligt at **når vi bevæger os væk fra dataen bevæger** vores fittede model sig tilbage mod prioren.

Hvis vi forøger lengthscale: vil det tage længere tid før vores fit fandt tilbage til vores prior når vi bevæger os væk fra dataen. og vores fit ville blive mere smooth og wiggle mindre.

Hvis vi forøger magnitude bliver funktioner meget højere/lavere i vores prior, men hvis vi sænker dem bliver vores fit trukket mod nul/trykket helt sammen og kommer til at underfitte vores data. for meget bias og for lidt variance.

sidst men ikke mindst er der sigma/noise variance. Hvis den er høj stoler vi ikke på vores data og vi lader derfor vores posterior predictive distribution være meget bred for at være sikker på vi fanger alt støjen og ikke er overconfident.

Vi kan optimere vores 3 parametre ved hjælp af evidence approximation.

$$\hat{\theta} = \arg \max_{\theta} \ln p(\mathbf{t}|\theta).$$

hvor theta er vores 3 parametre. i praksis gøres det ved at beregne gradienterne med hensyn til theta og laver numerisk optimering.

GP regression er ikke parametrisk idet den skalerer i kompleksitet med mængde af data. I praksis indeholder modellen alt vores data da K/kovariansmatricen har dimension givet ved NxN hvor N er antallet af trænings observationer. Herved skalerer metoden dårligt med store datasæt især fordi vi skal invertere matrice der er NxN, hvilket har O(N^3)

Week 5

1. Gaussian processes for classification
2. Non-gaussian likelihoods

Dataset D: vi har et dataset D med feature vektors x of labels t . Ved sædvanlig bayesian regression ønsker vi at modellere en fordeling afvægte givet vores data ved at benytte bayes theorem på nedenstående.

$$p(t, w) = p(t|w)p(w)$$

Så ville vi finde en mapping givet vores vægte som vil mappe vores feature vektorer ind i en latent funktion:

$$y_n = \phi(x_n)^T w$$

Hvad nu hvis vi beregnede alle vores y for alle vores x og lod y indgå i vores joint fordeling. Det er fuldstændig gyldigt, vi kan bare marginalisere den ud og vi ville igen være ved udgangspunktet. men vi kunne også marginalisere over vores vægte og vi ville omgås vægtene

$$p(t, y) = \int p(t|y)p(y|w)p(w)dw = p(t|y)p(y)$$

Herved kan vi konstruere en prior der mapper ikke ind i parameter rum men ind i **rummet over lineære funktioner** og fordelingen over y ville være:

$$\mathcal{N}(y|\mathbf{0}, \alpha^{-1}\Phi\Phi^T)$$

Hvis vi ændrer covariance funktionen til eksempelvis squared exponential kernel kan vi lige pludselig modlere ikke lineære forhold. og det er jo smadder smart da vi så kan opnå ikke lineære decision boundaries men meget få parametre. eks givet squared exponential kernel i GP til regression har vi principielt kun 3 parametre. Noise variance fra vores gaussian likelihood og så magnitude og length scale for squared exponential kernel. Men en GP er en ikke parametrisk og model og stiger i kompleksitet med mængden af data.

Men vi kunne også lave klassifikation med vores GP prior eks:

$$\begin{aligned} t_n &\sim \text{Ber}(\sigma(y(x_n))) \\ y(x_n) &= \phi(x_n)^T w & t_n &\sim \text{Ber}(\sigma(y(x_n))) \\ w &\sim \mathcal{N}(\mathbf{0}, \alpha^{-1}I) & y &\sim \mathcal{GP}(0, k(x, x')) \end{aligned}$$

Vi har omgået vægtene så at sige. og benytter en gp til at mappe over i vores latente funktion y . Herefter benytter vi en sigmoid til at klemme vores output y til at ligge mellem en sandsynlighed på 0 og 1, og lader den sandsynlighed være parameteren i

en bernoullis fordeling for binær klassifikation. Herved bliver vores likelihood en bernoullis likelihood

$$p(\mathbf{t}|\mathbf{y}) = \prod_{n=1}^N \sigma(y_n)^{t_n} (1 - \sigma(y_n))^{1-t_n}$$

I regressions tilfældet var alle vores fordelinger gaussiske og vi kunne udlede vores posterior analytisk, dette er ikke længere tilfældet og vi bliver nødt til at approksimere den. Det kunne vi jo passende gøre med en laplace approximation.

Laplace approksimerer en arbitrer fordeling med en normalfordeling hvor meanen i normalfordelingen er en mode/MAP solution til den fordeling man ønsker at approksimere. Her bestemmes MAP ved numeriske metoder for eks gradient descent. covariance af vores Laplace normalfordeling fremkommer som den inverse hessian beregnet i MODE af vores posterior.

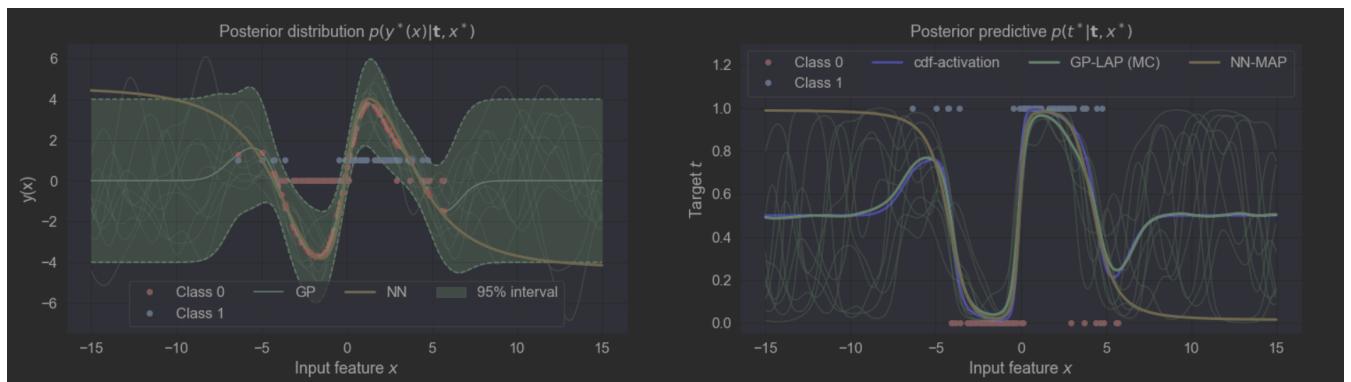
Posterior predictive distribution: kan ikke bestemmes analytisk vi er nødt til at enten at benyttes os af monte carlo samples, hvor vi sampler fra vores posterior latent funktion y^* . tager sigmoid til samplet og bestemmer sample average.

$$p(t^* = 1 | \mathbf{t}, \mathbf{x}^*) \approx \frac{1}{S} \sum_{i=1}^S \sigma\left(y^{(i)}\right) \quad \text{for} \quad y^{(i)} \sim \mathcal{N}(y | \mu_{y^*}, \sigma_{y^*}^2)$$

præcisionen afhænger hvor mange samples vi trækker. jo flere samples jo mere præcisest men og des mere computation. dog kan MC samples ofte bestemmes selv i tilfælde hvor vi ikke har analytiske probit approximationer.

Alternativt er der probit approximation Hvor vi approksimerer vores sigmoid med en CDF af en normalfordeling, hvilket giver en analytisk løsning til den forventede værdi.

Vi kunne også lade vores latente funktion være givet ved outputtet fra et NN. Her er et eks på en sammenligning mellem NN og GP for latent funktion y



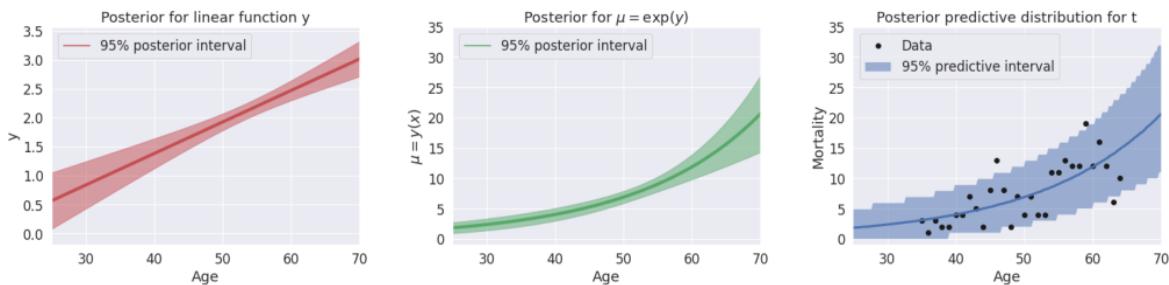
Man skal bemærke at det venstre plot bare er den latente funktion der ikke er blevet kørt gennem en sigmoid og derfor trykket ned mellem 0 og 1.

Til højre ser vi at både MC predictive og probit minder rigtig meget om hinanden og er meget enige. og det gør NN sådan set også. men og det er et stort men. Nå vores GP med squared exponential kernal bevæget sig ca omkring 3 length scales væk fra “the support of the data” da falder den tilbage til prioren, hvilket svarer til en sandsynlighed på 0.5 altså at den slet ikke tør/kan udtale sig. Her bliver NN ved den sidste klasse den har observeret og generaliserer alt herfra som den klasse.

Men hvorfor stoppe her:

Vi har set hvordan man principielt laver lineær regression om til classification ved at sigmoid funktion til at klemme outputte ned mellem 0 og 1 og benytte en bernoullis likelihood. Vi stødte da ind i at vores posterior ikke kunne regnes analytisk men det løste vi med en laplace approximation, og vores posterior predictive kunne vi bestemme ud fra monte carlo samples.

Hvorfor modellerer vi ikke bare alt sådan her: Generalized linear models, poisson regression.



