IMPLEMENT
CONSULTING GROUP_

DATA SCIENCE

Hvad er data science?

Big data-revolutionen har efterhånden raset i nogle år, og alle virksomheder og offentlige institutioner har mere eller mindre færdige datavarehuse. Men hvorfor? Vi har fået bedre rapportering og bedre indsigt i de daglige operationer. Mere ledelsesrapportering og flere dashboards. Alt det er sådan set værdifuldt, men der kan opnås langt mere værdi.

Data science lover at bringe operationelle løsninger og endnu dybere indsigt til organisationer. Særligt for data science er ønsket om at udvikle løsninger, som kommer med råd i konkrete operationelle situationer, eller fuldt automatisere et skridt i en arbejdsproces. Det er ikke det eneste, det kan, men det er et af de helt store fokuspunkter.

DATA SCIENCE ER EN DISCIPLIN, DER SAMLER STATISTIK, DATAANALYSE OG MACHINE LEARNINGS FÆLLES METODER I FORSØGET PÅ AT UNDERSØGE OG ANALYSERE VIRKELIGE FÆNOMENER VIA DATA

Et eksempel kunne fx være en læge. En patient kommer ind af døren, og en data science- løsning præsenterer ikke bare et dashboard baseret på patientens journal, men også et sæt af forudsagte opmærksomhedspunkter for lægen baseret på patientens historik, den nyeste forskning og det nuværende sygdomslandskab i verden.

Patienten får foretaget et scan af sine lunger, som giver besvær, og da scanneren afleverer billedet, foreslår den straks, at en region af billedet undersøges nærmere for potentielle kræftknuder og fremhæver straks området. Maskinen er trænet på millioner af billeder verden over, og lægen er straks mere opmærksom.

Efter patienten er gået, dikterer lægen et notat, som automatisk gemmes som tekst. En algoritme bemærker, at der nævnes et scan af lungerne og tilføjer automatisk en procedure-kode på patientens historik. Det bemærkes desuden, at der er mistanke om kræft, og et opmærksomhedsflag rejses på patienten til fremtiden.

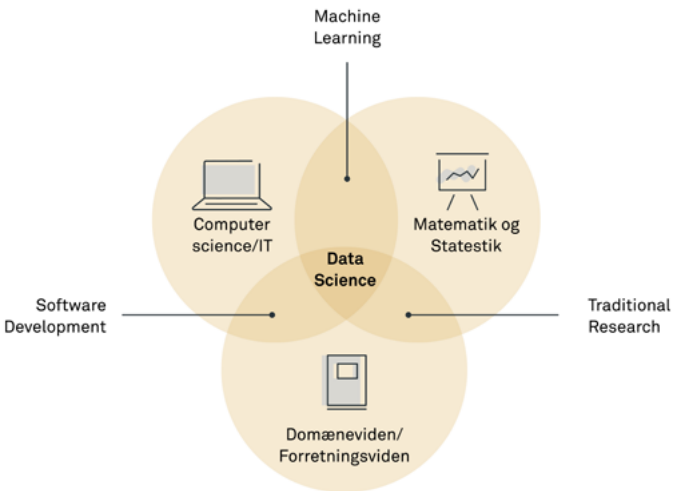
I eksemplet ovenfor blev data science først brugt til at give en bruger overblik ud fra et ellers uoverskueligt informationsgrundlag. Herefter blev der ydet beslutningsstøtte, og et område blev fremhævet. Endelig var der til sidst en fuldautomatisering af journaliseringen af arbejdet.

Vi er næppe helt dér endnu. Men vi er uden tvivl på vej, og i denne guide vil det give indtryk af, hvordan du kan efterprøve og styre data science-eksperimenter i din organisation.

Hvem er denne guide til?
Denne guide er først og fremmest til folk, der vil i gang med at øge data science- kompetencerne i deres organisation. Enten dem selv ved at blive bedre data scientist- eller softwareudviklere eller ved at lede en større eller mindre data science-afdeling. Denne guide giver et dybt indblik i et komplet data science workflow – ikke bare fra data til model, men fra identifikation af det rette problem til opsætning af modellen og til realisering af værdien og vedligeholdelse.

Hvorfor denne guide?
Udfordringen er, at selvom mange organisationer ønsker at begynde med kunstig intelligens, og en del endda forsøger, så møder de systematisk nogle udfordringer, som vi gerne vil forsøge at præsentere en moderne løsning på.

Udfordringen er, at en dygtig data scientist har tre kompetencer, der er yderst sjældne sammen: (1) En stærk forståelse for softwareudvikling, (2) en stærk forståelse for statistik og machine learning og (3) en stærk forståelse for den forretning eller det domæne, de arbejder inden for.



Inspirationen

Vi oplever i Implement en stor efterspørgsel på data science-kompetencer.

Det skyldes primært, at mange organisationer sidder inde med store – og hastigt voksende – mængder af data. Det rejser mange steder det naturlige spørgsmål: Kan al den data ikke bringes i spil og skabe forretningsværdi på en eller anden måde?

Det næste, der oftest sker, er, at man finder en medarbejder, som sættes i gang. Enten med at identificere en anvendelse eller blive givet en konkret opgave, som skal løses med en data science-løsning.

Hvis et af disse elementer mangler, møder data scientisten ofte udfordringer:

- Mangler **softwareudviklerkompetencerne**, risikerer data scientisten at blive en ekspert, der ikke kan levere løsninger, der rent faktisk bliver implementeret og skaber forandring. Løsningerne forbliver for tekniske eller bliver ofte bare aldrig færdige.
- Mangler **statistik- og machine learning-kompetencerne**, vil data scientisten ofte have svært ved at levere den tilstrækkelige kvalitet af løsninger, som der efterspørges. De bliver formentlig færdige, og de løser formentlig problemet, men vedkommende er afskåret fra at takle de nye og svære problemer, som de potentielt kunne, hvis de havde en større forståelse for machine learning.
- Mangler **forretningsforståelsen**, risikerer vedkommende at udvikle flotte løsninger, der ikke finder anvendelse i forretningen, og hvor gevinsterne ved udviklingen aldrig realiseres.

Ikke overraskende er netop udfordringer med at afslutte projekter, opnå den forventede kvalitet og faktisk opleve den ønskede effekt nogle af de mest udbredte problemer i nye data science-projekter.

Denne guides indhold

I denne guide vil du blive taget med på rejsen fra den tidligste start med use case- identifikationen og helt til råd om governance af modeller, når de er lagt ud til anvendelse. Guiden består af syv kapitler med følgende overskrifter:

- **Kapitel 1. Use case-identifikation**: Hvordan sikrer du, at du arbejder på en værdifuld idé?
- **Kapitel 2. Dataeksploration**: Få overblik over dine data, ikke bare mønstre og indhold, men skabelsen og håndteringen.
- **Kapitel 3. Definition af formål**: Hvad skal din model optimere for? Hvad er egentlig problemet, vi forsøger at løse? Og hvordan du sikrer, at din model faktisk gør det, du regner med.
- **Kapitel 4. Machine learning**: Hvad er de statistiske teknikker, vi bruger til at lave vores analyser? Hvordan rammer du den rigtige mængde kompleksitet og sikrer, at din model virker i virkeligheden.
- **Kapitel 5. Etik**: Data science berører mange menneskelige aspekter fra persondata til automatisering. Kapitel 5 giver en indflyvning i de vigtigste overvejelser og konkrete råd til, hvordan ting beskyttes.
- **Kapitel 6. Data science governance**: Når modeller skal ud og skabe værdi, skal de styres og vedligeholdes. De skal måske endda opdateres eller i hvert fald overleve, at it-landskabet omkring dem opdateres. Dertil har vi governance.
- **Kapitel 7. Governanceværktøjer**: Når governance skal implementeres, er der en række hjælpsomme værktøjer. Kapitel 7 giver et indblik i de fem vigtigste fra versionsstyring til modelmonitorering.

Kapitel 1.
Use case-identifikation

Hvordan sikrer vi, at vores data science-løsninger faktisk gør en forskel? Det er lettere sagt end gjort, og data science har nogle særlige faldgruber, man skal være opmærksom på. En del af udfordringen er formentlig, at det faglige overlap mellem folk, der har kendskab til de nyeste muligheder inden for machine learning og AI, ofte er ret unge og ikke nødvendigvis også har studeret innovationsprocessor eller forretningsudvikling ved siden af. Der er en (forståelig) fascination af teknologien, som jeg også selv indimellem kan falde for.

Udfordringen betyder, at mange data science-projekter kommer i gang med det forkerte problem, som enten ikke løser noget reelt værdifuldt eller viser sig umuligt at løse.

Hvad er en (data science) use case?

En use case er to ting: et mål og en vej dertil. Alternativt: et problem og en løsning.

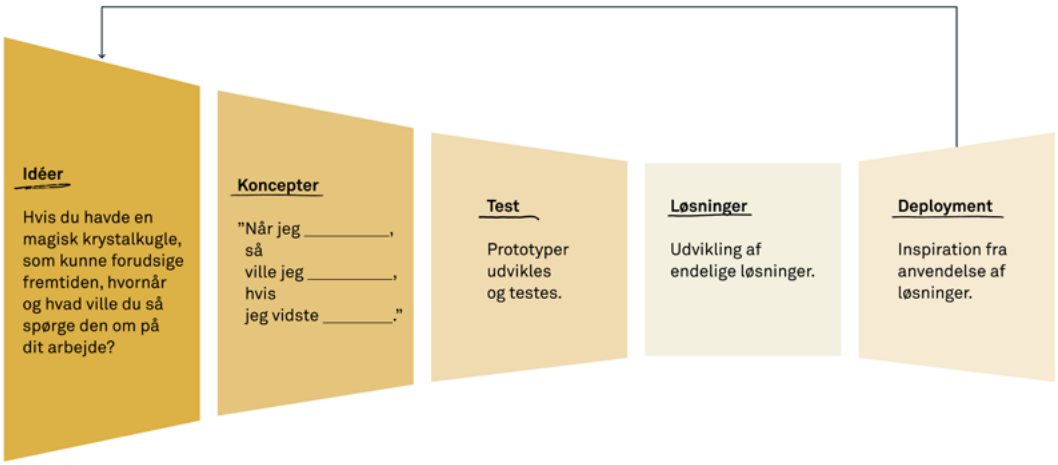
Et eksempel kunne være at oversætte tekster fra fransk til engelsk, og løsningen kunne være et machine translation-system som fx Google Translate.

Use case-identifikation handler om først at identificere interessante og værdifulde mål og problemer og herefter identificere veldesignede løsninger til dem.

Et sted at starte, men ikke stoppe

Hvis der skulle sidde nogen derude, som læser det her og føler, at jeg overser en masse tekniske muligheder (fx unsupervised learning, reinforcement learning), så er svaret ja. Værktøjerne her er ikke en udtømmende liste til at finde enhver data science use case i en organisation, men det er en måde at komme i gang på.

Use case-identifikation



Use case-identifikation kan ligne en klassisk innovationsproces. Der skal genereres et væld af idéer, som skal sorteres, prioriteres og beriges, indtil nogle af dem kan testes og endelig sendes ud i verden til feedback og inspiration.

Idégenerering

Den første regel i data science use case-identifikation er: **Gør det aldrig alene.** Kom ud og mød de potentielle brugere. Lav workshops, eller invitér folk ind. Start gerne med at fortælle om, hvad data science og machine learning er, og så kan et spørgsmål hjælpe til inspiration:

HVIS DU HAVDE EN MAGISK KRYSTALKUGLE, SOM KUNNE FORUDSIGE FREMTIDEN, HVORNÅR OG HVAD VILLE DU SÅ SPØRGE DEN OM PÅ ARBEJDET?

Når de indledende vittigheder om lottotal og venners privatliv er overstået, viser det sig ofte, at det spørgsmål faktisk er sværere end som så. Det kræver, at folk forholder sig meget konkret til det spørgsmål, som de ville stille til den magiske krystalkugle.

Det gode ved spørgsmålet er, at det parkerer diskussioner om, hvad der er realistisk og urealistisk inden for data science i første omgang, og tillader folk at være kreative.

Ofte kommer der mange interessante spørgsmål og diskussioner op, hvoraf rigtig mange naturligvis er umulige, og det er sådan set en del af pointen. Det er meget bedre at kassere en umulig idé tidligt end at udvikle noget realistisk og værdiløst.

Idéberigelse

På nuværende tidspunkt skulle vi gerne have 20-40 idéer noteret ned, som beskriver situationer, hvor viden fra en magisk krystalkugle kunne være interessant. Dette skal konkretiseres yderligere, og vi skal til at være kritiske over for vores problemstillinger. Vi ved fra tidligere, at de er interessante, men er de også *værdifulde*? Formålet med dette trin er at sikre i endnu højere grad, at det **problem, vi løser, faktisk er værdifuldt**. Derfor bruger vi denne sætning:

Jeg er en __, og når jeg __, ville jeg kunne __, hvis jeg vidste __.

Nøglen her er at have en handling i en situation. Se eksemplet her:

Jeg er operatør hos 112, og når jeg tager et opkald, ville jeg kunne sende en ambulance hurtigere, hvis jeg vidste, om det var et hjertestop.

(Se (Corti.ai 2020), for mere om denne use case)

Hvis ikke der kan konstrueres en sådan sætning, er der risiko for, at det, man har identificeret, er et "nice to know" type problem. En situation, hvor nogen gerne ville vide noget, fordi det kunne være rart eller interessant, men måske faktisk ikke er i stand til at bruge informationen til noget værdifuldt.