I Stedet for at bruge sigmoid some en link function kunne vi bruge en exponentialfunktion siden vi ved at poisson er en fordeling over antal og at den ikke kan være negativ. Ligeledes ville vi skifte vores bernoullis likelihood ud med en poisson likelihood.

(link funktion, en funktion der relaterer mean af lineær model med mean af response variable t)

Og som vi allerede har set **behøves det ikke at være en lineær model** den kunne vi passende udskifte med en GP eller et NN eller noget helt 3.

Week 6

1. Multi-class classification
2. Decision theory
3. Calibration

Multi-class classification er en naturlig udvidelse af binær:

$$\begin{aligned}t_n &\sim \text{Ber}(\sigma(y(\mathbf{x}_n))) \\y(\mathbf{x}_n) &= \phi(\mathbf{x}_n)^T \mathbf{w} \\\mathbf{w} &\sim \mathcal{N}(\mathbf{0}, \alpha^{-1} \mathbf{I})\end{aligned}$$

Givet at vi har K klasser, skal vi blot indføre lige så mange sæt af vægte \mathbf{w}_k som vi har klasse, K hvorved vi også tilsvarende får K forskellige latente funktionsoutput y_k . Nu sender vi ikke vores output y_k gennem en sigmoid men i stedet en softmax.

$$p(y_k) = \frac{\exp(y_k)}{\sum_{i=1}^K \exp(y_i)}$$

Den hedder max fordi den største y_k stadig vil være den største efter og soft fordi de værdier der ikke er de sidste stadig får lov at bibeholde lidt information. Vigtigst af alt kommer alle vores $p(y_k)$ til at summere til en 1 og bliver altså en gyldig sandsynlighed.

Udskifte bernoullis fordeling med en kategorisk parametriseret ved vores softmax output:

$$t_n | \mathbf{y}_n \sim \text{Categorical}[\text{softmax}(\mathbf{y}_n)]$$

Vores posterior bliver intractable og vi approximerer den med en laplace approximation og tilsvarende bruger vi montecarlo samples til at bestemme posterior predictive probabilities.

Predictions: Som udgangspunkt vil vi blot predicte det label der har den største softmax værdi.

Hvad nu hvis vores softmax outputs er 0.33, 0.33 og 0.34? da vil predicte klasse med den højeste sandsynlighed altså på den sidste klasse men vores model har så godt som ikke ide.

Vi indfører confidence:

$$\mathcal{C} = \max_k p(t^* = k | \mathbf{t}, \mathbf{x}^*) \quad (\text{range: } [\frac{1}{K}, 1])$$

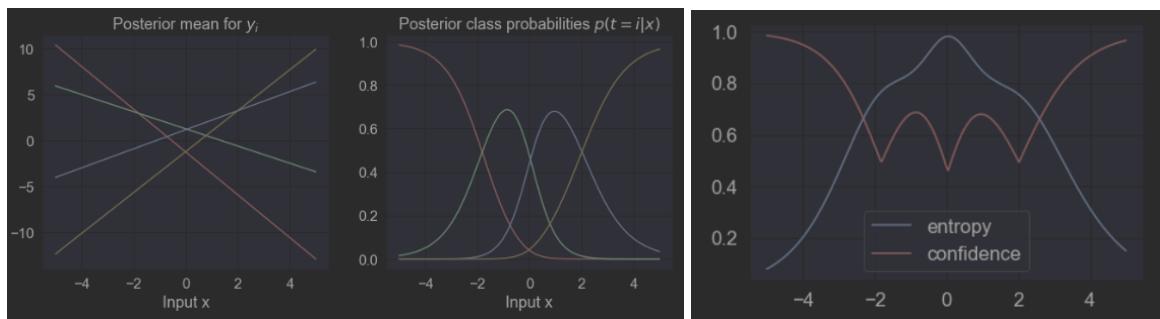
Confidence er blot svarende til den højeste af vores klasse sandsynligheder for et givent input. i det foregående tilfælde vil vores confidence være 0.34, og den skal lidt fortolkes i sammenhold med antallet af klasser og derfor er 0.34 rigtig lavt.

Vi kunne eksempelvis rejte et samples hvis confidence er for lav. Hvis vi ville være sikre på at vores model ikke kommer til at udtales sig om noget den ikke ved kan man eksempelvis thresholde confidence på 0.6 så alle inputs der giver en confidence på under 0.6 forkaster vi og vil ikke predice på.

Entropy er defineret således:

$$\mathcal{H} = - \sum_{k=1}^K \pi_k \log \pi_k \quad (\text{range: } [0, \log K])$$

Entropy er et mål for uorden/usikkerhed. des større den er des mindre er vi sikre. I den forstand er den lidt omvendt af confidence. den numeriske værdi er entropy kan være lidt svær at tolke på, og kan mest af alt sammenholdes relativt med entropy fra et andet input. entropien er maksimal for en uniform fordeling.



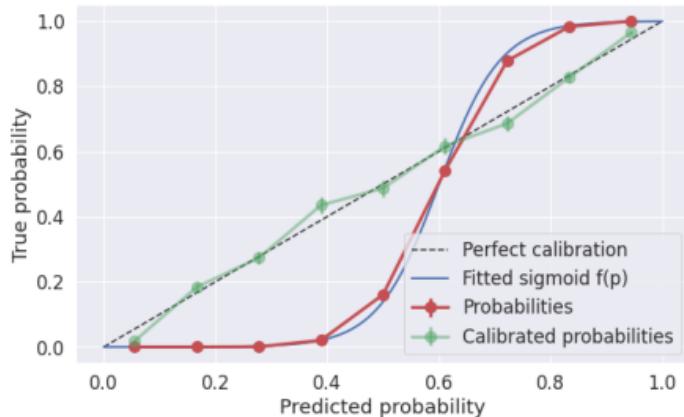
Utility funktioner. Selvom vi har fået et posterior output af vores model kunne man tænker sig at der var nogle konsekvenser associeret med den handling vores output fører til. Omfanget af de her konsekvenser kan formuleres som en utility funktion/matrix:

$\mathcal{U}(t, \hat{t}(x))$	$\hat{t} = 0(\text{not cancer})$	$\hat{t} = 1(\text{cancer})$
$t = 0(\text{not cancer})$	1	-10
$t = 1(\text{cancer})$	-100	1

Eksempelvis kunne vi være meget mange for at predicte at en person ikke har cancer hvis vedkommende faktisk har. Derfor kan vi associerer en meget stor negativ vægt med den prediction.

Vi laver altså en slags vægtet midling over vores posterior prediction, hvorved vi medregner at nogen udfald har større konsekvenser og at vi derfor skal være helt sikre i de tilfælde. Dernæst vælger vi så den handling der maksimerer vores expected utility.

Man ser tit at ens model enten er over eller under confident. Det viser sig at man kan kalibrere for det.



Her ser vi eks at den først er under confident og så overconfident: proceduren til at kalibrerer er som følger:

- Du fitter din model og predictor dit test set.
- Du rangerer din confidence fra mindst til størst. og deler dem op i eks 10 lige store bins.
- For hver bin beregner du den egentlige test acc. for eks burde den sidste bin som svarer til en model confidence på mellem 90-100% burde test acc jo være 95%.
- Du fitter en sigmoid (med MLE) med parametrene A og B, der giver predictive prob fra din model $f(x)$ bestemmer den egentlige sandsynlighed.

$$P(t = 1|x) = \frac{1}{1 + \exp(A + Bf(x))}$$

- Den kan ligesom tage højde for når din model er over eller under confident.

Det kræver et stor test set, især for multiclass klassifikation. Lad os sige du har 10 klasse og vil have 20 test observationer med hvert label i hver bin af dine 10 bins: allerede der er man oppe på $10 \times 20 \times 10 = 2000$ test samples. og så er der ikke engang nogen garanti for at de fordeler sig ligeligt i de forskellige bins.

(Det kan være du aldrig predikter klasse 5 med mere end 80% confidence, da vil de sidste 2 bins slet ikke være repræsenteret)

Week 7

1. Markov Chain Monte Carlo Methods
2. Metropolis-Hastings algorithm

Med bays regel opstiller vi vores posterior distribution over vores vægte givet vores model og vores data:

$$p(\mathbf{w}|\mathbf{t}) = \frac{p(\mathbf{t}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{t})}$$

Som oftes vil vi gerne lave predictions givet vores model or noget nyt data.

Bestemme likelihood af nyt punkt vægte den med vores posterior og marginalisere/integrere over alle vores vægte.

$$p(t_*|\mathbf{t}, \mathbf{x}^*) = \int p(t_*|\mathbf{w}, \mathbf{x}^*)p(\mathbf{w}|\mathbf{t})d\mathbf{w}$$

Begge indeholder integraler der er intractable for næsten alle interessante modeller. Man kunne bruge conjugate priors men de fungerer kun for meget simple modeller og man vælger dem typisk af matematisk ulejlighed mest af alt.

Vi er nødt til at approksimere den. eks laplace. Bruger en normalfordeling hvor mean af den er map/mode af din posterior. moden fundet ved numeriske optimering. covariance matricen bestemmes som den inverse hessian beregnet i moden.

Variational methods: Her approksimerer vi vores posterior med eks en mean field af gaussians. som vi fitter ved at minimere KL divergensen (eller et andet afstandsmål) mellem vores mean field og vores posterior. I praksis gøres det ved at maksimere vores evidence lower bound ELBO.

Til sidst har vi markov chain monte carlo methods:

Det er metoder der approksimerer/estimerer vores posterior eller en hvilken som helst sandsynlighedsfordeling ud fra samples. Med den eneste forudsætning af vi kan beregne densiteten af vores posterior et i get punkt.