Disse sætninger giver også konkrete mennesker at gå til for at få idéen verificeret. Er disse personer faktisk i den situation, der bliver beskrevet? Er den information, der bliver foreslået, nok til at træffe den pågældende beslutning/handling? Er handlingen faktisk tilgængelig? Hvis der kan svares ja til alle disse spørgsmål, så har vi en beriget idé.

Tip: Hvis muligt, så spørg personen i sætningen, om de kender noget data, der kunne give et hint om, hvad vi gerne vil vide. Folk er ofte meget kreative og kan have indsigt, som vi ikke har, så udnyt chancen for at snakke med brugerne!

Men hov, hvad med fuldkommen automatisering?
Det er sandt, at handlingen er mindre væsentlig i et fuldkomment automatiseringsscenarie (tag fx Enversion, som fuldautomatiserer fakturahåndtering, så arbejdsopgaven simpelthen forsvinder). Denne type use case falder også uden for værktøjerne her. Ikke fordi, de ikke er interessante, men identifikationen følger et noget andet mønster.

Kritiske spørgsmål

Vi skulle nu gerne have en håndfuld gode koncepter at gå videre med, og det er på tide at angribe dem fra data science-siden. Kan det her faktisk lade sig gøre? Jeg bruger ofte tre tommelfingerregler:

- Hvor lang tid ville det tage at lave vurderingen?
- Ville ti mennesker være nogenlunde enige?
- Findes der data, der kan forklare fænomenet?

Hvor lang tid ville det tage et menneske at lave vurderingen? Dette spørgsmål drejer sig om den forventede kompleksitet i problemet. En gammel regel i data science siger: *Hvis det tager et menneske mere end to sekunder, så kan computeren formentlig ikke løse det.*

I dag er vi noget længere, og jeg ville være villig til at gå op på nogle minutter, men princippet i det er rigtigt. Hvis mennesker oplever det som en svær vurdering, så er maskinerne formentligt mindst lige så udfordrede. De bedste use cases er ofte *nemme* for mennesker. At vurdere, om et billede er en hund eller en kat, er nemt, og vi gør det næsten uden at tænke. At vurdere, om indholdet af en artikel er faktuelt korrekt, kan omvendt kræve, at vi tænker os rigtig godt om.

Ville ti mennesker være nogenlunde enige? Spørgsmålet forsøger at vurdere graden af subjektivitet i et givent problem. Bemærk, at verden er fuld af ting, som vi *ved* er subjektive, men der er endnu flere ting, som vi *tror* er objektive, men som i praksis ikke er det. Kreditvurderinger, karaktergivning, diverse ansøgninger. Alle disse områder er svære for data science, fordi de (på godt og ondt) indeholder rigtig meget subjektivitet.

Det er værd at bemærke her, at en algoritme kan tvinge objektivitet ind i en proces, der måske mangler det. Hvis vi ikke tror, at ti mennesker ville være enige, så er der umiddelbart grund til bekymring, men et opfølgende spørgsmål kunne måske være: *Men burde de være enige?* Hvis ja, så er data science måske stadig interessant, men vi skal træde varsomt, for noget kunne tyde på, at vi kæmper med bias (mere om det i kapitlet om etik).

Findes der data, der kan forklare fænomenet? Nogle fænomener egner sig ikke til forudsigelse, fordi de er grundlæggende uforudsigelige. Jordskælv og vulkanudbrud har længe været mål for videnskaben ikke bare at forstå, men også at kunne forudsige, men det er ikke reelt lykkedes endnu. Store politiske begivenheder som valg og borgerkrige er også notorisk svære at forudsige. Aktiemarkedet er også en klassisk udfordring. Vi kunne formentlig godt opstille *nogle* data, men vi ville næppe kunne opstille nok til at opnå en specielt stærk model.

De bedste use cases er dem, hvor kilden til viden er meget klar. Hvis vi skal diagnosticere, om et ben er brækket, så har vi et røntgenbillede. Svaret på spørgsmålet ("Er benet brækket?") gemmer sig i billedet, men vi *ved*, at svaret findes i billedet.

Test og udvikling

Hvis du nu har nogle idéer tilbage, så har du formentlig en interessant use case. Tillykke! Idéen egner sig formentlig godt til en data science-løsning, og du er rimelig sikker på, at der er en brugergruppe, der har en reel situation, hvor løsningen ville kunne give dem værdi.

Det burde nu være realistisk at begynde at finde ud af, hvor krævende løsningen ville være at udvikle og lave en cost-benefit-analyse af, om det står mål med værdien.

Mere om det i næste kapitel.

Opsummerende

Hvis du skal identificere værdifulde data science use cases, så er det hjælpsomt først at indlede en kreativ proces ved at parkere begrænsningerne i teknologien og fokusere på at identificere situationer, hvor der kunne træffes bedre beslutninger på baggrund af information. Det kan fx gøres med en sætning som: "*Jeg er en __, og når jeg __, ville jeg __, hvis jeg vidste __.*"

Sådan nogle sætninger kan typisk behandles af en data scientist, som ud fra en kritisk vurdering vil kunne prioritere dem og begynde afsøgningen af data.

Kapitel 2.
Definition af formål

Hvis der er en ting, jeg har lært i forbindelse med Implements data science-arbejde, så er det vigtigheden af *formålet*. Data science er forholdsvis unikt ved, at formål ofte vitterligt sættes på formel kaldet en omkostningsfunktion. Kort sagt er det kunsten at sikre, at vores data science-løsning *faktisk* løser det værdifulde problem, vi har identificeret, og ikke en banal (men beslægtet) problemstilling, og at det faktisk sker under realistiske forudsætninger.

Har vi ikke styr på vores omkostningsfunktion, risikerer vi at prioritere vores kunder forkert, eller vi risikerer at optimere for situationer, der ikke opstår.

Indtil videre har vi drøftet, hvad data science er, hvordan vi identificerer et værdifuldt problem, og hvordan vi sikrer os, at vi har de fornødne data til at løse vores problem. Næste skridt i processen er at definere målet og grænserne.

- Hvad er en omkostningsfunktion?
- Hvorfor er omkostningsfunktioner vigtige for forretningen?
- Tip #1. Vægte
- Tip #2. Ikke alle fejl er lige alvorlige
- Tip #3. Lad være at gøre det umulige
- Baselines

Hvad er en omkostningsfunktion?

Når vi træner en machine learning-model, opstiller vi ofte en omkostningsfunktion (også kaldet en loss function). Formålet med en omkostningsfunktion er at beskrive matematisk for modellen, hvad det er, vi forsøger at optimere for.

Typisk beskæftiger vi os med tre typer problemer i machine learning:

1. **Regression:** Gæt et tal, fx en aktiepris.
2. **Klassifikation:** Gæt en type ud af mange mulige, fx en diagnose til en patient.
3. **Binærklassifikation:** Ja/nej-spørgsmål, fx: "Er det spam?"

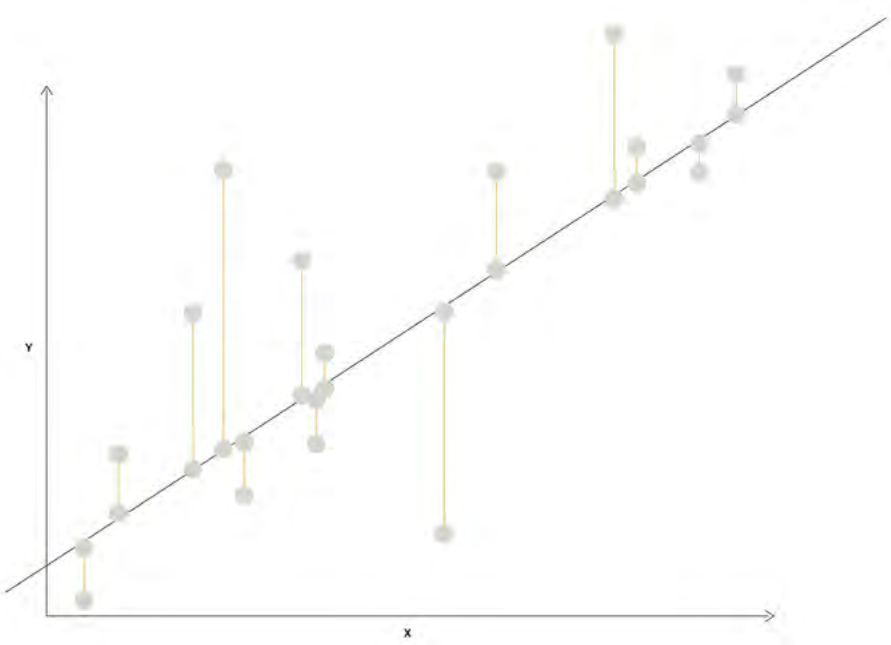
Til hver af disse typer problemer er nogle klassiske formuleringer af omkostningsfunktioner.

På næste side ses fx et eksempel med regression. Den diagonale linje er vores model, og prikkerne er vores observationer. Vi ønsker en linje, der ligger således, at distancen ned til linjen er så lille som muligt for alle prikkerne.

For klassifikation er der flere varianter, men en naturlig løsning er naturligvis bare at se på andelen af rigtige svar og forsøge at estimere en algoritme, der minimerer antallet af forkerte svar.

Hvorfor omkostningsfunktioner er vigtige for forretningen

Det overrasker ofte forretningsfolk, at når vi fx laver regressionsmodeller (altså gæt et tal, fx antallet af varer solgt i morgen), så er standard practice blandt data scientists i dag at minimere den gennemsnitlige fejl (som nævnt ovenfor), men at vægte større fejl eksponentielt tungere, jo større de er.



Denne formulering har nogle appellerende matematiske egenskaber, men er en *meget* væsentlig forretningsbeslutning, der ofte bare glider igennem. I praksis har forretningen ofte holdninger, såsom at *fejl under en vis størrelse er reelt ubetydelige, eller at fejl over dette niveau bør vi stoppe med at skelne mellem. Det er bare "fejl"*.

Gøres dette forkert, kan en data scientist fx fejlagtigt komme til at udvikle en algoritme, der forkastes af forretningen, fordi algoritmen (grundet dens eksponentielle fejl) optimerer for meget for nogle ekstreme perioder på bekostning af den normale hverdag. Modellen bliver i gennemsnit for upræcis, og projektet opgives. Det, selvom en korrekt formulering af omkostningsfunktionen ville have løst det.

Tip #1. Vægte

Spørg forretningen tidligt: "Er noget vigtigere end andet?" Svaret er næsten altid ja. Måske er det vigtigere at finde cancertilfældene end at undgå at tage for mange patienter ind. Præcis hvor meget vigtigere? Det er et vigtigt forretningsspørgsmål, som skal besvares! Et projekt i Danmark fandt fx ud af, at det var 16 gange mere omkostningsfuldt at overse et "ja", der faktisk skulle være et "ja", end at komme til at sige "ja" til noget, der faktisk skulle have været et "nej". Da spørgsmålet blev stillet, viste det sig faktisk, at der var en klar forretningskalkule, der kunne give det præcise svar. Selvsagt ændrede det dramatisk på algoritmen, end hvis "ja" og "nej" var blevet vægtet lige.

Tip #2. Ikke alle fejl er lige alvorlige

Der er ofte også *størrelser* på fejl, som er mere væsentlige end andre for forretningen.

Ligegyldige små fejl: Afhængigt af løsningens anvendelse kan det være, at en fejl på +/- 1% er helt ubetydelig. Så er der ingen grund til at optimere, for den omkostningsfunktion kan designes til at se bort fra fejl under dette niveau, og teknikker som fx support vector machines gør dette meget eksplicit.

Ligegyldige store fejl: En endnu vigtigere kategori er de ligegyldige store fejl. Dette er fejlestimater, som bør håndteres af regler eller mennesker på anden vis. Måske er tærsklen 50%, så hvis løsningen først skyder *meget* forkert, er det faktisk underordnet, om det er 50% eller 500%. Det er ganske enkelt *ikke hjælpsomt* set fra forretningens side.

Mellem disse to filtre får modellen en chance for at fokusere på de cases, hvor den kan gøre en værdifuld forskel. Casene kan sorteres fra i data (her skal man dog tænke sig godt om!) eller implementeres som en del af omkostningsfunktionen.

Tip #3. Lad være at gøre det umulige

Hvis et problem viser sig særligt svært, kan introduktionen af en "ved ikke" eller "grå" kategori give forretningsmæssig mening. I et klassifikationsproblem ("Er det ..."-spørgsmål) sker det ofte ved at tage et stort antal sjældent forekommende typer og lægge sammen til én stor "grå" klasse, som så håndteres manuelt eller efter nogle kendte regler. I regression (hvad er tallet-spørgsmål) kan det ske ved fx først at estimere en model, som forsøger at estimere det præcise tal, hvorefter man inspicerer de cases, hvor modellen laver store fejl. Herefter trænes en individuel model til at vurdere, om en case formentlig vil have en stor eller en lille fejl. Endelig gentrænes den oprindelige model kun på den andel af cases, som havde en lille fejl (og fejlene undersøges så igen, hvor vi gerne skulle se, at performance nu var meget bedre).

Når modellen så skal anvendes, bruges først "stor eller lille fejl"-modellen.

Cases med forventede store fejl håndteres manuelt, mens kun dem med forventet lille fejl får lov at passere videre.

For et mere komplet overblik vil jeg anbefale det overblik, som scikit-learn giver over deres forskellige modeller (scikit-learn 2020).

Baselines

En baseline er, hvad end den nuværende løsning er. Måske er det en simpel regel (vi siger bare altid "ja") eller en ren menneskelig vurdering. Måske beror det på nogle simple heuristikker ("vi ved, at vores medarbejdere generelt siger det samme som i går +/- lidt afhængigt af disse kendte forhold").

Baselines er enormt vigtigt, for ofte er forretningen ikke interesseret i at høre, at vi estimerer tallet rigtigt 84% af tiden. De vil vide, hvor meget bedre end den nuværende praksis vi er – det såkaldte **"modelløft"**. (Eller hvor tæt kan vi komme på den menneskelige baseline med en algoritme, som kan arbejde gratis 24/7?)

Modelløft

Et begreb for, hvor meget bedre en model er i forhold til en baseline. Det kan være fristende at fokusere alene på præcisionen af en model, men i praksis er det ofte forbedringen i forhold til udgangspunktet, der er det afgørende. Altså modelløftet.

Opsummerende

Inden vi begynder at udvælge og designe modeller, bør vi gøre os klart, hvad formålet er, helt ned på det matematiske niveau. Er alle typer fejl lige vigtige? Hvor god er den baseline, vi er oppe imod? Hvad er omkostningen ved at lave en fejl kontra at sende en case til manuel behandling?

De spørgsmål udgør kernen i at designe omkostningsfunktionen for en data science-løsning.

Kapitel 3. Dataeksploration

Dataeksploration er formentlig en af de mest undervurderede data science-discipliner. Man skal nemlig ikke undervurdere den tid, man kan spare på at få en stærk fornemmelse for sin data, inden man påbegynder sin modellering. Man kan spare tid på at kunne gå mere direkte til den gode løsning, og man kan samtidig undgå graverende fejl eller bias senere i arbejdet.

Da dataeksploration ikke egner sig til at sætte på formel, har jeg i stedet samlet de fem vigtigste tips, som jeg prøver at følge, samt nogle af de vigtigste værktøjer.

Forstå datakilden

Alle data kommer et sted fra. Om det er manuel indtastning eller en sensor i en maskine. Det er vigtigt at prøve at forstå kvaliteten og konteksten for kilden.

Hvis det er muligt, så prøv at møde de mennesker, der enten indtaster eller arbejder med maskinen, som skaber data. Hvad er deres indtryk af arbejdet i forbindelse med de data, de skaber? Har de selv tillid til det? Er der undtagelser og skyggetal i deres arbejde?

Hvis det er muligt, så følg data fra kilden og frem til det sted, hvor du regner med at tilgå det. Er der forretningsbeslutninger undervejs, som du skal være opmærksom på, som fx frasorterer nogle data?

Du kan fx stille følgende spørgsmål til kilden eller kildeejeren:

- Ville du selv stole på de data, du leverer videre?
- Er der ting, der ikke bliver opfanget og udtrykt i data?
- Er der data, der ikke bliver leveret videre?

Spørg lokale dataeksperter

Nu forstår du de relevante kilder, men det kan stadig være væsentligt at spørge eksperter på de respektive dataområder.

Hvis du bruger data fra flere kilder eller selv foretager transformationer af dine data, så sørg for, at de transformationer, du foretager, er kendte og godkendte af folk på området. Henter du labels ind fra en tabel og kobler den til nogle data i et andet system? Giver den kobling mening? Hvis den er oplagt, kan det så passe, at du selv laver den, og at den ikke allerede findes?

Du bør også benytte chancen til at spørge ind til datatilgængelighed: Hvis det problem, du søger at løse, er tidsfølsomt, er data så overhovedet opdateret ofte nok til, at det giver mening at lave din løsning?

Du kan fx stille følgende spørgsmål til dataeksperten:

- Kan jeg koble disse kilder sammen? Er der noget, jeg skal være opmærksom på?
- Hvor lang er historikken for de forskellige kilder?
- Hvor ofte opdateres data fra kilderne?
- Hvad er dit indtryk af datakvaliteten på disse områder?

Hold styr på dine datastrukturer

Jeg kommer altid med denne anbefaling: **Hvis det er muligt, så etablér et Analytical Base Table (ABT).**

En ABT er en meget sikker og nem måde at håndtere sine data på i et data science-projekt, og den er enormt velegnet til dataeksploration. En ABT er grundlæggende en stor tabel, der har alle dine observationer som rækker, og kolonnerne er så de egenskaber, som vi kender om vores observationer.

Analytical Base Table (ABT)
En ABT er en tabel, der strengt overholder principperne for tidy data. En ABT har således én række per observation og én variabel per kolonne. Blandt kolonnerne er et eller flere *"targets"*, som er det, vi ønsker at forudsige eller analysere, og én eller flere "features", som er det, vi ved om en given observation, som vi skal bruge som baggrund for vores forudsigelser eller analyser.

Det er en simpel og meget effektiv datarepræsentation, som gør brugen af en lang række data science-løsninger meget intuitiv.

Du bygger din ABT på følgende måde:

1. **Etablér en grundpopulation.** Hvad analyserer du? Er det "sager"? "Sagstrin"? "Kunder"? Eller "Kunder per dag"? Det her spørgsmål er sværere end som så at besvare, så tænk dig godt om. Du har et svar, når du kan svare præcist på, hvor mange observationer din grundpopulation har (hint: Det skal være et helt tal).

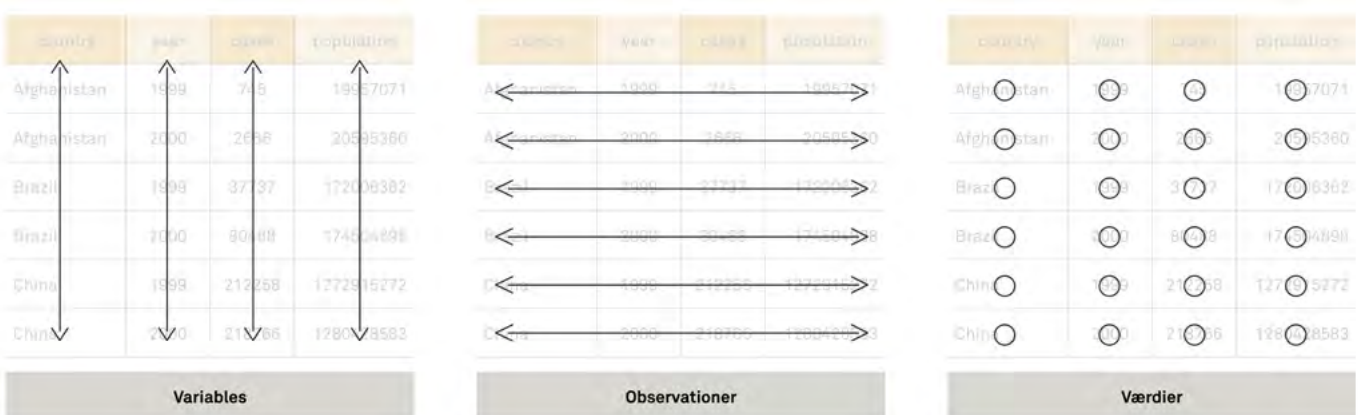
2. **Left join-egenskaber på din grundpopulation.** Et left join sikrer, at der aldrig fjernes observationer fra din grundpopulation (din venstre tabel), når du tilføjer nye egenskaber fra en kilde (den højre tabel). I stedet opstår der missing data, hvis der er en uoverensstemmelse mellem din grundpopulation og din kilde. Det er en vigtig sikkerhed, så du ikke pludselig mister data. Samtidig er det nemt at tjekke, om der udløses dubletter. Du kan altid tælle rækkerne i din grundpopulation, og den må ikke blive større, end da du etablerede den (W3Schools 2020).

Hvis du har gjort det rigtigt, har du såkaldt tidy data (Grolemund og Wickham 2020). Hver observation er i en række, hver egenskab er i en kolonne, og hver celle er en egenskab for en given observation.

Tidy data

Tidy data er en datastruktur, vi altid stræber efter at overholde i alle data science-tabeller. Tidy data har en række per observation, en variabel per kolonne og en værdi per celle. Det er således en flad todimensionel tabel. Det er langt fra altid, vi kan få lov at beholde vores data tidy, når analyser og behov bliver komplicerede, men vi bør altid stræbe efter det.

Eksempel på Tidy data



Visuel inspektion og værktøjer

Hvis du har fået etableret en ABT (eller noget, der ligner), kan du foretage en visuel inspektion af dine data. Det er vigtigt at kigge på sine data. Visualiseringer er meget rigere end lister af fx gennemsnit, standardafvigelser osv. Hvis du vil se et skrækeksempel på, hvordan deskriptive statistikker kan lyve, så se bare på Anscombe's Quartet (Ascombe 1973) eller endnu værre Autodesk Research – Datasaurus (Matejka og Fitzmaurice 2020).

Nedenfor kan du finde tre fremragende og gratis værktøjer til dataeksploration:

- Google Facets: Et meget simpelt og hurtigt dataprofileringsværktøj. Du uploader ABT'en (eller downloader værktøjet og kører det lokalt), og så får du lynhurtigt deskriptiv statistik og simple fordelinger på alle dine centrale egenskaber. Den virkelige værdi kommer imidlertid, når du krydser variable. I Google Facets har du alle data i et simpelt interface, hvilket gør facets til det letteste visuelle inspektionsværktøj (Google 2020).

- Tensorflow Projector: Et væsentligt mere specialiseret værktøj, som er designet til at tage komplekse data (med mange egenskaber) og "koge det ned" til noget, der kan ses i et 3D scatter plot (Tensorflow 2020). Teknikken hedder T-SNE (Wattenberg, Viegas og Ian 2016). Projector er enormt stærkt til at overskue meget komplekse data hurtigt.
  Selv bruger jeg ofte Projector på to måder:

  » Dels til at sikre, at der er en god og varieret struktur i data. Hvis Projector laver en kunstig eller unaturligt udseende form, skyldes det ofte, at strukturen i data ikke er så rig, som vi ellers godt kunne tænke os.

  » Dels er Projector unikt velegnet til at håndtere ekstremt kompleks data, som fx tekst, lyd eller billeder, når det er blevet behandlet og forberedt til analyse.

- Microsoft Power BI: I virkeligheden er det et komplet Business Intelligence-værktøj, men den hurtige regnemotor og det stærke univers af visualiseringer gør Power BI til et vigtigt data-eksplorationsværktøj. Værktøjet er særligt velegnet, fordi du i Power BI kan lave faste rapporter, så du med få klik kan reproducere resultater fra tidligere og holde øje med, at dine data ikke ændrer karakter undervejs i arbejdet som følge af forskellige transformationer. Desuden er Power BI særligt velegnet til at håndtere geografiske data (Microsoft, PowerBI 2020).

Automatisering

Lav automatiserede rapporter, så din dataeksploration – og særligt din dataprofilering – er automatisk og sker ofte. Pakker som fx Pandas Profiling (Profilling 2020) til Python gør det nemt. Hele pointen er at gøre det nemt at lave dataeksploration, så disciplinen bliver holdt oppe, og du får det gjort.

Dataprofilen kan være meget grundlæggende, men sørg for, at du altid har styr på: Hvor mange observationer, hvor mange – og hvilke – egenskaber, hvad er minimum og maksimum af hver egenskab, og hvor mange *missing* datapunkter hver egenskab har.

Automatisering har yderligere den fordel, at det kan gøres igennem et analyseflow. Det er en stor hjælp at have ensartede datarapporter for alle kilder for den samlede ABT og for alle væsentlige transformationsstep hen imod den endelige model og det endelige output.

Disse rapporter er også en essentiel hjælp i overdragelse og vedligeholdelse, der hjælper andre til at forstå de data, som du har arbejdet på.

Kapitel 4. Machine learning

Machine learning er kunsten at have en algoritme, der af sig selv lærer mønstre i data.

Et basalt eksempel kunne fx være en algoritme, som tager en serie af tal (x) og ved at gange dem hver for sig med ét tal (*a*) og lægge ét tal til (*b*) forsøger at få serien til at komme så tæt på en anden serie af tal (*y*). Altså en formel af typen:

$$a*x+b=y$$

Dette er også kendt som lineær regression. Vi ved fra gymnasiet, at vi kan regne en løsning på det her ud, men der er også en anden løsning: Hvad med at vi tager en hurtig computer og prøver 1.000 forskellige tal som a og 1.000 forskellige tal som b? Beregner, hvilken kombination af a og b der i gennemsnit gjorde, at x kom tættest på y. På en moderne bærbar computer tager det under et sekund og giver et resultat, der er meget tæt på den fine matematiske løsning. Vi kan forbedre det endnu mere ved at benytte en algoritme, som kigger på, hvordan forskellige kombinationer af a og b giver et mere eller mindre rigtigt svar, og bruger den information til at gøre enten a eller b større, og forsøger med færrest mulige gæt at få et godt estimat på a og b.

For simpel lineær regression er det måske fjollet (der har vi den gode matematiske løsning), men for mange mate-

matiske modeller (flere er nævnt her) findes der ikke en matematisk perfekt løsning. Så kan algoritmiske løsninger faktisk være vores eneste mulighed, og så er det jo heldigt, at det virker så urimelig godt.

Så nu er dine data klar, og problemstillingen er på plads. Det er tid til at estimere den rette model.

- Hvordan finder vi den rette model?
- Overfit og underfit
- Hvad kan vi gøre?
- Klassiske modeller
- En advarsel
- Bonus: Krydsvalidering

Hvordan finder vi den rette model?