Helt generelt er det ofte svært at få i.i.d. samples fra en fordeling men dette muliggør MCMC.

En første ordens markov chain siger at “the future does not depend on the past only the present”: så hvis vi har en markov chain kan den værdi/variable et skridt længere fremme forklares fuldstændig ud fra den variable der er i det nuværende tidsskrift.

$$p(z^{m+1}|z^{(1)}, z^{(2)}, \dots, z^{(m)}) = p(z^{m+1}|z^{(m)})$$

Vi vil bruge vores markov chain til at designe en random walk:

Hertil skal vi bruge en transition kernel.

$$T_m(z^{(m)}, z^{(m+1)}) \equiv p(z^{(m+1)} | z^{(m)}) \quad x_{k+1} = x_k + e_k, \quad e_k \sim \mathcal{N}(0, \sigma^2)$$

Den kunne meget vel være en normalfordeling således at det næste skridt $k+1$ er en normalfordeling men hvor mean er svarende til værdien i det nuværende tidsstep k .

Nu har vi alle ingredienser til vores Metropolis hasting algorithm:

Initialiser vores θ_0 til en eller anden værdi og så for $1..K$ steps gør:

sample fra vores proposal distribution for eksempel en normalfordeling:

$$q(\theta^* | \theta^{k-1}) = \mathcal{N}(\theta^* | \theta^{k-1}, \tau I)$$

beregn acceptance probability:

$$A_k = \min\left(1, \frac{p(\theta^k) q(\theta^{k-1} | \theta^k)}{p(\theta^{k-1}) q(\theta^* | \theta^{k-1})}\right)$$

Accept hvis A er større end u , hvor u er trukket fra en uniform 0-1 fordeling.

Rejekt hvis A e mindre og behold det gamle sample θ_{k-1}

$$\theta^k = \begin{cases} \theta^* & \text{if } u_k < A_k \\ \theta^{k-1} & \text{otherwise} \end{cases}$$

Hele ideen er altså at vi har designet vores random walk således at vi befinder os mest muligt i områder af vores posterior med høj densitet. For hvis vores proposal sample har højere densitet end vores previous så beholder vi det altid da acceptance prob bliver over 1. Og selv hvis vi tager et skridt i den forkerte retning er der stadig en vis sandsynlighed for at vi bibeholder vores sample.

Metropolis versus metropolis hastings. For metropolis er the proposal distribution symmetrisk således forholdet mellem den i acceptance prob er 1 og det bare forsvinder. Her tillade metropolis hastings as vi bruger ikke symmetriske fordelinger.

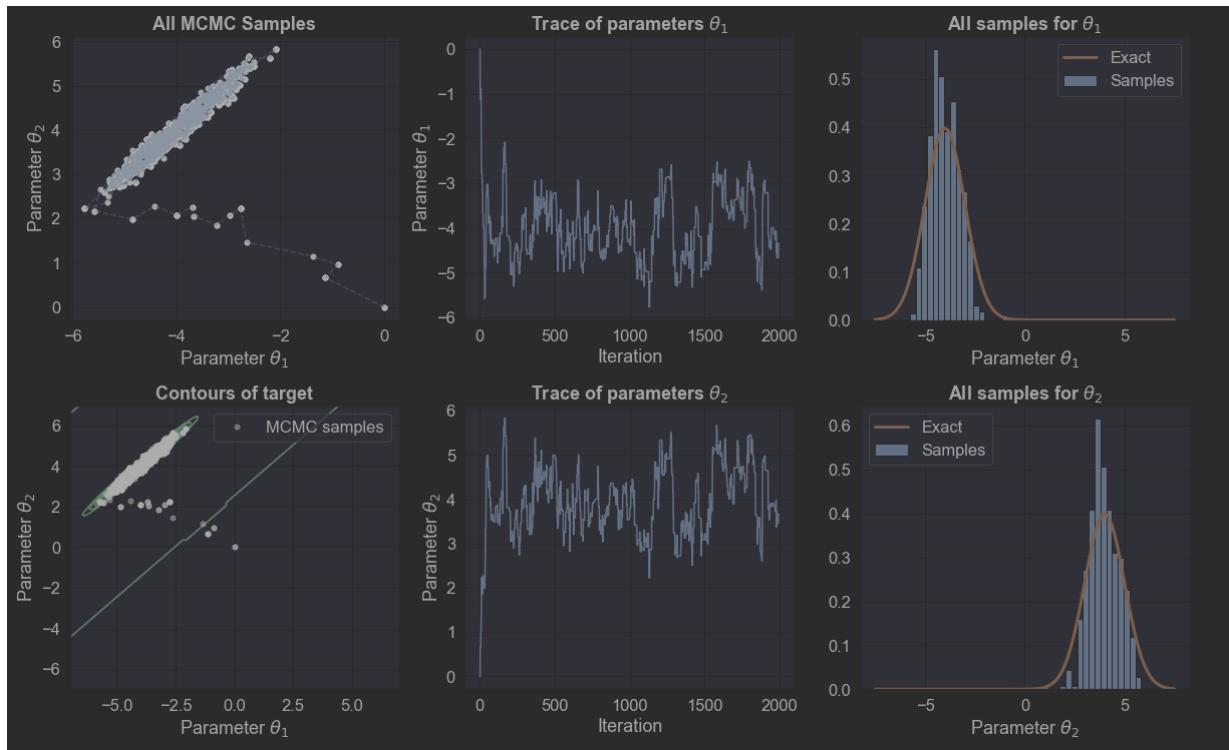
Dog skal vores proposal fordeling altid kunne tillade at vi sampler negative værdier. Ellers kan vi kun gå i en retning.

Men vi kender jo ikke tætheds fordelingen over vores posterior? pga marginal fordelingen over vores data. Typisk har vi kun vores prior og vores likelihood og ikke vores marginal. Heldigvis ændrer marginal fordelingen sig ikke og svarer bare til en normaliserings konstant. fordi den indgår i både tæller og nævne forsvinder den. Vi behøves altså kun at kunne evaluere vores density op til en konstant.

Step size.

Vores MCMC har en hyper parameter og det er tau i vores proposal distribution. tau er variance i vores normalfordeling go kommer til at svare til stepsize. jo større tau jo længere skridt tager vi. Hvis tau er for stor kommer vi dels til at rejecte mange af vores samples og vi kommer til at overshoot vores target density.

Hvis tau er for lille kommer vi til at accepte basically alle vores samples men vi kommer også til at tage meget små skridt. Vi kommer til at have problemer med at med at komme fra theta0 til vores egentlige fordeling. Og dels kan vi få problemer med at udforske hele vores sandsynlighedstæthed.



MCMC har også andre udfordringer.

- Hvornår er vores random walk convergeret?
- Alle vores samples er meget korrelated med previous and consecutive samples?
- Hvis vores target distribution har internt meget korreleret parametre eks en multivariate normalfordeling med høj covariance, så kommer vi til at projekte mange af vores samples

Week 8

1. MCMC and Convergence diagnostics
2. Gibbs sampling
3. Change point detection

For stort set alle bare lidt interessante modeller og derfor lettere komplekse modeller er vores enten vores posterior eller predictive posterior intractable. vi kan ikke regne dem exact og vi er nødt til at approksimere dem eller “sample” dem.

Vi er nødt til at approksimere den. eks laplace. Bruger en normalfordeling hvor mean af den er map/mode af din posterior. moden fundet ved numeriks optimering. covariance matricen bestemmes som den inverse hessian beregnet i moden.

Variational methods: Her approksimerer vi vores posterior med eks en mean field af gaussians. som vi fitter ved at minimere KL divergensen (eller et andet afstandsmål) mellem vores mean field og vores posterior. I praksis gøres det ved at maksimere vores evidence lower bound ELBO.

MCMC methods:

MCMC methods gør det muligt at indsamle samples fra vores posterior, selv når vi ikke kan sample fra den men kun evaluere dens densitet for et givet input. In short virker det ved at vi bruger markov property (future depends on the present and not the past) til at gå en random walk som afdækker hele vores posterior sandsynlighedsfordeling. Herved opnår vi en masse samples hvorved vi kan estimere eks average og standardafvigelse af vores posterior parameters.

MCMC har en række fordele og ulemper.

- har meget stærke teoretiske garantier:
- hvis vi sampler længe nok vil vores markov chain converge til den rigtige fordeling.
- den er nem at implementerer og nem at forstå
- det er nemt at afprøve forskellige modeller

Ulemper

- det er garanteret at vores markov chain konvergere men det kan være meget svært at afgøre hvornår og måske vil det tage uendelig lang tid.
- vores acceptance ratio er måske meget lav således vi rejecter alle vores samples.
- Den er langsom for store dataset
- Vores proposal kræver måske tuning, såsom variance parameteren i en normalfordeling kommer til at svare til step size som enten kan være for stor eller for lille

Gibs sampling er et forsøg på at komme nogle af ulempene til livs.

Den simple forskel fra metropolis hastings til gibbs sampling er at man ikke bruger en proposal distribution men at man sample en ny værdi for en af sine parametre ved blot at conditionere på de restrende: så givet at du har 3 parametre sampler du i hver runde den første mens du fastholder de resterende 2 på deres nuværende værdi. så sampler du den næste mens du fastholder de foregående 2 osv.

Når du så samplet en ny værdi for hver af dine parametre vil det være dit næste skridt i din random walk. Det kan vises matematisk at det næste skridt i din random walk altid vil have acceptance probability 1, og at vi derfor aldrig rejecter det.

Gibbs sampling er et særligt tilfælde af metropolis hvor man altid beholder sit proposal sample

Men det betyder også at man skal udlede hver af de konditionerede sandsynligheder for din model. og hvis du ændrer lidt i din model skal du udlede dem på ny. Så det er et trade off mellem fleksibilitet og effektivitet.