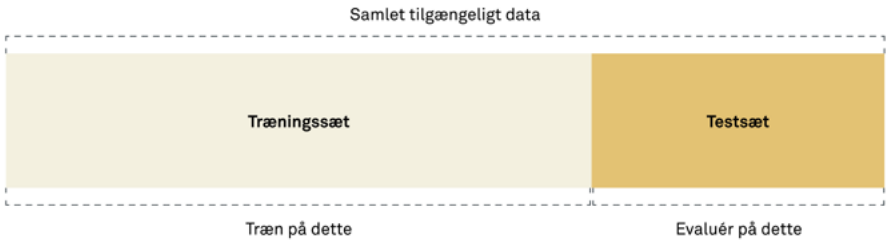
Den bedste model er den model, der gør det, vi gerne vil have den til at gøre i fremtiden, altså på nye data. Det vil sige, at vi er interesseret i at bruge de data, vi har samlet, til at lave en model, der kan gøre os kloge på nye data. Det kunne fx være 100.000 mails, hvoraf 10% er spam, hvortil vi ville lave en model, som kunne fungere som filter på alle fremtidige mails imod vores mailserver.

Den klassiske måde at gøre dette på er med et test-train-split. Princippet er ganske simpelt. Vi tager vores data og splitter det i to klumper: en til "træning" og en til "test".

Vi estimerer så vores model alene på træningsdata og forsøger at bruge denne model til at gætte på vores testdata. Charmen er så, at vi kender sandheden for vores testdata, så vi kan udtale os meget præcist om, hvor nøjagtig modellen er.

Tag fx eksemplet med de 100.000 mails. Vi ville tage 20.000 mails tilfældigt til side (her er der stadig cirka 10%, der er spam). Vi ville så "træne" en model på de 80.000 resterende mails ved at vise den indholdet i mailen – også om den var spam eller ej.

Når vi er tilfredse med modellens evne til at regne ud, om en mail er spam eller ej i træningssættet (måske har den ret 95% af tiden), så er det tid til eksamen! Vi tager så modellen og *tester* den imod de 20.000 mails, vi gemte først. Her rammer den måske rigtigt i 91% af tilfældene. Så kan vi regne med, at vi har en model, som nok i virkeligheden er korrekt cirka 91% af tiden.



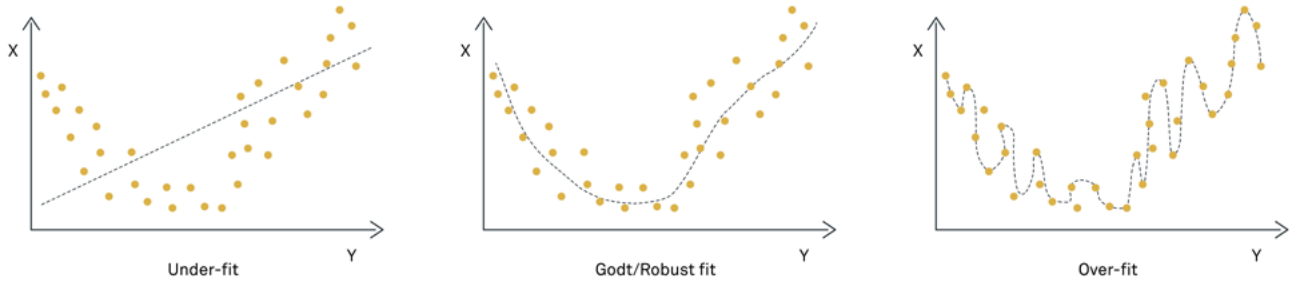
Overfit og underfit

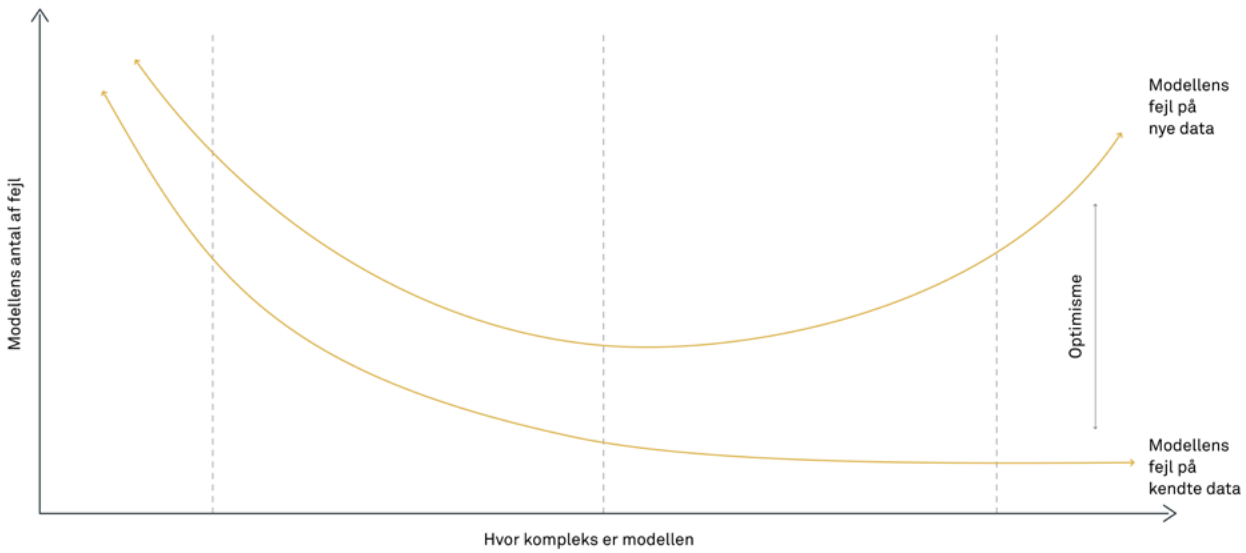
Overfitting og underfitting er de to vigtigste fænomener at balancere, når vi bygger modeller. Overfitting betyder, at vores model er for kompleks og lærer nuancer i data, der ikke har noget med den generelle virkelighed at gøre, men handler om specifikke mønstre i de eksempler, vi har. Intuitivt kan man sige, at svarene læres udenad uden at lære de sammenhænge, der giver svarene. Underfitting er det modsatte, hvor modellen ikke har den tilstrækkelige kompleksitet til at opfange de mønstre, der er i data. Intuitivt kan man sige, at den ikke er klog nok til at lære den information, der er tilgængelig i data.

Overfit og underfit

Grunden til, at vi er nødt til at lave det her test-train-split, er fordi moderne machine learning er *virkelig* godt. Så godt, at jeg med en tilstrækkelig avanceret model (mere om det lige om lidt) kan love, at enhver data scientist kan sikre 100% præcision på træningsdata. De mest avancerede modeller kan lære store datasæt fuldstændig udenad, men de lærer ikke noget *generelt* om det område, de agerer i. Fænomenet kaldes *overfitting* og er helt afgørende i kvaliteten af en machine learning-model. Det kan illustreres som over for.

I nederste diagram ses kernen i udfordringen: Jo større modelkompleksitet, jo bedre bliver modellen på træningsdata og i første omgang også på testdata (den "lærer" faktisk noget). Men efter et vist punkt begynder læringen at gå skævt. Modellen bliver bedre og bedre til træning, men begynder at "overtolke" data så at sige. Når den senere møder nye data, begynder den pludselig at tage *mere* fejl.





Fænomenet er illustreret i de tre figurer ovenfor. Kompleksiteten i modellen kan fx ses som, hvor "kompliceret" en streg vi giver modellen lov til at tegne igennem vores datapunkter. I det første vindue ("Underfit") giver vi kun modellen lov til at tegne en lige linje. Her er modellen bare dårlig. Den passer ikke særlig godt med data, og både test og træning har mange fejl. I næste vindue tillader vi modellen mere. Den må ikke lave skarpe sving, men har lov til at bøje linjen. Nu kan vi se den følge data rigtig fint, og vi kan have en formodning om, at hvis bare fremtidige data følger cirka den samme *struktur* som vores træningsdata, så vil modellen nok gøre et udmærket stykke arbejde.

Til sidst ser vi overfitting-scenariet. Her har modellen fået for meget frihed, og den er begyndt simpelthen bare at forbinde prikkerne. Den er formentlig 100% korrekt (den rammer alle prikkerne), men lagde vi en ny prik ind, så er chancen for, at den ville ligge specielt godt på linjen, meget lille.

Hvad kan vi gøre?

Der er et par klassiske greb i disse scenarier:

Test ofte! Variér parametrene i din model, og se, hvordan forskellen mellem test og træning varierer. Bliv ved med at øge kompleksiteten i din model, indtil du ikke ser nogen forbedringer, eller at forbedringerne kun ligger på træningsdata.

Støt modellen med udefrakommende informationer! Også kaldet feature engineering. Her handler det om, at man manuelt skærer forstyrrende elementer fra i data og forsøger at fremhæve de væsentligste ting. Dette forudsætter, at vi ved en masse om vores data og vores område, men fx i vores

spammail-eksempel er et klassisk trick at smide ord som "at", "og", "det", "der" og "er" væk, fordi de ikke er vigtige for indholdet. Så er der færre ord tilbage til at distrahere modellen. En anden metode kan være at forsimple sproget, så ord som "prinsen", "prinsens" eller "prinserne" reduceres til deres fælles stamme "prins", så modellen ikke skal bruge information på at lære grammatik.

Skaf mere data! Overfitting er altid relativt til en given mængde data. Husk på, at udfordringen er, at modellen kommer til at lære data udenad. Hvis det er muligt bare at give modellen *mere* data, så den ikke længere kan "huske" det hele, bliver modellen i stedet tvunget til at lære mere generelle egenskaber i stedet for, og problemet bliver løst.

Feature engineering
Feature engineering er, når en data scientist manuelt foretager matematiske operationer for at hjælpe en model til at se de relevante mønstre i data. I en situation, hvor vi ønsker at vurdere, om en credit card transaction er svindel, kunne det fx være relevant, at "feature engineere" en gennemsnitlig transaktionsstørrelse for kortholder, en gennemsnitlig transaktionsstørrelse for sælger, antallet af gange, kortholder har købt hos sælger tidligere, og hvor lang tid siden sidst køb. Disse værdier skal beregnes og vil gøre vores model stærkere og mere kompleks.

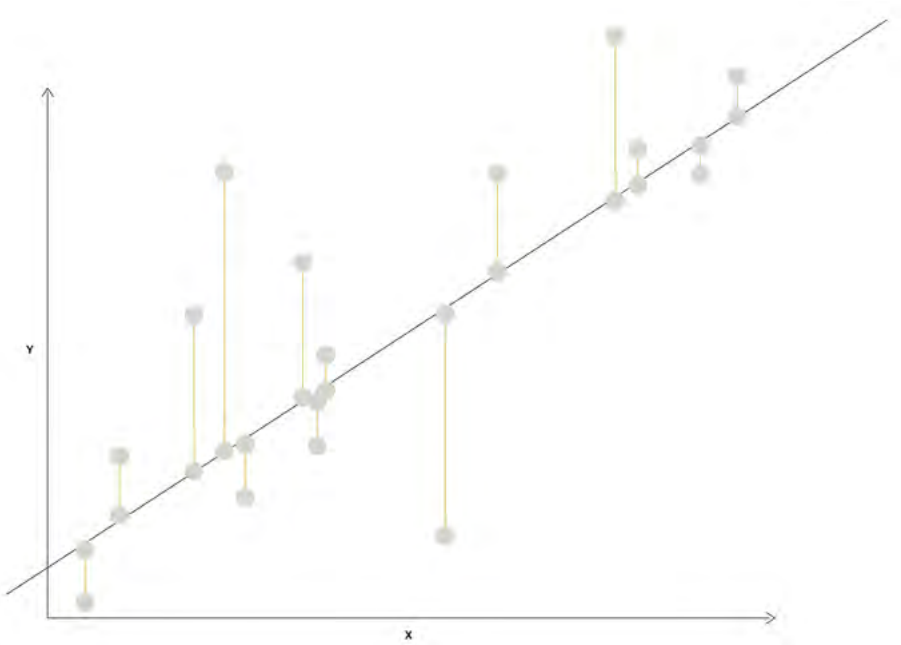
Klassiske modeller

Der ser ud til at være et uudtømmeligt udvalg af modeller inden for data science, som alle løser inden for hver deres lille niche. Men når det så er sagt, er der en håndfuld klassikere, som alle bør kende, og som ofte kan løse 99% af alle ens problemer.

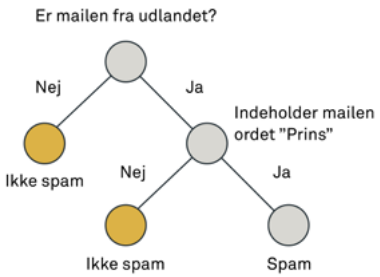
Logit og lineær regression
Klassisk lineær og logistisk regression er stadig i dag nogle af de mest udbredte og hjælpsomme modeller overhovedet.

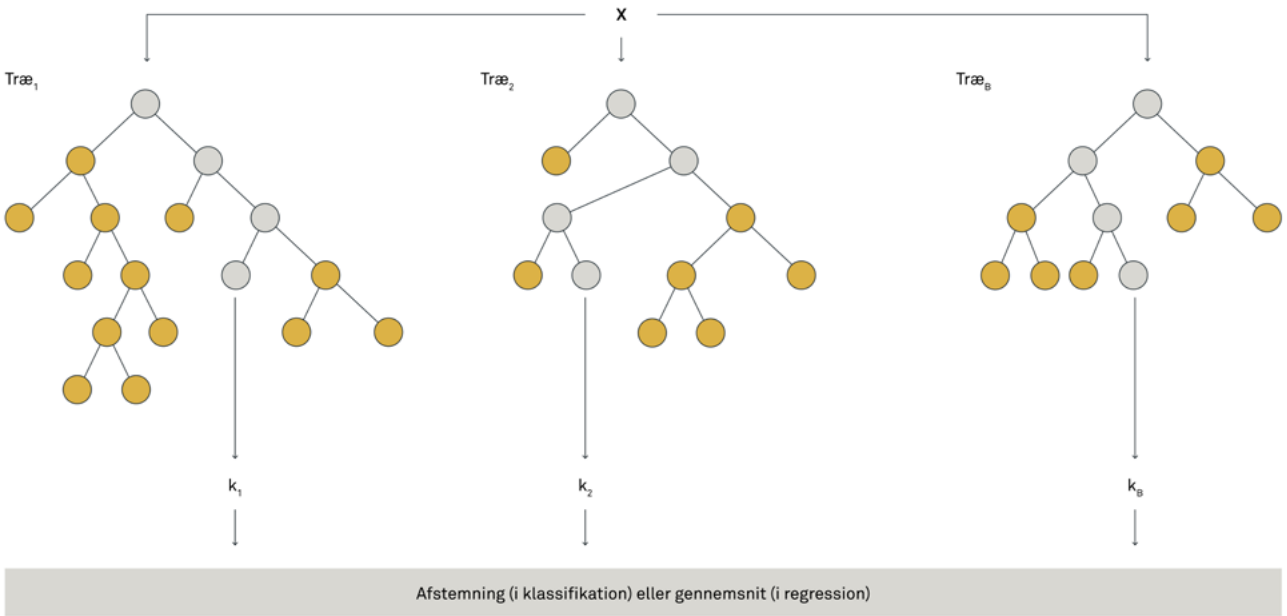
Begge modeller virker ved at forsøge at estimere et givent mål (fx prisen på en aktie i morgen) ud fra en masse input (fx priserne over de sidste syv dage) og så *fitte* en linje. Dette fit foretages således, at når priserne fra de sidste syv dage ganges med de lige linjer og lægges sammen, så får man prisen i morgen. Figuren nedenfor viser cirkaprincippet.

Logistisk regression er en anelse mere kompliceret, selvom princippet er det samme, men her kan linjen have andre former end en lige streg. For eksempel kan den have form som et S.



Beslutningstræer og tilfældige skove
Et mindre berømt alternativ til lineær regression af træbaserede modeller. De er træbaserede, fordi de alle tager udgangspunkt i et såkaldt beslutningstræ. Et (meget simpelt) beslutningstræ til at afgøre, om en mail var spam, kunne fx se således ud:





Der findes teknikker til at lære den slags beslutningstræer ud fra data ved hele tiden at spørge: "Hvad skal jeg splitte på nu, for at de grupper, jeg inddeler i, er mest rene i forhold til det, jeg er interesseret i (fx spam/ikke-spam)."

En videreudvikling af et enkelt beslutningstræ er den såkaldte random forest (på dansk: "tilfældig skov"). Her tager vi et udpluk af vores data (fx 20% af rækkerne og 50% af kolonnerne) og estimerer et lille beslutningstræ på den baggrund. Så tager vi et nyt udpluk af data og laver et nyt beslutningstræ. Sådan bliver vi ved, indtil vi har fx 50 små træer.

Når vi vil have forudsigelser ud, spørger vi hvert enkelt træ og lader dem stemme (eller tager gennemsnittet af deres værdier).

Præcis hvorfor er en matematisk kompliceret historie, men random forest håndterer det tidligere nævnte overfitting-problem meget bedre end et enkelt beslutningstræ. Desuden har random forest mange flere håndtag at trække i (Hvor meget data har hvert træ? Hvor mange træer i skoven?), så vi har en meget naturlig måde, vi kan øge kompleksiteten på (fx ved at have flere træer).

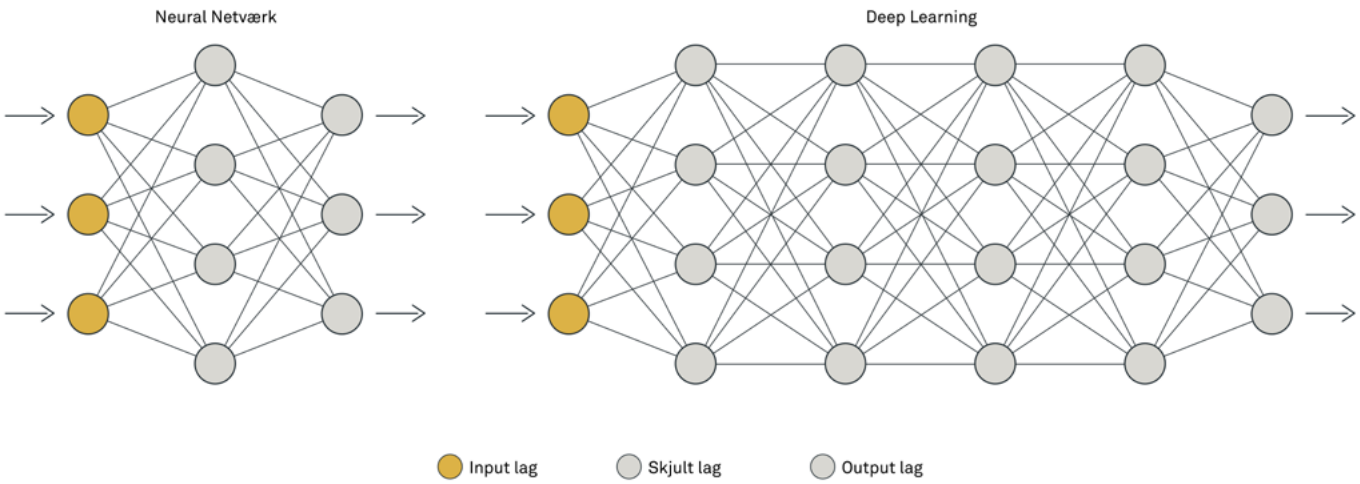
Slutteligt skal nævnes *gradient boosting*, som er en yderligere udvikling af random forest, hvor de enkelte træer deler fejl for at lære af hinanden. Det er en ekstremt effektiv metode særligt kendt for at vinde mange mindre datakonkurrencer rundt omkring på internettet.

Deep learning

Deep learning (eller dybe neurale netværker) er formentlig en af de hotteste teknikker lige nu og er drivende bag næsten alle større gennembrud inden for kunstig intelligens i de seneste år. Teknikkerne er blevet meget avancerede, og det er et helt felt for sig, hvorfor det er næsten uretfærdigt at beskrive bare én teknik.

Kort sagt fungerer deep learning ved at "stable" matematiske funktioner ovenpå hinanden med ikke-lineære transformationer imellem. Hvis ikke det lige gav mening, så tænkt på det sådan her:

Måske har du tre "inputs" til din model (højde, alder og indkomst), og du ønsker at gætte, hvilket køn folk har (mand/kvinde). I en deep learning-model som den nedenfor tager man de tre parametre, krydser dem med hinanden og transformerer dem så igennem en af fire forskellige "neuroner" (det første lag efter input). Hver neuron har en ikke-lineær transformation, så output fx bliver mellem 0 og 1, og høje og lave tal kommer tæt sammen, mens tal omkring midten lægger sig spredt ud mellem 0,2 og 0,8 (det kaldes også en "squashing function", fordi de har sådan en "presseegenskab"). De fire nye outputs kombineres herpå alle sammen med hinanden i modellens andet lag, hvorpå de transformeres igennem nye ikke-lineære funktioner, så videre til tredje lag osv. Til sidst kommer de til den sidste output-neuron. Her vil modellen være indrettet således, at et tal tæt på 1 fx betyder kvinde, mens et tal tæt på 0 betyder mand (og når vi skal bruge den senere, ville vi fx sætte et cut-off-punkt ved 0,5).



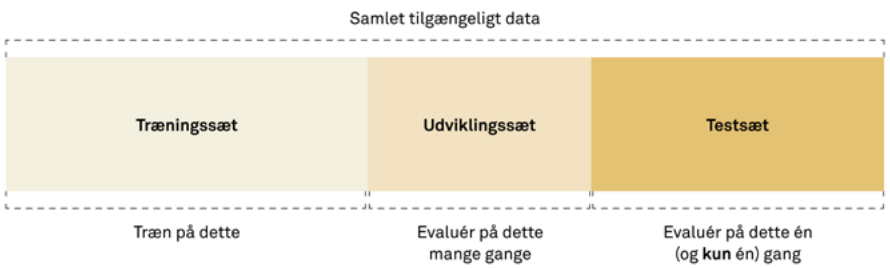
Når modellen trænes, identificeres der vægte for alle inputs mellem alle lag (så inputs fra forrige lag ikke bare naivt sættes sammen). De specifikke egenskaber for de ikke-lineære funktioner estimeres også (hvornår skal de begynde at "squashe"?). Det bliver hurtigt *meget* kompliceret, og deep learning er derfor en ekstremt kompleks metode. Fordelen er, at det kan opfange selv de mest utrolige nuancer i data (hvorfor det ofte bruges på fx sprog og billeder, som indeholder rigtig meget information og kompleksitet). Ulempen er, at de er *meget* tilbøjelige til at overfitte og derfor er *ekstremt* datakrævende.

En advarsel

Ovenfor har jeg beskrevet en proces, hvor der tages en enkelt klump data til side til test, mens der trænes på resten. Hvis det står på meget længe, og der afprøves et stort antal modeltyper, så kan selve processen med at vælge model blive drevet af ens testsæt frem-for noget reelt i data. Mange arbejder derfor med et ekstra split, et såkaldt "udviklingssæt".

Basalt set deles data i tre, fx 60% træning, 20% udvikling og 20% test.

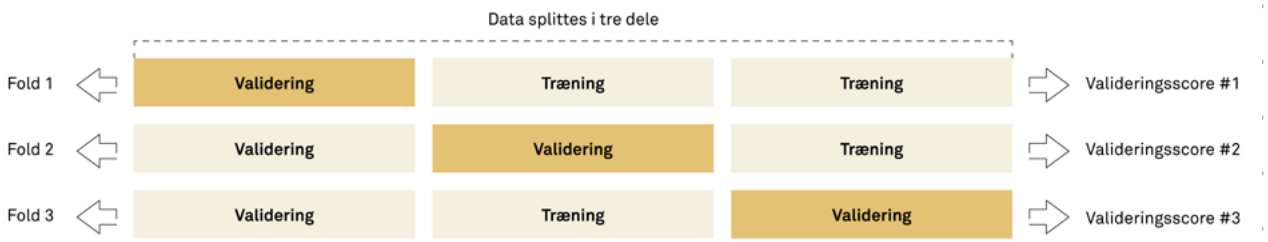
Træning bruges som tidligere til at estimere modellen. Udvikling bruges til at "teste" på til daglig og under udviklingen og til at finde den optimale model. Testsættet bruges nu kun én gang – som en endelig eksamen.



Det sikrer, at testsættet *virkelig* er ny data, og at den præcision, vi rapporterer for vores model, vil være på det endelige sæt.

Bonus: Krydsvalidering

Et alternativ til test-train-splittet, som nogen er glade for, er krydsvalidering. Her splittes data i fx tre dele (33% data i hver). Modellen estimeres og testes så én gang for hver klump data – hele tiden med en ny klump til test og alle øvrige inde som træningssæt. For hvert "fold" opnås altså en testscore, og vi beregner så et gennemsnit på tværs af alle fold som vores endelige score.



Krydsvalidering er generelt betragtet som både mere præcist og mere sikkert end et simpelt test-train-split, men det kan også være tidskrævende og kompliceret at implementere, fordi modellen skal estimeres mange gange.

Bemærk, at det ofte anbefales at splitte data i 5-10 stykker, når der laves krydsvalidering, så der foretages flere fold, og størrelsen på testsættet udgør mellem 10-20% af det samlede data hele tiden.

Kapitel 5. Etik

Inden for data science fylder etik rigtig meget for tiden og med god grund. Når vi arbejder med data science, går vi ofte meget tæt på mennesker, og vi bruger værktøjer, som vi ikke altid forstår til fulde endnu. Der er gang i en rivende udvikling, hvilket vi også kan se, når vi kigger på lovgivning fra både EU og andre aktører, der forsøger at sætte nogle rammer særligt omkring brugen af kunstig intelligens og dataindsamling.

Inden for moralfilosofi er etik den disciplin, som beskæftiger sig med spørgsmålet om, hvad der er godt og ondt eller rigtigt og forkert. Her vil jeg dog beskæftige mig specifikt med *anvendt etik,* som er spørgsmål om, hvad en person er forpligtet til (eller hvad der er tilladt) i konkrete situationer.

Jeg skal også som sådan sige, at de ting, jeg skriver her, er for egen regning. Jeg har både studeret og arbejdet aktivt med etik både generelt og inden for data science i flere år. Desuden læner jeg mig op af People and AI Research (PAIR-gruppen) og deres arbejde med People + AI Guidebook. Jeg har ikke selv bidraget med filosofiske gennembrud, og jeg kan ikke påstå at hvile systematisk på et etableret moralfilosofisk system.

Lidt populært kan man for data science opstille en liste med dos (gode ting) og dont's (dårlige ting).