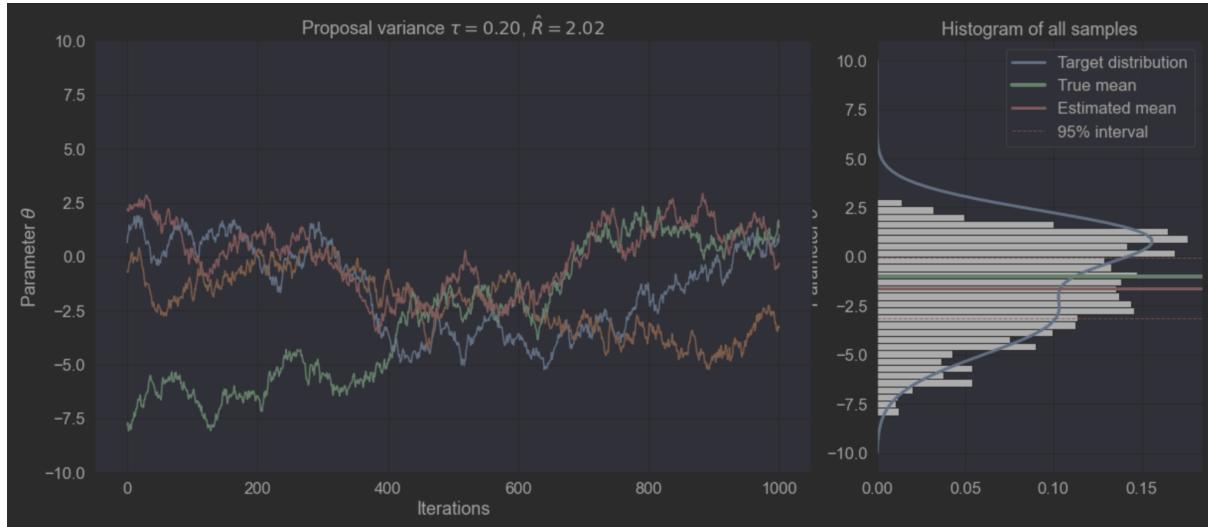
(den generelle metode til at udlede de konditionerede sandsynligheder er som følge:
1) skriv log join densitet op. 2) find alle termer der afhænger af dine parametre, smid resten væk. 3) identificer sandsynlighedsfordelingen af parameter w_i konditioneret på resten af dine parametre baseret på den funktionelle form af w_i)

Convergence diagnostic

Det er garanteret at din MCMC algoritme konvergerer til din taget fordeling men der er ikke nogen stringent teori der siger noget om hvor mange iterationer det vil tage: men der findes heuristikker der kan give et ret god bud på om kæderne er konvergeret:

Intuitionen er at vi kører vores mcmc en række gange men fra forskellige initial parameter settings og så sammenligner/kvantificerer om de samplede fordelinger er "ens". Det kan formuleres som R statistikken:

$$\hat{R}^2 = \frac{N}{N - 1} + \frac{1}{N} \frac{B}{W}$$



hvor B er between-chain variance og W er within-chain variance. Ud fra formlen kan det tydeligt ses at hvis variansen mellem kæderne er større end internt i kæderne bliver brøken større end 1 og vores R bliver større end en.

Hvis B er lig W bliver R lig 1 og de er altså fuldstændig konvergeret men i realiteten siger man at oftes at kæderne er konvergeret hvis R er mindre end 1.1

Siden vores samples tilhører en markov chain er det meget autokorreleret. altså et sample vil være gradvis mere korreleret med et andet sample i kæden jo tættere de er på hinanden. Derfor er hvert af vores samples individuelt ikke så informative som hvis de havde været ukorreleret og det svarer altså det vi effektivt har færre samples end der egentlig er i vores kæde.

Derfor har man dels indført udtrykket effektive sample size og endvidere markov chain standard error.

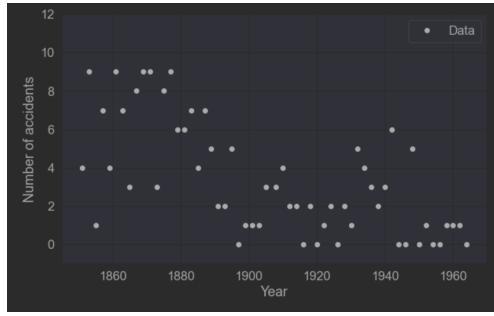
$$S_{\text{eff}} = \frac{s}{\sum_{t=-\infty}^{\infty} \rho_t} = \frac{s}{1 + 2 \sum_{t=1}^{\infty} \rho_t} \quad \rho_t = \frac{1}{\sigma^2} \int (\theta^i - \mu)(\theta^{i+t} - \mu) p(\theta) d\theta$$

fejlen på montecarlo estimator er skalerer med $\sqrt{f} = \frac{1}{s} \sqrt{V[f(\theta)]}$ derfor skalerer standart fejlen med $1/\sqrt{s}$

herved er MCSE der tager højde for den egentlig information i datasættet givet ved.

$$\text{MCSE} = \frac{1}{\sqrt{S_{\text{eff}}}} \widehat{\text{sd}}(f(\theta))$$

Change point detection. Givet et dataset vil vi undersøge om vi kan identificerer et punkt hvor i en trend ændrer sig:



$$c \sim \mathcal{U}[1, N]$$

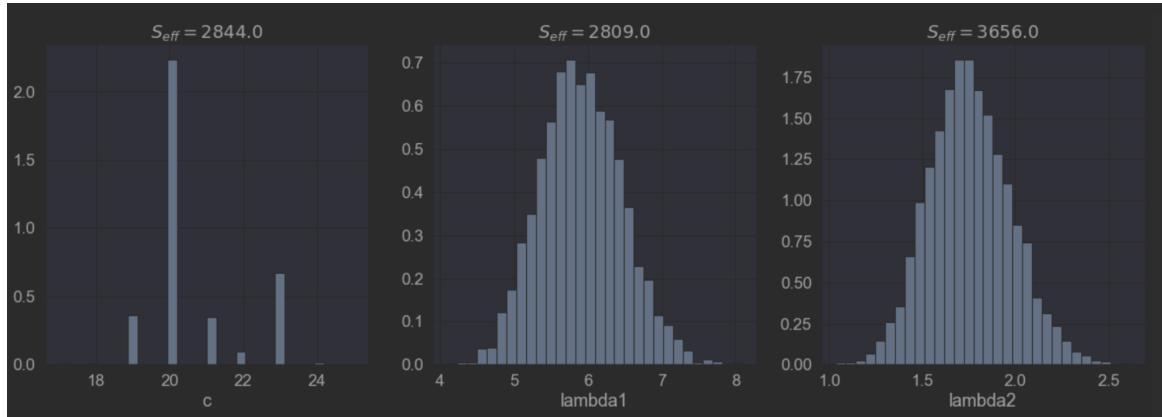
$$\lambda_1 \sim \text{Gamma}(\alpha, \beta)$$

$$\lambda_2 \sim \text{Gamma}(\alpha, \beta)$$

$$x_i | \lambda_1, \lambda_2, c \sim \begin{cases} \text{Poisson}(\lambda_1) & \text{if } 1 \leq i \leq c \\ \text{Poisson}(\lambda_2) & \text{if } c < i \leq N \end{cases}$$

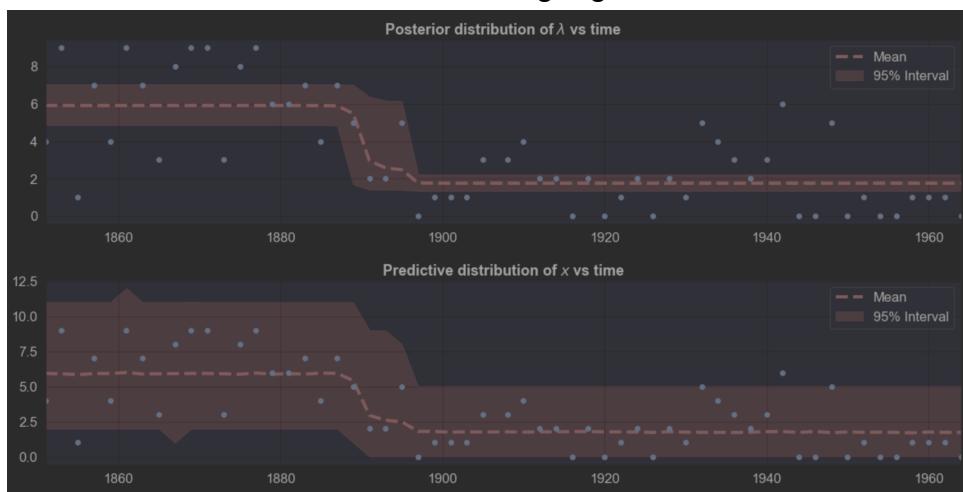
hvor vi kunne formulerer ovenstående model, med fully bayesian treatment hvor vores hyperparameter har hyper priors. Modellen siger ganske simpelt at hvis vi er under et change point c så følger daten en poisson fordeling og hvis vi er over følger dataen en anden.

Hvis man kører gibbs sampling med 4 chains af 2000 samples hver for den model og det data får vi.



Vi ser at vores effektive samples sizes er meget store og desuden er vores r^2 værdi er 1.00 for alle parametre.

Nu ville vi meget let kunne svare på hvad sandsynligheden for at at change point kom i 1870 eks. eller at λ_1 er 3 gange så står som λ_2 osv.



desuden kan vi meget let sample en fordeling over x ved at sample c, λ_1, λ_2 , for vores model og dermed konstruere en predictive fordeling.

Week 9

1. Variational inference (KL divergence and ELBO)
2. Bayesian formulation of the Gaussian mixture model

Som altid i det bayesianske framework har vi en noget data og en model som tilsammen udtrykkes ved en joint fordeling. Vi ønsker da at modererer modellens parametre ved at conditionere dens parametre på dataen. Det gøres ved benytte bayes sætning hvor ved vi får vores posterior på den ene side som er proportional til vores prior og likelihood på den anden. Hvor proportionalitetskonstanten er vores marginal eller evidence, som vi som udgangspunkt kan se bort fra.

Det er mange måde at estimere vores posterior på. eksempelvis analytisk i de tilfælde hvor vores model er meget simpel og der findes en analytisk løsning, men som oftest må man tage approximationer i brug. Det kunne være en laplace approximation eller man kunne sample fra sin posterior med en afart MCMC, hastings, metropolis hastings gibbs eller lignende.

Eller vi kunne bruge en helt 3. metode: variational inference. Vi hvor ganske simpelt ønsker at approksimere vores sande posterior med en simplere sandsynlighedsfordeling, som vi selv kan vælge og tilpasse alt efter problemet:

Metoden hertil kan deles op i 3 steps:

- **vælg variational family. vælg afstandsmål. estimer parametre i din variational family der minimerer afstanden mellem, dem og posterioren.**

$$q_* = \arg \min_{q \in Q} \mathbb{D}[q||p]$$

Vælg variational family

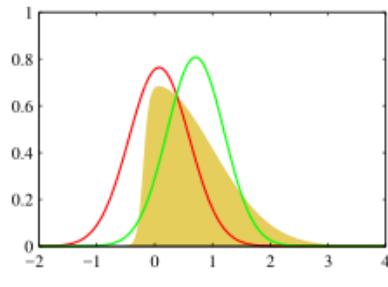
- Først definerer man en gruppe af simplere sandsynlighedsfordelinger som man ønsker at approksimere sin posterior med. De defineres Q og kaldes for the variational family, en instance af Q skrives lille q.
- Der er en række af approaches til at vælge sin variational family og valget af q giver en mulighed for at trade off speed/accuracy, alt efter hvor simpel q er bliver den nemmere at fitte, men bliver også mere approksimativ.
- **Full rank gaussian.** hvor man bruger en multivariate gaussian der har samme dimensioner som antallet af parametre i din posterior.
- **Factoriced** som kan tolkes lidt som en blok diagonal af en full rank.
- **Mean-field** Som bare svarer til diagonalen af en full rank. Mean field gaussian er meget populær, man kan selvfølgelig også skiftes ud med andre fordeling en gaussian.

Distance measure:

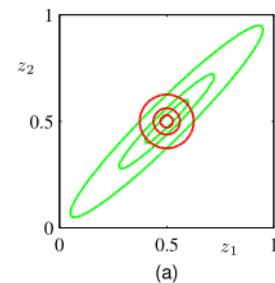
- Et meget populært distance mål mellem sandsynlighedsfordelinger er KL divergence og er givet ved:

$$KL[q||p] = \int q(z) \ln \left[\frac{q(z)}{p(z)} \right] dz$$

- **Den kaldes en divergence** fordi det ikke er et reelt afstandsmål da den ikke er symmetrisk, og afstanden fra q til p er forskellig fra p til q.
- **Men den er altid positiv**
- **Den er kun nul når de 2 fordelinger er identiske.**



laplace=rød, VA-q=grøn, true=gul



VA-q=rød, true=grøn

- Giver en meget bedre approksimation end Laplace for ikke symmetriske fordelinger.
- Under estimerer variance fordi den er mere optaget af at sandsynlighedsdensiteten af q overlapper med p end omvendt.
- **Der findes mange andre similaritetsmål.** vi kunne eksempelvis blot bytte rundt på q og p i KL hvorved i stedet vil bruge expectation propagation, som vil give os en anden løsning der måske ville være bedre afhængigt af problemet.

3. skridt at minimere KL divergence.

- **Hvis man omskriver i KL divergence får man:**

$$\ln p(\mathcal{D}) = \mathbb{E}_q [\ln p(\mathcal{D}, z)] - \mathbb{E}_q [\ln q(z)] + KL[q||p]$$

- **Det i blå kalder vi evidence lower bound:**

Vi ser at højre siden er en konstant da det er log marginal fordelingen over vores data som jo ikke ændre sig. den kaldes også evidence. Vi ser nu at hvis vi forøger vores elbo (det blå term) må det betyde at KL må falde da lighedstegnet skal holde.

- **Den kaldes elbo, evidence lower bound** da den altid vil være mindre end evidence/marginal likelihood. og kun lig med den i det tilfælde hvor KL er 0
- **Det viser sig at det er nemmere at maksimere elbo** end at minimere KL så det gør vi:

$$q^* = \arg \min_{q \in \mathcal{Q}} KL[q||p] = \arg \max_{q \in \mathcal{Q}} \mathcal{L}[q]$$

- **Coordinate ascent variational inference (CAVI)**

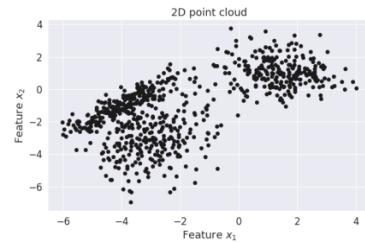
- For en factorised variational family viser det sig at den optimale faktor er givet ved (jeg ønsker ikke at uddybe hvorfor):

$$\ln q^*(w_k) = \mathbb{E}_{\prod_{i \neq k} q(w_i)} [\ln p(\mathcal{D}, w)] + K$$

- **Den optimale fordeling over $q(w_k)$ fremskaffes** ved at tage log join fordelingen af model og data og average den med hensyn til andre andre faktorer.
- **Herved har vi nu en opdatering regel.** For K iteration opdater hver faktor en ad gangen givet ovenstående regel. Monitorer undervejs ved at beregne ELBO og så om den bliver større.
- **Minder meget om gibbs sampling.** Men hvor vi ikke sampler fra den konditionerede fordeling, men i stedet beregner en expectation med hensyn til.

Unsupervised learning: density estimation med gaussian mixture models

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)$$



Kan vi dele datasættet ind i K clusters?

Her kunne vi bruge en gaussian mixture model med K komponenter. Som egentlig bare er K gaussians hver vægtet med en mixing component π_i . π_i summerer til 1.

MLE: expectation maximization. En iterativ algoritme til fitte gaussian mixture models til ikke supervised dataset.

- **Komponenter kan kollapse** på enkelte punkter hvor ved variance gøres uendelig lille og algoritmen crasher
- **Det kan være svært at afgøre hvad K skal være.** Kan være meget sensitiv til initialisering. Bayesian forsøger at komme nogle af de problemer til livs. især nr1.

$$p(x|z) = \sum_{k=1}^K \mathcal{N}(x | \mu_k, \Lambda_k^{-1})^{z_{nk}}$$

$$p(z) = \text{Categorical}(z|\pi) = \prod_{k=1}^K \pi_k^{z_k}$$

Vi skal bruge priors:

- Først indfører vi z_n som er en onehot encoded vektor der udtrykker klassen.
- Den er også categorical og den conjugate prior til en categorical er directlet fordeling. som er genereliseringen af beta fordeling fra binær klasser til multi

- Wishart fordelingen er generalisering af gammafordelingen fra en dimension til multi dimensioner. Og er en fordeling over symmetriske positiv definite matrice, som jo er vældig egnet til covariance/præcision matricer.
Fra den kan vi få sample K gange. en for hver kluster. de bruges også til at sample clusters means μ_k med, men bruges også i selve clustered.

$$\begin{aligned}\pi &\sim \text{Dirichlet}(\alpha_0) \\ \Lambda_k &\sim \text{Wishart}(W_0, \nu_0) \\ \mu_k | \Lambda_k &\sim \text{Normal}(\mu_0, (\beta_0 \Lambda_k)^{-1}) \\ z_n &\sim \text{Categorical}(\pi) \\ x_n | \mu, \Lambda, z_n &\sim \text{Normal}(\mu_{z_n}, \Lambda_{z_n}^{-1}),\end{aligned}$$

Hvis vi skal beregne vores posterior af den fætter her kræver det for at kunne beregne evidence at vi skal summe over all tænkelige K^n muligt assignments: Det eksploderer fuldstændig i kompleksitet.

Men hvis vi blot laver den meget harmløse antagelse er at vi kan lave faktoriseringen hvor vi trækker $q(Z)$ ud af vores joint:

$$\begin{aligned}q(Z, \pi, \mu, \Lambda) &= q(Z)q(\pi, \mu, \Lambda) \\ &= \underbrace{\prod_{n=1}^N \text{Categorical}(z_n | r_n)}_{q(Z)} \underbrace{\text{Dir}(\pi | \alpha)}_{q(\pi)} \underbrace{\prod_{k=1}^K \mathcal{N}(\mu_k | \mu_k, [\beta_k \Lambda_k^{-1}])}_{q(\mu, \Lambda)} \underbrace{\mathcal{W}(\Lambda_k | W_k, \nu_k)}_{q(\Lambda)}\end{aligned}$$

Kan vi få opdaterings reglerne til cavi således:

$$\begin{aligned}\ln q(Z) &\propto \mathbb{E}_{q(\mu, \Lambda, \pi)} [\ln p(X, Z, \mu, \Lambda, \pi)] \\ \ln q(\pi, \mu, \Lambda) &\propto \mathbb{E}_{q(Z)} [\ln p(X, Z, \mu, \Lambda, \pi)]\end{aligned}$$

Ovenstående udtryk skal selvfølgelig stadig udledes analytisk men det er ladsiggørligt.