Privacy

Jeg vil her gå forholdsvis let henover privacy-spørgsmål. Det er ikke fordi, det er uvæsentligt (det er faktisk rigtig vigtigt), men af hensyn til plads og fokus i guiden. Det britiske Information Commissioner's Office (ICO 2020) er et rigtig godt sted at starte. *Udover* det juridiske fra fx GDPR er privacy (ligesom resten af den etiske diskussion) en afvejning mellem den værdi, der skabes for brugerne og samfundet, og de krænkelser, der kan være associeret med at opbevare brugeres information. Mange af de samme regler gælder som for øvrig data science-etik: Vær klar omkring, hvad der gemmes, og giv brugerne mulighed for at styre, hvad der gemmes og hvor længe. Endelig er der nogle hjælpsomme teknologier, som jeg gerne vil fremhæve, bl.a. differential privacy, hvor data tilføjes en lille smule forkerte informationer (såkaldt støj). Det vil sige, at hvis data skulle komme i de forkerte hænder, kan det ikke vides, hvilken der er ægte, og hvilken der er falsk. Tilføjes de falske informationer rigtigt, vil det ikke betyde noget for data science-værdien af data.

Do (Det gode):
- Skab værdi
- Red liv
- Gør folks liv nemmere
- Gør ekspertise tilgængelig for flere mennesker
- Automatisér opgaver, og frigiv mennesker til mere værdifulde mål

Don't (Det dårlige)
- Bias (her brugt, som når modellen er "skæv" eller systematisk upræcis)
- Diskrimination (her brugt, når biasen går specifikt på problematiske karakteristika)
- Skjule/fordreje
- Lyve

Sørg for at skabe værdi

Når vi afvejer fornuften i vores data science-løsninger, skal vi huske hele billedet. Alle data science-løsninger skal sikre, at de er værdifulde (fx gennem en effektiv use case- identifikation (Kapitel 1)). Data science indeholder enormt meget potentiale, og vi har eksempler på AI, der redder liv (Corti.ai 2020), gør folks liv nemmere (Muhajlovic 2020), gør ekspert-intuition tilgængelig for flere (Seeing Potential 2020) eller automatiserer opgaver og frigiver mennesker til mere værdifulde formål (Enversion, KAUNT 2020).

AI og data science skaber – og skal skabe – en masse værdi. Hvis ens løsning ikke er værdifuld, så nytter det naturligvis ikke noget (fx DeepNude, (Vincent 2019)).

Det betyder imidlertid også, at når vi vurderer etikken i en data science-løsning, skal spørgsmålet være, om vores omkostninger står mål med den værdi, vi skaber. Er det fair at analysere folks patientjournaler for at optimere

reklamer for kosttilskud til dem? Og hvad hvis formålet er reelt at forhindre blodpropper (Enversion, Sundhed 2020)?

Langt de fleste data science-løsninger forsøger faktisk at skabe værdi, og rigtig mange lykkes også med det. Derfor er det vigtigt at blive ved og gøre det endnu bedre.

Undgå bias og overmodighed

Bias bliver et problem, når vi bringer en model i anvendelse, og den ikke virker som forventet. Det kan fx være, at ens model til at forudsige, om folk ville blive langtidsledige, var korrekt 99% af tiden, da man udviklede den, men da den begyndte at blive brugt i virkeligheden, viste den sig kun at have ret 70% af tiden.

Bias opstår typisk af to veje: (1) Vi har været uforsigtige og lavet en fejl i vores databehandling, eller (2) vores data var dårlige og passede ikke med den virkelighed, modellen skulle bruges i.

Vi kan afhjælpe punkt 1 ved at anvende en god metode. Test modellerne grundigt (Kapitel 4. Machine learning), og sørg for, at modellerne faktisk gør som forventet (Kapitel 3. Definition af formål). Andre gode teknikker er et særligt fokus på outliers og edge cases, og sørg for, at der også er en elegant håndtering af disse (dette kan inkludere fx People + AI Guidebook, kapitlet Graceful Failure). Og endelig kan færdige modeller underlægges simulatorstudier, hvor kontrafaktiske situationer opstilles for at tjekke modellernes robusthed i særlige situationer.

Punkt 2 handler om at forstå sine data. Der er ingen undskyldning for ikke at have styr på sine data (Kapitel 2. Dataeksploration).

Jeg mener således i ramme alvor, at der pålægger data scientists et *etisk* ansvar for at sikre *robuste* modeller bygget på et *klart* datagrundlag.

Undgå diskrimination

Diskrimination bruger jeg her som en betegnelse, hvor modellen har lært fx racistiske eller sexistiske mønstre. Modellen kan være teknisk velfungerende (den har lært det mønster, der var i data), men den er stadig etisk uforsvarlig, fordi de konklusioner, den drager i sig selv, er uforsvarlige.

Det er en svær situation, for som Cathy O'Neil så grundigt viser i sin bog, Weapons of Math Destruction (O'Neil 2016), så kan rens af data være svært, umuligt eller gøre ting værre. Hun ender faktisk med at konkludere, at det bedste kan være at lade diskriminationen indgå og i stedet være opmærksom på den og håndtere den rundt om modellen. Dette kræver naturligvis et *grundigt* studie af modellens diskrimination (fx med simulationer af kontrafaktiske cases).

En note på diskriminationsspørgsmålet kommer fra John Shawe-Taylor, UNESCO's (FN)'s formand for kunstig intelligens (Synced 2019):

"HUMANS DON'T REALIZE HOW BIASED THEY ARE UNTIL AI REPRODUCES THE SAME BIAS."

Inden vi raser imod en diskriminerende model, er det ofte værd at spørge, hvorfor data er så uforsvarligt diskriminerende i første omgang. Ofte kan det lade sig gøre at lave en *mindre* diskriminerende algoritme end det menneskelige alternativ. Så er det etiske spørgsmål ofte, hvilken af følgende tre løsninger der er *mindst problematisk:*

1. Forbedringen er værd at lave trods bekymringen om automatisering.

2. Den diskriminerende proces kan helt nedlægges eller omlægges radikalt.

3. Holde fast i den diskriminerende praksis, som den er.

Jeg antager i det følgende, at det bliver besluttet at forsøge at lave en fornuftig løsning baseret på algoritmer og uddyber derfor, hvordan det gøres på den etisk mest forskellige måde.

Undgå løgne, hemmeligholdelse og uklarheder

Vores model kan være nok så god, men det er meget vigtigt, hvordan vi taler om den.

Vi kan fx forestille os en data science-løsning, som behandler folks ansøgninger om erstatning hos et forsikringsselskab. Vi antager, at løsningen er forholdsvis velfungerende og gør to ting: (1) Vurderer, om folk har udfyldt deres ansøgning med tilstrækkelig information, og (2) giver dem et umiddelbart bud på deres rettigheder.

Her er tre eksempler på dårlige måder at tale om data science og et godt eksempel:

Undgå at gøre data science til sort magi

✗ *Tak for din ansøgning. Vi vil nu behandle den med vores Auto-Request-Analysis-AI™ og give svar hurtigere, end du nogensinde har fået før.*

Svaret her efterlader netop folk forvirrede og uklare på, hvad der er sket. Hvad er det for et system, jeg bliver behandlet af? Hvad hvis jeg ikke bryder mig om det? Hvorfor sker det? Det er uhjælpsom marketing-hype.

Undgå at skjule modellen

✗ *Tak for din ansøgning. Vi vil behandle den hurtigst muligt og vende tilbage til dig.*

Hvis brugerne få sekunder senere modtager en besked om, at deres ansøgning er ukomplet, virker det mærkeligt. Er den virkelig blevet behandlet? Hvorfor gik det så hurtigt? Lad være med at skjule automatiseringen, da det er en opskrift på uærlighed og forvirring.

Undgå at forklare folk teknikken

✗ *Tak for din ansøgning. Vi har for nylig implementeret en non-negative matrix faktoriseringsalgoritme, som hjælper os til at sortere indkommende ansøgninger og identificere manglende features. Vi vil med cirka 96% præcision kunne svare dig om lidt.*

Denne version er reelt lidt en afart af den første og handler om, at brugerne ender med at stå tilbage og ikke forstå systemet. Forklaringen er måske korrekt, juridisk fyldestgørende osv. Men den tiltaler kun de 1% mest tekniske, kompetente og krævende brugere og misser de 99% andre.

Sig, hvad der sker, hvorfor det sker, og hvad der kan gøres

✓ *Tak for din ansøgning. Vi scanner nu din ansøgning automatisk for formelle mangler, og, hvis muligt, giver vi dig et umiddelbart bud på en afgørelse. Hvis nogle af delene giver anledning til spørgsmål eller uddybelser, kan du svare på denne mail. Ellers vender en af vores ansatte tilbage til dig snarest muligt.*

Vær ærlig om, at du benytter et automatiseret system. Det er sjældent vigtigt for folk, om det er machine learning, et sæt simple regler eller en robot. Sørg for, at det er klart, hvad analysen går ud på (her: mangler og umiddelbar afgørelse), og lige så vigtigt: Hvad beror analysen på (ansøgningen)? Endnu bedre er det også at give brugeren en mulighed for at kommunikere i anledning af automatiseringen. De fleste data science-løsninger laver fejl ind imellem, og det giver brugeren plads til at hjælpe med at håndtere dem foruden en chance for at reagere på resultaterne i øvrigt.

Øvrige pointer om dialogen om en model

Jeg tror, at rigtig mange af de etiske udfordringer i forhold til data science, AI og machine learning løses med UX eller *user experience design*, hvis man sørger for, at brugerne/borgerne/kunderne forstår, hvornår der benyttes data science-løsninger, hvorfor det gøres, og hvad der er fordelen ved det. Hvis der gives mulighed for at give feedback og håndtere det, når systemerne fejler. Hvis der gives forklaringer på det rigtige niveau.

(Kapitlerne Feedback + Control, Errors + Graceful Failure og Explainability + Trust i People + AI Guidebook)

Mange af disse tiltag er ikke nye for erfarne UX-designere. Men data scientists er ikke vant til at tænke i det og bliver ofte konfronteret med dilemmaerne og forventes at kunne svare.

Som referencerne i denne guide antyder, anbefaler jeg på det kraftigste People + AI Guidebook som et sted at få en dybere indsigt i, hvordan man sikrer data science-løsninger, der arbejder godt sammen med mennesker.

Kapitel 6.
Deployment og governance

Før en data science-løsning kan skabe værdi, skal den *bruges*. Det betyder i langt de fleste tilfælde, at løsningen skal bringes i *produktion*, således at den enten kan bruges løbende, når der er behov, eller via fx daglige kørsler på partier af data.

Deployment og Governance

Deployment er den specifikke handling at sætte noget (fx en data science-løsning) i produktion. Det vil sige at gøre den tilgængelig for enten de interne eller eksterne brugere, som skal drage værdi af den. Selve handlingen er ofte så nem som at trykke på en knap. Når jeg alligevel fremhæver den her, er det fordi, at det netop er dét tryk på knappen, der markerer skiftet fra udvikling til gevinstrealisering.

Governance er alle processerne og aktørerne op til og omkring deployment, vedligeholdelse og drift. Hvem har lov til at sende en ny løsning i produktion? Hvad er de krav, de skal leve op til? Governance drejer sig også om de processer, der fx sikrer, at fejl kan spores tilbage til deres kilder.

Hvorfor data science governance?

Governance har tre formål: fremtidig udvikling, vedligeholdelse og sikkerhed.

Fremtidig udvikling: De fleste organisationer har et it-landskab tilstrækkeligt kompliceret til, at alle deployments påvirker eller afhænger af mindst en håndfuld andre systemer (og tro mig, en deployment kommer sjældent alene. Du kommer til at ville opdatere din model!). Governance sikrer bl.a., at de eksisterende systemer er kendte, og eventuelle rettelser til dem kan ske så smidigt som muligt.

Vedligeholdelse: Der vil ske fejl. Spørgsmålet er, hvor alvorlige, hvor ofte og hvor svære de bliver at rette. En god governancemodel sikrer en høj kvalitet via test, og når der sker fejl, så kan de spores tilbage til deres kilder.

Sikkerhed: Governance er også den struktur, der sikrer, at der er en ensartet og gennemtænkt beskyttelse af fx data og modeller.

En organisation med god governance vil altså opleve, at de har en smidig udviklings- og deploymentproces, at de har nemt ved at drifte deres løsninger og har en stærk sikkerhedsprofil.

Bad governance, good governance

Alligevel klinger *governance* bureaukratisk og tungt for de fleste. Inden for governance generelt er det værd at tale om *good* governance og *bad* governance.

Grundlæggende handler good governance om at have et **effektivt, retfærdigt** og **strategisk** system.

Et **effektivt system** opfylder sine mål (fremtidig udvikling, vedligeholdelse og sikkerhed) med den *mindst mulige omkostning* både i form af tid og ressourcer, men også kompleksitet og bureaukrati. Der skal fx være færrest mulige aktører inde over hver beslutning (der stadig sikrer, at kvaliteten er høj, og ansvaret er korrekt fordelt).

Et **retfærdigt system** er *proportionelt*, dvs. at regler og kontrol følger vigtigheden af et system. Desuden er det *konsistent*, dvs. at ensartede handlinger og problemer behandles ens. Endelig skal systemet være **strategisk** og sørge for at understøtte data/ virksomhedsstrategiens overordnede mål.

Og husk: Fravær af governance er også governance.

Rammerne for data science governance

Lidt forenklet drejer data science governance sig om at tage nogle discipliner og forholde sig systematisk til dem på tværs af de tre væsentligste områder i en data science-løsning: **data, model** og **kode**.

Jeg antager her, at det infrastrukturmæssige er på plads (altså servere og anden hardware) og springer indtil videre over det organisatoriske (kan brugerne finde ud af at bruge vores model? Skal de have træning?).

De præcise discipliner, der kræver fokus, kan variere fra organisation til organisation, men et overordnet sted at starte kunne være:

- **Ejerskab og ansvar:** Hvem bestemmer (og derfor har ansvaret) over hvilke dele af en løsning?
- **Processer:** Hvordan behandler vi dette område? Hvad er vores procedurer? Hvad er best practice? Har vi en operating model?
- **Kvalitet:** Hvad er kvalitetskriterierne, før vi deployer? En given præcision? Tolerance for falske positiver? En grad af gennemtestning?
- **Sikkerhed:** Hvem har adgang til hvad? Monitorerer vi for mistænkelig brug af vores løsning? Er der potentielle sikkerhedshuller i vores kode?

Kort sagt er målet en matrixstruktur som denne:

Hvordan laver vi så data science governance?

Data science governance er i virkeligheden en række spørgsmål og svarene dertil.

Nedenfor vil jeg gennemgå de tre områder og komme med nogle af de konkrete spørgsmål, som organisationer, der har planer om at bringe data science-løsninger ind i deployment, skal kunne svare på.

Data science governance: Dataperspektivet

Data Governance (Knight 2017) er et stort felt for sig. **Ejerskab og ansvar** er ekstremt vigtigt, og ofte bruges en formel rolle til at holde ansvaret for en datakilde, en såkaldt "data steward". Jeg kan *kun* støtte op om den slags koncepter. **Data lineage** (på dansk: dataafstamning) kendes også fra almindelig data governance, men bliver særligt vigtigt i data science, fordi vi ofte flytter data meget rundt. Ideelt set gør vi notat af alle transformationer med, hvornår det skete, hvad transformationen hed, hvor mange data der kom ind, og hvor mange data der kom ud.

Ved centrale trin i data science-processen kan der også med fordel sættes automatisk rapportering op. Jeg skriver også om det under kapitlet om dataeksploration, men kort sagt kan et spor af dataprofiler, der beskriver, hvilke variable der findes og fx deres fordelinger og centrale tendenser, være en stor hjælp i at kunne gå tilbage trin for trin og identificere problemer i et dataflow.

Der bør gøres brug af **automatiske tests**, som fx sammenligner rapporter på tværs af trin og slår alarm, hvis de varierer på uforudsete måder.

Endelig er der datasikkerhed, som håndteres med fx korrekte adgange og opmærksomhed hos de respektive data stewards.

Nøglespørgsmål:

- Hvem har ansvar og ejerskab for denne data?
- Hvilke kilder består disse data af?
- Hvilke transformationer har disse data været igennem?
- Er nogle data blevet fjernet? Hvilket? Hvor meget? Hvorfor?
- Hvem har adgang til data?
- Indsamler vi mere data, end vi har brug for?

Data science governance: Modelperspektivet

Ligesom data bør modeller have en **ejer** (i agile ofte en product owner). Modeller bør underkastes **automatiske tests**, som både dækker deres generelle performance (Er modellen bedre end baseline?) og dens specifikke performance (Håndterer modellen nogle af de vigtigste særtilfælde korrekt?), og desuden at den generelt fungerer (hvad gør den, hvis den får nonsens-input?).

Ligesom data bør der eksistere **model lineage**. Hvilken data var modellen trænet på? Hvilke indstillinger var brugt under træning? Hvornår blev modellen trænet?

En model bør **monitoreres**. Både input og output. Over tid er der risiko for, at verden ændrer sig. En model, der fx bruger lønniveau i kroner som et parameter, vil skulle opdateres løbende med inflationen.

Endelig bør der monitoreres for målrettede forsøg på at snyde og misbruge modellen. Dette kan ofte også gøres via input-monitorering. Her kan man fx være opmærksom på adversial attacks (Goodfellow 2017).

Nøglespørgsmål:

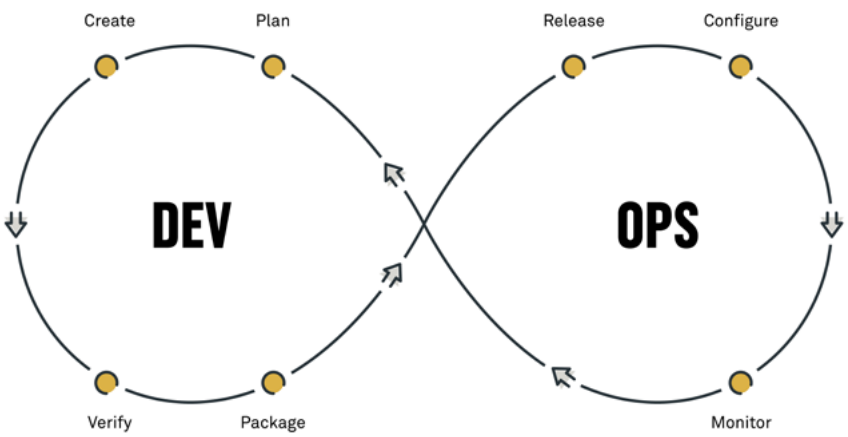
- Hvem ejer modellen?
- Hvad er vores forventninger til modellens generelle performance?
- Hvad er de vigtigste særtilfælde, som modellen skal håndtere? Gør den det?
- Hvordan er modellen blevet trænet? På hvilken data?
- Hvordan monitorerer vi input og output af modellen? Hvor stor er vores tolerance for variation i data?
- Hvordan opdager vi snyd og misbrug?

Data science governance: Kodeperspektivet

Når vi laver data science, skriver vi ofte ganske almindelig kode. Governance-modeller inden for softwareudvikling er et meget modent felt, og jeg vil anbefale at læse op på fx DevOps (Loukides 2012).

I et moderne udviklingsmiljø bør data science-løsningen indgå i en Continuous Integration/ Continuous Development (CI/CD) pipeline, der inkluderer unit testing af enkelte komponenter, sammen med en Continuous Integration test for samlingen af komponenter fra dataindlæsning til forudsigelse. Værktøjer som GitLab, Jenkins eller CircleCI gør den slags nemt og kan fx gøre succesfuld gennemførelse af automatiske tests til en forudsætning for deployment (mere om alle disse ting i næste kapitel).

Et tilsvarende vigtigt aspekt af at sikre kvalitet i data science er kodereviews, hvor nye tilføjelser eller ændringer til en løsning skal godkendes af en anden udvikler, inden de bringes i produktion. På små løsninger kan det virke unødigt bureaukratisk, men værdien bliver hurtigt klar: Læringspotentialet er stort, og det fanger mange af de mest banale fejl.



Endelig er det meste kode en potentiel sikkerhedsrisiko. Cyberangreb starter ved at identificere svagheder i koden, og det kan der også fx sikres bedre imod via kodereviews.

Nøglespørgsmål:

- Har vi en fælles udviklingsmetode (fx DevOps, Scrum, SAFe)?
- Hvad er vores forventninger til unit testing på de enkelte komponenter?
- Har vi en CI-test? Er den automatiseret og en forudsætning for deployment?
- Hvad er vores praksis for kodereviews?
- Hvordan håndterer vi sikkerhedshuller i koden?

Opsummerende

Deployment er skridtet, hvor vi begynder at omsætte al vores data science til værdi, og governance er systemet, der sikrer, at vi kan lave deployments nemt og ofte.

God governance sikrer ansvar, processer, kvalitet og sikkerhed på tværs af data, modeller og kode og gør det med mindst mulig kompleksitet, størst mulig konsistens og med et blik for at opfylde de strategiske mål for virksomheden.

Hvis du kan svare på nedenstående spørgsmål, så er du godt på vej.

	EJERSKAB OG ANSVAR	PROCESSER	KVALITET	SIKKERHED
Data 	Hvem ejer data, og hvem har ansvaret for det? Hvem er data steward? Hvad er den officielle definition af dataområdet?	Automatisk rapportering. Automatisk logning af input og output ved alle væsentlige transformationer. Hvilke data fjernes hvor?	Hvor kommerdata fra? Hvilke transformationer har det gennemgået? Ser data ud, som det plejer på dette tidspunkt i processen?	Hvem har adgang til data? Hvad gemmer vi? Hvor længe? Og hvor? Hvordan gemmes det?
Model 	Hvem ejer modellen, og hvem har ansvaret for den?	Hvordan performer modellen generelt? Hvordan performer modellen specifikt på edge cases?	Hvor kommer modellen fra? Hvad var indstillingerne, da den blev trænet? Hvilke data var modellen trænet på?	Overvåger vi modellens brug for mistænkelig adfærd? Har vi testet den for robusthed? Er modellens svagheder lige for alle brugere?
Code 	Hvem ejer denne del af koden, og hvem har ansvaret for den (eventuelt product owner og scrum master)?	Automatisk unit testing af de enkelte funktioner. CI/CD (Continuous Integration/Continuous Deployment) af den samlede løsning.	Hvad er vores praksis for kode-review? Hvem kan godkende nye deployments? Følger vi DevOps eller en anden formel metode?	Har vi overblik over vores risiko? Har vi kendte huller i vores kode? Er de prioriteret?

Kapitel 8.
Governanceværktøjer

I forrige kapitel beskrev jeg forholdsvist omfattende de overvejelser, man bør gøre sig, når man starter på at udvikle sin governancemodel for data science. Governance er i høj grad en organisatorisk og menneskelig øvelse, der kræver noget så banalt som møder, aftaler, tillid, mandat og legitimitet.

Men der er også en række virkelig hjælpsomme værktøjer.

Jeg vil her gennemgå en håndfuld af mine favoritter. Mange af dem har nærliggende alternativer. Jeg kan varmt anbefale alle værktøjerne, og hvor mellemstore projekter bør have dem alle (eller lignende), kan små projekter måske nøjes med at udvælge enkelte.

Værktøjerne:

1. Git
2. Metadata store
3. Logning
4. Automatiske tests
5. Data profiling

Git

Git er svaret på alle spørgsmål om review og sporbarhed i kode. Git er det mest udbredte *version control*-system i verden. Det er en måde at have et centralt repository til al sin kode, som alle udviklere tilgår. Et repository er ligesom et drev med filer, men et *git repository* har *branches* og vigtigst af alt en *master branch*, som er den vigtigste og primære version af al kode. Enkelte udviklere kan oprette grene eller såkaldte *branches*, hvor de kopierer *master* ud og kan udføre deres arbejde. På den måde påvirker den enkelte udvikler aldrig master-versionen. Udviklere kan gemme (*committe*) deres arbejde løbende og tage notater af, hvorfor de laver deres rettelser. Når udviklingen af en ny funktion er færdig, kan en udvikler søge om en *pull request*. Så vil der typisk afvikles en serie af automatiske tests (mere om det nedenfor), og en seniorudvikler/ data scientist blive bedt om at *reviewe* koden. Hvis testene bliver bestået, og reviewen bliver godkendt, kan rettelser føres ind i masterversionen og blive "deployet" i næste omgang.

Dette flow:

1. Ny branch baseret på master
2. Udvikling
3. Søge om pull request
4. Teste
5. Reviewe
6. Integrere i master
7. Deployment

... er rytmen, der ofte ses i velfungerende agile projekter, og i praksis kan det være mere rodet, men det er idealet, der bør stræbes efter, og det er i høj grad Git, der er det teknologiske fundament for at få det til at virke.

I praksis bruger jeg oftest GitHub, men gode alternativer er Azure DevOps og GitLab.

Metadata store

En metadata store er en database, som indeholder data om de modeller, vi bygger og bruger, og den data, vi baserer modellerne på. Metadata store er svaret på alle spørgsmål om: Hvor kommer modellen fra? Hvad er der sket med data? Hvor mange data er blevet sorteret fra?

I kapitlet om governance kaldte jeg flere steder på at notere informationer ned om transformationer af data og af modellers egenskaber og performance på test og træningsdata (foruden hvilken test og træningsdata de blev trænet på). Men hvor? Og hvordan?

Tensorflow Extended er formentlig det mest modne svar, og Spotifys rejse (Spotify_Labs 2019) giver lidt indsigt i hvordan.

Ønsker man ikke at læne sig op af Tensorflow Extended, kan mindre også gøre det. Enhver database, som fx open source MySQL, kan bruges som en metadata store. En tabel til modeller med et ID, et tidsstempel, de forskellige parametre for modellen og performance på test og træning er et godt sted at starte. Dertil bør komme en tabel til data- transformationer, hvor der er kolonner med ID for kørsel, ID for trinnet i kørslen, navn på transformationen, tidsstempel, hvor mange datapunkter der kom ind, og hvor mange datapunkter der kom ud af transformationen. Eventuelt kan et trin knyttes til en dataprofil, som gemmes på en disk for sig (mere om dataprofiler til sidst).

Logning

Logning (Ajitsaria 2018) findes oftest som en log-fil i forbindelse med en kørsel. En logfil indeholder linje for linje med tidsstempler, hvad et program gjorde, og eventuelle fejl, det har mødt undervejs. Logning kommer indbygget i både Python og R, og der findes også flere tredjepartsloggere (fx loguru).

Logning er, hvad der tillader dig at finde ud af, hvad der er sket op til en fejl. Det er det første sted at slå op, hvis en model er gået i stå eller har lavet en fejl.

Logning-eksempel:

2020-04-16 16:35:11.997 | INFO | __main__:<module>:12 - Starting...