Week 10

1. This week cannot be drawn as an exam topic, but is still part of the curriculum

Week 11

1. Black-box variational inference
2. Stochastic optimization

Målet er at kunne bestemme vores posterior distribution over weights givet vores data. Variational inference er et forsøg på at approksimere vores posterior.

Variational inference kan groft sagt deles op i 3 skridt:

- **Man bestemmer sig for et set af “simple” sandsynlighedsfordelinger,** kaldet en variational family. Det kunne eksempelvis være en fullrank gaussian eller en række mean field af gaussian. Det er det set af sandsynlighedsfordelinger vi ønsker at approksimere vores posterior med.
- **Dernæst bestemmer vi os for et afstandsmål.** Eksempelvis KL divergence der er disimilaritetsmål mellem 2 sandsynlighedsfordelinger. Den er ikke symmetrisk og afstanden fra a til b er ikke den samme som b til a. Den er strengt større end 0 og er kun 0 i tilfældet hvor de 2 sandsynligheder er ens.
- **Det sidste step** Er at minimere afstanden get ved eks KL divergencen mellem sin variational family og sin posterior:

$$q_* = \arg \min_{q \in \mathcal{Q}} \mathbb{D}[q||p]$$

I praksis for KL divergensen gælder der at den kan omskrives til:

$$\ln p(\mathcal{D}) = \mathbb{E}_q [\ln p(\mathcal{D}, \mathbf{z})] - \mathbb{E}_q [\ln q(\mathbf{z})] + \text{KL}[q||p]$$

Venstresiden er vores marginal eller evidence og er konstant.

Det vil altså sige at hvis vi betragter højre siden kan observer at vi kan minimere KL divergencen ved at maksimere udtrykket i blå der er vores ELBO, evidence lower bound. Og har navnet fordi den altid vil være mindre end vores marginal, og kun i det tilfælde hvor $q=p$ og KL divergence giver nul vil vores ELBO være lig the log marginal.

Hvis vores Q er en meanfield family gælder der at hvert mean field q^* der minimerer KL divergence kan udtrykkes:

$$\ln q^*(\mathbf{w}_k) = \mathbb{E}_{i \neq k} [\ln p(\mathbf{t}, \mathbf{w})] + K \quad q(\mathbf{w}) = \prod_{j=1}^J q(\mathbf{w}_j), \quad \text{where } \mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_J]$$

Med andre ord: det er en expectation taget over vores log joint fordeling af parametre og data. Expectation er taget med hensyn til alle andre parametre end \mathbf{w}_k .

Dette giver anledning til en iterativ algoritme hvor man på tur for hver mean field udregner ovenstående expectation. Undervejs kan man udregne vores ELBO for at tjekke progression og se om vi er konvergeret. Den her algoritme hedder CAVI. coordinatewise ascent variational inference.

Problemet er bare hver gang vi ændre model skal vi på ny analytisk udlede hvorledes de her expectations skal beregnes.

Black box variational inference:

Vi giver op på ideen om at skulle udlede vores variational family i dens optimale form og affinder os i stedet med et fixed-form variational family så som mean field gaussians. Nu vil vi optimere vores elbo direkte ved at finde de parametre lambda der maksimerer den. Det vil vi gøre med traditionelle numeriske optimeringsmetoder.

$$\lambda^* = \arg \max_{\lambda} \mathcal{L}[q_{\lambda}] = \arg \max_{\lambda} \{\mathbb{E}_{q_{\lambda}}[\ln p(\mathbf{t}, \mathbf{w})] - \mathbb{E}_{q_{\lambda}}[\ln q_{\lambda}(\mathbf{w})]\}$$

umiddelbart står vi overfor 2 problemer: at evaluere vores elbo, og at beregne gradienterne at vores elbo i et givet punkt.

Det andet led af vores elbo er vores også kaldet entropy og kan for mange variational families blot bestemmes analytisk og tilsvarende kan dens gradienter. Eksempelvis er den for en mean field gaussian:

$$\mathcal{H}[q(w_i)] \equiv -\mathbb{E}_{q(w_i)}[\ln q(w_i)] = \frac{1}{2} \log(2\pi e v_i)$$

Nu mangler vi kun expectation over vores logjoint. Dette kan estimeres med monte carlo samples, hvis vi blot sampler vægte fra vores variational family:

$$\mathbb{E}_q[\ln p(\mathbf{t}, \mathbf{w})] = \int q(\mathbf{w}) \ln p(\mathbf{t}, \mathbf{w}) d\mathbf{w} \approx \frac{1}{S} \sum_{s=1}^S \ln p(\mathbf{t}, \mathbf{w}^{(s)}) \quad \mathbf{w}^{(s)} \sim q_{\lambda}(\mathbf{w}) \quad \text{for } s = 1, \dots, S$$

Desværre kan vi ikke bare bruge monte carlo samples til at vi havde en sandsynlighedsfordeling over gradienter som vi kunne sample fra $\nabla_{\lambda} q_{\lambda}(\mathbf{w})$, hvilket vi jo ikke har.

Der er 2 approaches: Og jeg ønsker ikke at gå i detaljer med nogen af dem:
Score function gradient estimator som opnåes ved at benytte log derivative trick:

$$\nabla_{\lambda} \mathbb{E}_{q_{\lambda}}[\ln p(\mathbf{t}, \mathbf{w})] \approx \frac{1}{S} \sum_{s=1}^S \nabla_{\lambda} \ln q_{\lambda}(\mathbf{w}^s) \ln p(\mathbf{t}, \mathbf{w}^s) \quad \text{where } \mathbf{w}^s \sim q_{\lambda}$$

Og så er der reparamitration trick som er mindre generelt og ikke fungerer på diskrete variable. men det mindske variance af ens gradienter således man kan tage mere præcise skridt og derfor større skridt således optimeringen konvergerer hurtigere.

For store datasæt kan det være meget dyrt at skulle beregne N expectation for hver iteration, derfor kan vi indføre yderligere stochasticity i et trade off for performance. Her er der tale om mini bathing hvor man for hver iteration kun bruger et subset M af dataen, altså en batch dette kan drastisk forbedre convergnce tiden:

$$\mathcal{L}[q] \approx \frac{1}{S} \sum_{s=1}^S \left[\frac{N}{M} \sum_{n \in M} \ln p(t_n, \mathbf{w}^s) + \ln p(\mathbf{w}^s) \right] + \mathcal{H}[q]$$

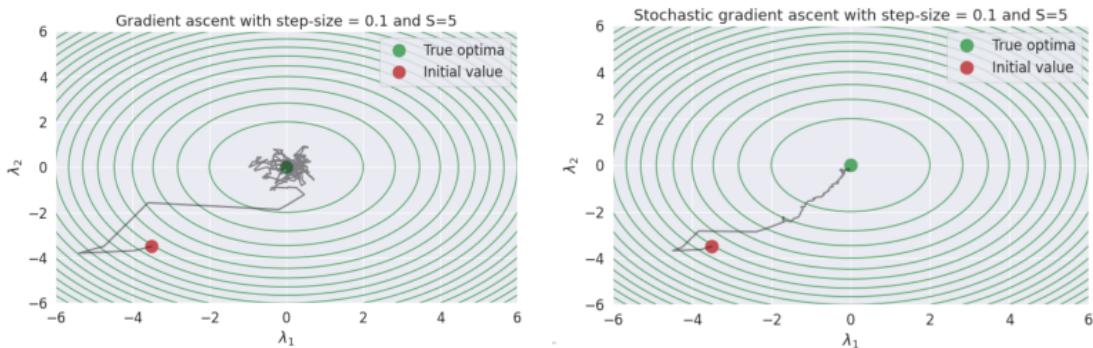
2 elementer af stocastistet. Dels estimeres vores expectations ved brug af S monte carlo samples og dels beregnes de kun på et subset af data M. Ved at øge både M og S kan vi trade off præcision for performance.

Fordi vi har 1 eller 2 elementer af stocastisitet er der ingen konvergens garantier for gradient accent.

Men hvis vi i stedet bruger en stochastic gradient accent optimizer.

$$\boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^t + \eta \hat{g}(\boldsymbol{\lambda}^t) \quad \boldsymbol{\lambda}^{t+1} = \boldsymbol{\lambda}^t + \rho_t \eta \hat{g}(\boldsymbol{\lambda}^t)$$

Hvilket vil sige at lærings raten “nu” bliver gradvist mindre for hver iteration, kun da kan der garanteres convergence. Hvis ikke vil vi se at gradienterne bliver ved med at overshoote vores optimum som i plottet.



Der er en lang række af optimerings tricks der kan forøge konvergens hastigheden.

Momentum: hvor man bruger gradienterne fra tidligere runder ogs. Hvis man forestiller sig en kugle der ruller ned gennem en dal vil den skiftevis rulle op ad siderne hvis vi ikke bruger momentum

Adaptive læringsrate der eksempelvis afhænger af størrelsen på gradienterne.

Individuel læring: Hvor man lader de individuelle parametre have individuelle lærings rater.

Adam: Er en meget populær optimizer der kombinerer alle disse metoder og fungerer rigtig godt i praksis og er et rigtig godt starting point når man vil træne sin model

Week 12

1. The Erdős-Rényi model
2. The infinite relational model

Det viser sig at vi også kan modellerer grafer med vores baysian approach.

En graf er givet ved en række nodes og en række edges, hvor en edge er en forbindelse af 2 nodes.

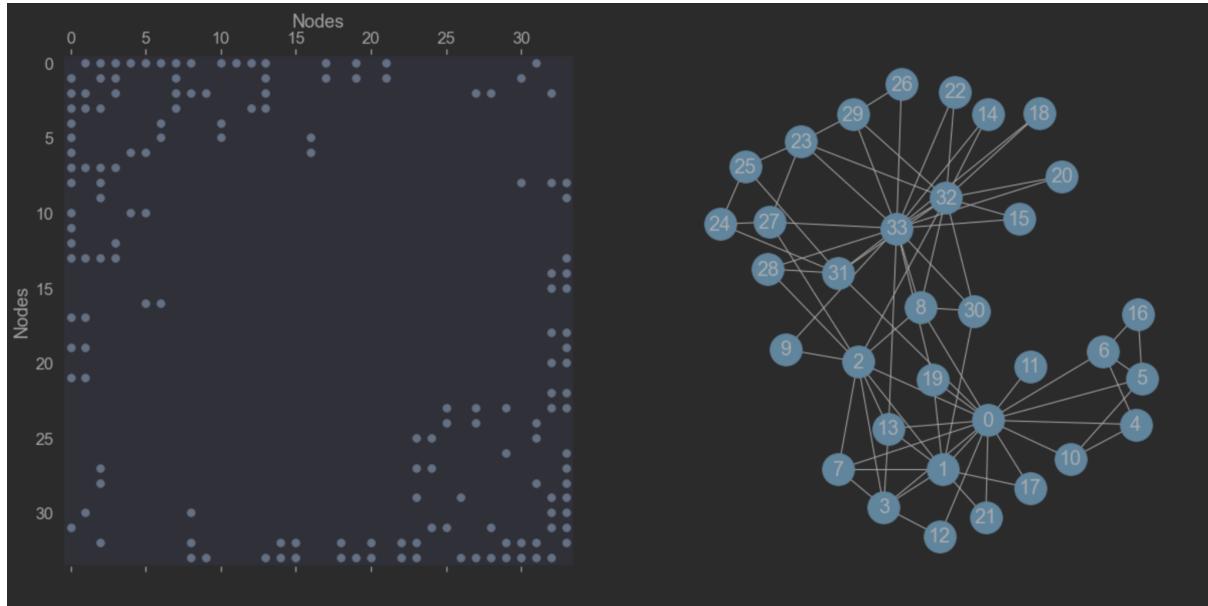
Graferne kan være directed: hvilket vil sige at vi skelner mellem en edge fra a til b og en edge fra b til a.

En graf kan også være weighted, hvilket vil sige at enten nodesne eller edges har individuelle vægte.

Vi vil kun modellere undirected unweighted graphs.

Et virkelighedsnært eksempel på sådan en graf kunne være et socialt netværk over en gruppe mennesker. Hver person ville være en node og hver edge ville være om de kendte hinanden. Den ville typisk være undirected og unweighted.

Grafer kan man repræsenterer med en adjacency matrix: hvor for eksempel index i, j i matricen er 1 hvis der er en edge mellem noden svarende til henholdsvis i og j, og 0 hvis der ikke er en edge mellem de 2 nodes. Da vi kun vil arbejde med undirected grafer vil vores adjacency matrix være symmetrisk da en edge fra i til j tilsvarende er en edge fra j til i.



Graf statistikker - degree er vores centrale sammenlignings matric

Antallet af af edges hver node har. Og så ville man sammenligne 2 grafer ved at kigge på deres degree fordeling:

Her kan man kigge på **max degree, avg degree, eller standardafvigelsen**

Erdos Renyi model:

Her antager vi ganske simpelt at der er EN enkelt parameter phi der afgør sandsynligheden for at 2 nodes er forbundet med en edge i vores graf. Phi er konstant for alle kombinationer af nodes i vores graf.

Og vores likelihood kan defineres for hele grafen.

$$\begin{aligned}
 p(\mathbf{X}|\phi) &= \prod_{(i,j)} \text{Ber}(X_{ij}|\phi) \\
 &= \prod_{(i,j)} [1 - \phi]^{1-X_{ij}} \phi^{X_{ij}} \\
 &= [1 - \phi]^{\bar{m}} \phi^m
 \end{aligned}$$

Og er er produktet over bernoullis sandsynligheder for alle kombination af nodes der ligger over/under diagonalen i matricen. Den kan forkortes. hvor:

$m \equiv$ number of links

$\bar{m} \equiv$ number of non-links

De er "sufficient" statistiks og når vi har beregnet dem kan vi smide resten af vores data væk.

Nu mangler vi blot en prior og det kunne passende være en beta fordeling da den er conjugate prior til vores bernoullis, hvorved vores posterior bliver beta parametriceret ved:

$$\begin{aligned}
p(\phi | \mathbf{X}) &\propto P(\mathbf{X} | \phi) p(\phi) \\
&= \phi^{a+m-1} (1-\phi)^{b+\bar{m}-1} \\
&\propto \text{Beta}(a+m, b+\bar{m})
\end{aligned}$$

Problemet er at alle nodes har samme sandsynlighed for edges og der bliver ikke skabt særlig store hubs/grupperinger: en node med mange edges som man ellers ser i virkelig grafer.

Jeg vil kun kort nævne The stochastic block mode:

Adjacency matrix is structured into blocks

	1	2	3	4	5	6	7	8	9
1	ϕ_{11}			ϕ_{12}			ϕ_{13}		
2									
3									
4									
5				ϕ_{22}			ϕ_{23}		
6									
7									
8							ϕ_{33}		
9									

$z_i | \pi \sim \text{Cat}(\pi)$
 $\pi \sim \text{Dirichlet}(\alpha)$

Her skal vi manuelt spicificere K klasse. Og tildele hver node en klasse z. Det gøres som sædvanligvis med en kategorisk fordeling med parametriseret med ved pi som følger en direchlet fordeling.

Alle blokke opfører sig nu individuelt som en Erdos Renyi model. Men hver Erdos Renyi model har nu 2 phi værdier. en der siger hvad sandsynligheden er for at 2 nodes i samme klasse er forbundet og en for hvis det er nodes for 2 forskellige klasse.

$$\phi_{ij} = \begin{cases} 0.5 & \text{if } i = j \\ 0.05 & \text{otherwise} \end{cases}$$

Problemet med den her model er at vi skal manuelt specificere antallet af klasser K.

The infinite relational model løser det. Den er det særlifælde af stokastisk block hvor vi lader vores prior over z, givet ved alpha og K. Lader alpha gå mod nul og K mod uendelig.

Principielt har vi uendelig mange klasser men dels kan der aldrig være flere klasser end der kan være nodes og dels behøver vi kun at repræsenterer de ikke tomme klasser.

$$\bar{z} \sim \text{CRP}(A, N) \quad P(\text{customer } N+1 \text{ joins table } k | N \text{ customers}) \propto \begin{cases} \frac{|k|}{A+N} & \text{if } k \text{ in use} \\ \frac{A}{A+N} & \text{otherwise} \end{cases}$$

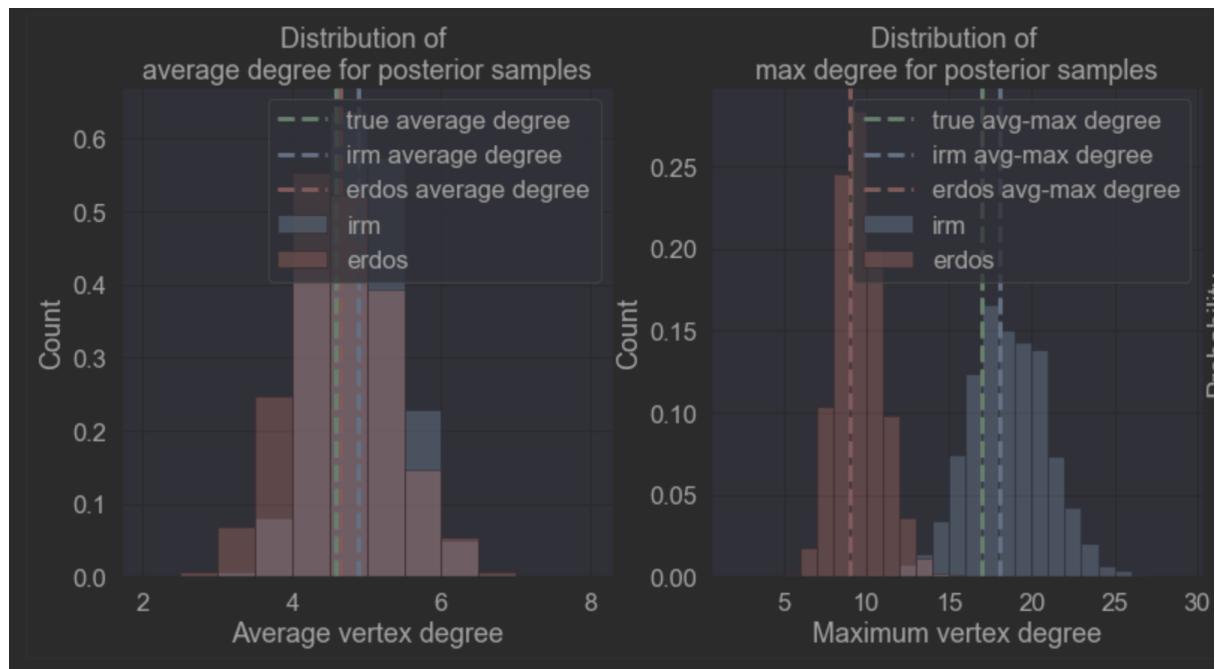
klassetildelingen følger nu en chinese restaurant proces, parametriseret ved A og N. A er tilbøjeligheden til at starte et nyt bord, (en kunde er en node og hvert bord svarer til et cluster).

- Rich gets richer
- omend spinkel så er der altid en ikke 0 sandsynlighed for at starte et nyt cluster.

Hvordan ved vi om vores model passer? Posterior predictive checking.

Vi sampler parameter fra vores posterior og bruger den parameter til at simulere/genererer en graf. gentag et antal gange og beregn væsentlige statistikker og sammen lign med vores observerede.