2020-04-16 16:35:11.997 | INFO | load_data:get_data_with_cache:13 - Loading from cache

2020-04-16 16:35:12.018 | INFO | load_data:get_data_with_cache:19 - Shape of dataframe: (17829, 6)

2020-04-16 16:35:12.051 | INFO | __main__:<module>:23 - Main input is: progress_to_plan

2020-04-16 16:35:12.076 | INFO | __main__:<module>:44 - Shape of X: (17472, 20) --- Shape of y: (17472, 1)

2020-04-16 16:35:12.159 | INFO | __main__:<module>:73 - All incomming variables: [...]

2020-04-16 16:35:12.162 | INFO | utils:__init__:41 - Constructing Data-object:

Automatisk test

Tests handler om at opspore fejl, før de indtræffer. Automatiske tests har den charme, at de kan bruges ofte og uden omkostninger. Generelt findes der to typer tests: **unit tests**, som tester en specifik funktionalitet, og en **Continuous Integration test**, som tester et system fra start til slut.

```python
import pytest
def test_pythagoras():
    # Funktion til at teste funktionen "pythagores()"
    a = 2
    b = 3
    forventet_c = 3.6
    assert pythagoras(a, b) == forventet_c, "Testen fejlede"
```

Unit tests er bygget ind i Python og R og understøttes også af de fleste udviklingsværktøjer (som PyCharm og VS Code) foruden diverse Git-værktøjer (som GitHub og GitLab). De er små stumper kode, der fx kunne se sådan her ud:

Ideelt set bør alle funktioner, der udvikles og bringes i produktion, have unit tests. Unit tests sikrer, at når andre udviklere eventuelt ændrer dele af koden, så bliver funktionerne ved med at give de forventede resultater.

Unit tests kan også understøtte dokumentation og give eksempler på forventet input og output af funktioner til nye udviklere.

Unit tests lever i deres egne testfiler og kan sættes op til at generere automatiske rapporter. De kan opsættes til at være forudsætninger for at oprette *pull requests* i Git og fungere som en form for automatisk governance.

Jeg er selv stor fan af PyTest (Krekel 2020), men der er også andre gode værktøjer som Robot og det indbyggede unit test.

Continuous Integration tests (CI-test) er større tests, som tester en data science-løsning fra start til slut. Typisk har CI-testen adgang til et sæt eksempeldata, som der køres igennem alle transformationer én efter én. CI-testen vil trække data ud, foretage alle de nødvendige transformationer én for én, inden der trænes en model, testes og trækkes et sæt resultater ud. CI-testen skal ses som en form for generalprøve inden deployment og sikrer, at alle de enkelte dele også virker sammen som en helhed. En CI-test er ikke fokuseret på kvaliteten af outputtet af modellen (det bør ske under udviklingen af modellen i test-train-fasen), men alene på spørgsmålet: *Virker modellen?*

Dataprofiler

Dataprofiler er rapporter over et datasæt. Antallet af variable og observationer. Antallet af manglende datapunkter og forskellige egenskaber ved de enkelte variable. Jeg nævnte også dataprofiler i kapitlet om dataeksploration. Tricket er at gøre skabelsen af disse dataprofiler automatisk enten via et redskab som fx Pandas profiling, at smide datafiler, som kan visualiseres i et PowerBI dashboard, eller via et redskab som Google Facets.

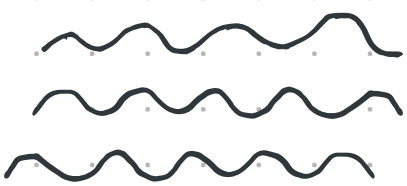
Dataprofiler kan være omkostningsfulde at beregne for store datasæt og bør derfor bruges mere sparsomt. Det kan fx være ved starten og slutningen af et komplet dataflow eller ved de mest vigtige datatransformationer. Dataprofiler kan med fordel knyttes ind i metadata store via ID'er, men bør gemmes som fx HTML-rapporter, så de kan åbnes og deles senere.

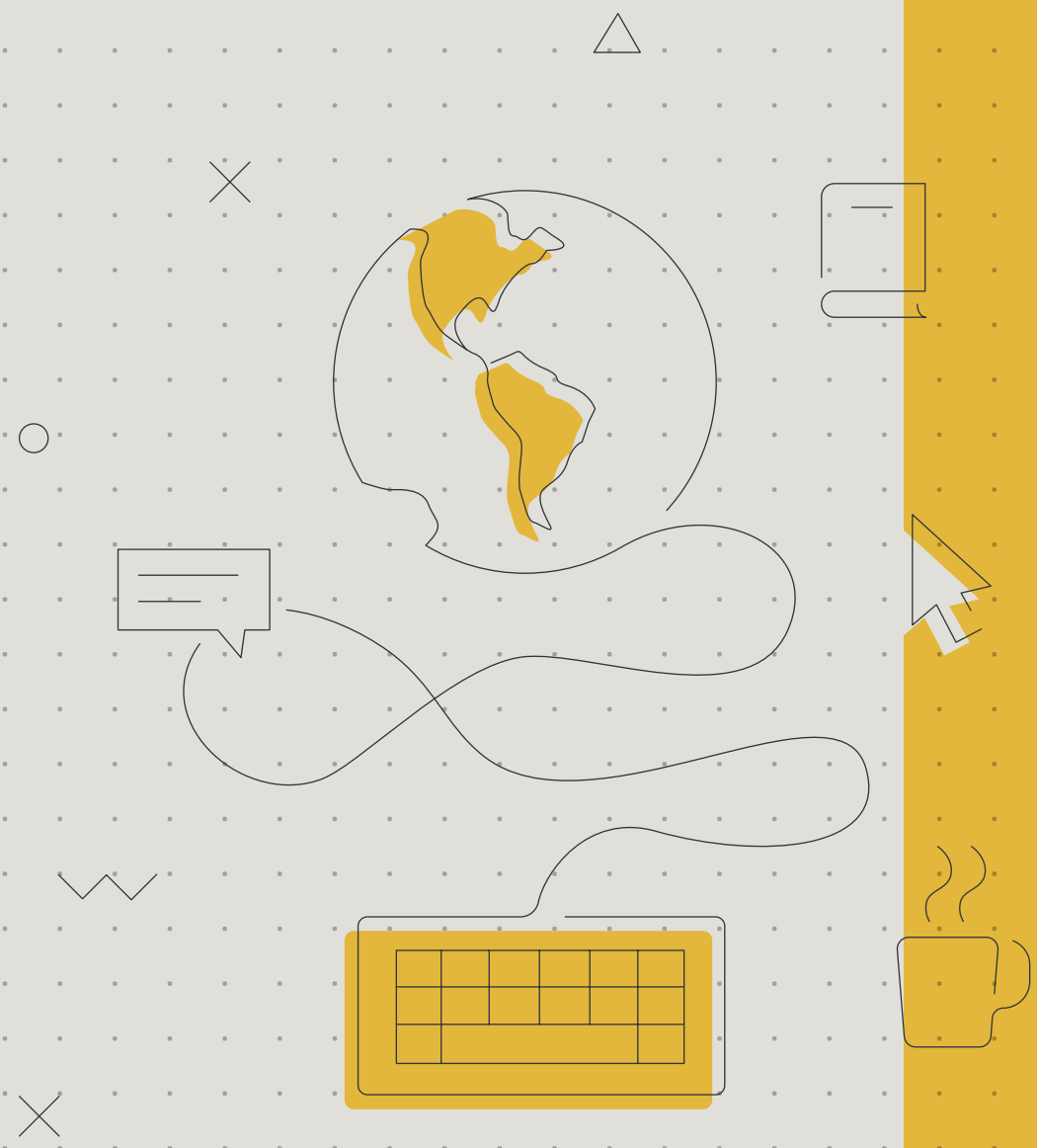
Data science for dig

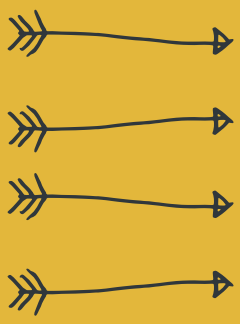
Jeg håber, at gennemgangen her har hjulpet til at give et overblik over data science- processen – ikke bare fra data til model, men fra problem til løsning. Som jeg også startede med, så er data science en kombineret disciplin af både programmering, statistik og domæneviden, og ingen af de tre dele må blive glemt. Guiden her har givet et overblik og en start, og jeg håber, at du er klar til at gøre dig dine egne erfaringer, og du kan formentlig med fordel vende tilbage til nogle af emnerne senere på din egen data science-rejse.

Slutteligt er her en liste med referencer til forskellige ressourcer nævnt i guiden. Det er både artikler og bøger, men måske endnu vigtigere en samlet liste af alle de softwareværktøjer, der er blevet nævnt.

Skulle du have nogle tanker, spørgsmål, ønsker, rettelser eller andet, så tøv ikke med at række ud! Jeg vil enormt gerne høre fra dig og hjælpe, hvor jeg kan.







Referencer

Wikipedia. 2020. Wikipedia: Data Science. 21 April. https://en.wikipedia.org/wiki/Data_science.

Corti.ai. 2020. Corti.ai. https://www.corti.ai/.

Hede, Adam. 2020. Implement Learning Institute, Data Science Master Class. April. https://learninginstitute.implement.dk/courses/data-science-masterclass.

scikit-learn. 2020. scikit-learn.org/stable. https://scikit-learn.org/stable/.

W3Schools. 2020. SQL LEFT JOIN keyword. https://www.w3schools.com/sql/sql_join_left.asp.

Grolemund, Garrett, og Hadley Wickham. 2020. R for Data Science. https://r4ds.had.co.nz/tidy-data.html.

Ascombe, F. J. 1973. »Graphs in Statistical Analysis.« The American Statistician 27:1, 17-21.

Matejka, Justin, og George Fitzmaurice. 2020. »Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing.« Autodesk Research.

Google. 2020. Facets. https://pair-code.github.io/facets/.

Wattenberg, Martin, Fernanda Viegas, og Johnson Ian. 2016. »How to use t-SNE effectively.« Distill.

Microsoft. 2020. PowerBI. https://powerbi.microsoft.com/.

Tensorflow, Google. 2020. Google Tensorflow. https://projector.tensorflow.org/.

Profilling, Pandas. 2020. Pandas profilling. https://github.com/pandas-profiling/pandas-profiling.

PAIR_Research. 2020. People + AI Guidebook. https://pair.withgoogle.com/guidebook/.

ICO. 2020. Information Commissioners Office. https://ico.org.uk/.

Wikipedia. 2020. Differential Privacy. https://en.wikipedia.org/wiki/Differential_privacy.

Muhajlovic, Illuja. 2020. Towards Data Science. https://towardsdatascience.com/how-artificial-intelligence-is-impacting-our-everyday-lives-eae-3b63379e1.

Seeing Potential, Google. 2020. Seeing Potential. https://about.google/stories/seeingpotential/.

Enversion. 2020. KAUNT. http://enversion.dk/okonomi/.

Vincent, James. 2019. The Verge. https://www.theverge.com/2019/6/27/18760896/deepfake-nude-ai-app-women-deepnude-non-consensual-pornography.

Enversion. 2020. Sundhed. http://enversion.dk/sundhed/.

O'Neil, Cathy. 2016. Weapons of Math Destruction. Crown Books.

Synced. 2019. Humans Don't Realize How Biased They Are Until AI Reproduces the Same Bias, Says UNESCO AI Chair. https://medium.com/syncedreview/humans-dont-realize-how-biased-they-are-until-ai-reproduces-the-same-bias-says-unesco-ai-chair-9968bb1f5da8.

Knight, Michelle. 2017. Dataversity: What is Data Governance? https://www.dataversity.net/what-is-data-governance/.

Goodfellow, Ian. 2017. Adversarial Example Research. https://openai.com/blog/adversarial-example-research/.

Loukides, Mike. 2012. O'Reilly, What is DevOps? http://radar.oreilly.com/2012/06/what-is-devops.html.

Git. 2020. Git. https://git-scm.com/.

GitLab. 2020. GitLab. https://about.gitlab.com/.

Jenkins. 2020. Jenkins. https://jenkins.io/.

CircleCI. 2020. CircleCI. https://circleci.com/.

GitHub. 2020. GitHub. https://github.com/.

Azure. 2020. Azure DevOps. https://azure.microsoft.com/da-dk/services/devops/.

TensorFlow. 2020. TensorFlow Extended. https://www.tensorflow.org/tfx.

Spotify_Labs. 2019. The Winding Road to Better Machine Learning Infrastructure Through Tensorflow Extended and Kubeflow. https://labs.spotify.com/2019/12/13/the-winding-road-to-better-machine-learning-infrastructure-through-tensorflow-extended-and-kubeflow/.

MySQL. 2020. MySQL. https://www.mysql.com/.

Python. 2020. Python.org. https://www.python.org/.

Project, R. 2020. The R Project for Statistical Computing. https://www.r-project.org/.

Ajitsaria, Abhinav. 2018. Logging in Python. https://realpython.com/python-logging/.

Delgan. 2020. Loguru. https://github.com/Delgan/loguru.

JetBrain. 2020. PyCharm. https://www.jetbrains.com/pycharm/.

Microsoft. 2020. Visual Studio Code. https://code.visualstudio.com/.

Krekel, Holger. 2020. PyTest. https://docs.pytest.org/en/latest/.

Robot. 2020. Robot. https://robotframework.org/.

Foundation, Anaconda. 2020. Anaconda. https://www.anaconda.com/.

Pandas. 2020. Pandas. https://pandas.pydata.org/.

Matplotlib. 2020. Matplotlib. https://matplotlib.org/.

Kontakt

Du kan få fl ere oplysninger ved at kontakte:

Adam Hede
Implement Consulting Group
+45 2929 9395
adhe@implement.dk