(For erdos renyi, sampler vi gange fra vores posterior beta fordeling en værdig af phi_s. For hver phi_s laver en en adjacency matrix og beregner avg degree, max degree osv.)



Modellering af karakte datasettet med Erdos og IRM.

De fanger begge 2 average. her er IRM faktisk lidt off. men IRM fanger max distribution meget bedre, den fanger altså den egentlige struktur i netværket meget bedre.

Week 13

1. Regression modelling with heteroscedastic noise
2. Deep ensembles
3. Last-layer Laplace approximations

Deep learning og herunder neurale networks er indenfor det sidste årti blevet state of the art indenfor machine learning. Et simpelt 2 lags NN kunne være givet som følger:

$$\begin{aligned} z_1 &= h(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) \\ z_2 &= h(\mathbf{W}_1 z_1 + \mathbf{b}_1) \quad y_{\mathbf{w}}(\mathbf{x}) = \mathbf{W}_2 h(\mathbf{W}_1 h(\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2 \\ y &= \mathbf{W}_2 z_2 + \mathbf{b}_2 \end{aligned}$$

Hvilket kan udtrykkes som et matrix vektor produkt mellem vægtene og et input \mathbf{x} plus en bias vektor. Dette efterfølges af en ikke linear aktiverings funktion.

Vi kan lave vores NN til en probabilistisk model ved at lade outputtet y for et givent input \mathbf{x} være for eks vores mean parameter i en normal fordeling i et regressions problem eller sandsynlighedsparametre i en bernoullis fordeling for binær klassifikations problem. Nu behøves vi blot at tildele en prior til vores weights \mathbf{W} og \mathbf{b} .

Traditionelt set når man træner et NN vælger man bare et set parametre men der findes typisk mange konfigurationer af parametre som leder til lave losses men til vidt forskellige predictions.

Her er den helt centrale ide i bayesian inference at vi når vi laver predictions vil vi tage højde for alle konfigurationer af parametre, hvilket jo gøres ved at middle over

dem vægtet med vores posterior distribution. I Stedet for at satse alt på et enkelt set parametre.

Ideelt skulle vi jo beregne den egentlige posterior af vores netværk men det er en rigtig svær opgave dels pga den meget komplekse og ikke lineære struktur af et NN og fordi datasæt er blevet så store helt generelt. Derfor må man i stedet approksimere dem.

$$p(t^* | \mathbf{t}) = \int p(t^* | \mathbf{x}^*, \mathbf{w}) p(\mathbf{w} | \mathbf{t}) d\mathbf{w} \approx \int p(t^* | \mathbf{x}^*, \mathbf{w}) q(\mathbf{w}) d\mathbf{w}$$

Her er q vores approksimative posterior, og vores predictive distribution kan bestemmes ud fra monte carlo samples ved at sample forskellige set af vægte fra vores q :

$$p(t^* | \mathbf{t}) \approx \int p(t^* | \mathbf{x}^*, \mathbf{w}) q(\mathbf{w}) d\mathbf{w} \approx \frac{1}{S} \sum_{i=1}^S p(t^* | \mathbf{x}^*, \mathbf{w}^{(i)}) \quad \text{for } \mathbf{w}^{(i)} \sim q(\mathbf{w})$$

Der er overordnet 2 tilgange til at bestemme samples \mathbf{w}_i fra vores posterior approximation for et NN.

Først er det der tilfælde hvor $q(\mathbf{w})$ rent faktisk er en reel sandsynlighedsfordeling: Variational Inference, laplace approximation osv.

Og så er der de tilfælde hvor $q(\mathbf{w})$ ikke er en reel sandsynlighedsfordeling: for eks deep ensambles og MCDO/montecarlo drop out.

Lad os starte med laplace approximation af vores posterior:

Den er givet ved at man finder MAP solution ved numerisk optimering af vores posterior. MODEN bruger man så som mean i en multivariate normalfordeling, og som covariance bruger man den inverse hessian beregnet i MAP solution.

Her er problemet bare at vore hvis vores NN har 1mil parametre så bliver vores hessian 1mil i anden. Hvilket er en absurd stor martrix ikke for nævne at vi skal invertere den hvilket computational complexity N^3 .

Hertil har man fundet på en række approximationer:

- lowrank
- Kfac
- diagonal
- block diagonal: hvis man har et fully connected feedward, så giver det god intuitiv mening af vægte i det første lag næppe har meget indflydse/kovarierer med vægtene i senere lag.

- last layer.

Vi vil beskæftige os med last layer. Ideen er meget simpel vi betragter alle de første lag i netværket som determinisme og beregner kun laplace approksimation på det sidste lag. Her træner vi bare netværket som sædvanligt ved at bestemme MAP og eventuelt kan vi bare bruge et prætränet netværk, hvilket jo er vældig belejligt og gør det mere fleksibelt.

Når vi så skal lave vores posterior predictive distribution for et nyt input x : lave vi et forward pass af x gennem vores netværk og stopper så ved det sidste lag. Dernæst bruger vi vores laplace approximation af det sidste lag til at sample S set af vægte w_s . For hvert sæt af vægte w_s laver vi den sidste del af vores forward pass gennem det sidste lag af vores netværk og får på den vis en række monte carlo samples ud som udgør vores posterior predictive distribution.

Og så er der deep ensembles

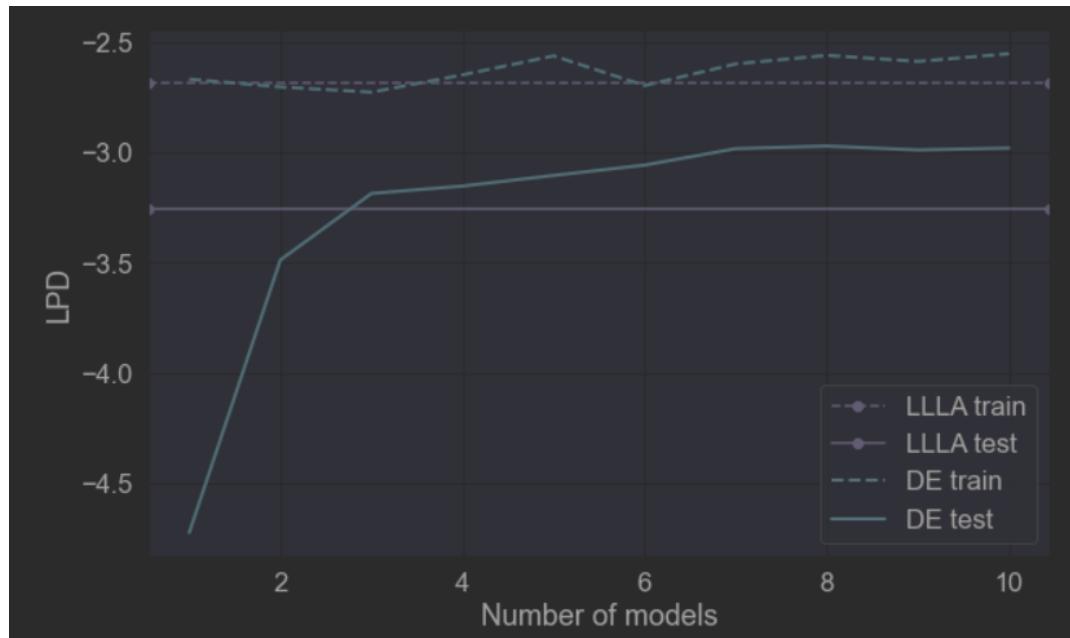
Det er en uhyggeligt simpel men også uhyggeligt effektiv måde at konstruerer dine posterior predictions på. Her træner du simpelthen bare S modeller som er forskellige udelukkende pga forskellige starting conditions. Hvis du ønsker at predicted x laver du for hver af dine modeller et forward pass med x og lader de tilhørende outputs være din posterior predictive distribution af x .

Ulempen er selvfølgelig at du skal træne S modeller og at det tager længere tid at lavet en enkelt prediction fordi du skal lave S forward passes. Men det er meget simpelt at implementere virker som sagt utroligt godt i praksis.

Som honorable mention har vi MC do, montecarlo drop out.

Det har så godt som intet teoretisk grundlag men det er meget enkelt og folk har rapporteret gode resultater med det.

Ved dropout slukker slukker man hver af sine vægte i netværket individuelt med sandsynlighed p . Så vi gentager bare vores forward pass S gange hvor hver af de S svarer til en anden konfiguration af vægte de har været slukket/tændt. Vi kan nu lade outputtet være vores posterior predictive distribution for X .



Hvor LPD er log predictive density og er et performance mål der også tager højde for usikkerheds estimerne.

$$\left\{ y^{(i)}(\mathbf{x}^*) \right\}_{i=1}^S \quad \text{hvor } w \text{ er et sample af vægte: } \left\{ \mathbf{w}^{(i)} \right\}_{i=1}^S$$

Siden det er et regressions havde vi lavet vores NN til en proberlistisk model ved at lavde outputtet y parametriseret en univariat normalfordeling:

$$\text{LPD} = \log \left[\frac{1}{S} \sum_{i=1}^S \mathcal{N}(t^* \mid y_1^{(i)}(\mathbf{x}), e^{y_2^{(i)}(\mathbf{x})}) \right]$$

LPD er altså egentlig bare log af den gennemsnitlige likelihood taget over alle vores y_i .

- Why Bayesian neural networks?

1. Epistemic uncertainty quantification
2. Calibration for predictive distribution
3. Sequential updating
4. Less prone to overfitting
5. Data efficiency
6. Side step pathologies with maximum likelihood learning
7. Active learning

- Posterior geometries of NNs are complicated!

1. NNs are highly non-linear
2. Posteriors are highly multi-modal
3. NNs have weight-space symmetries
4. Often underdetermined by the data