Downloaded from orbit.dtu.dk on: Dec 18, 2017

Technical University of Denmark

Semantiske analysemodeller

Med fokus på brugergenereret data

Lange, Jens Christian; Petersen, Michael Kai; Larsen, Jakob Eg

Publication date:
2011

Document Version
Også kaldet Forlagets PDF

Link back to DTU Orbit

Citation (APA):
Lange, J. C., Petersen, M. K., & Larsen, J. E. (2011). Semantiske analysemodeller: Med fokus på brugergenereret data. (IMM-M.Sc.-2011-26).

DTU Library
Technical Information Center of Denmark

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Semantiske analysemodeller

Med fokus på brugergenereret data

Jens Christian Lange (s052804)
29 april, 2011

Institut for Informatik og Matematisk Modellering, IMM.
Danmarks Tekniske Universitet, DTU.
Vejledt af Michael Kai Petersen og Jakob Eg Larsen.

Resumé

Denne afhandling beskriver fire forskellige semantiske analysemodeller. De fire modeller er udvalgt på baggrund af, at de varierer i kompleksitet og benytter forskellige teknikker til at analysere data. Modellerne leverer forskellige former for resultater fra deres analyser, hvilket gør at de har forskellige anvendelsesmuligheder.

Der er blevet brugt en god del tid og indsats på, at udvikle en applikation der kan indsamle mobil kontekstdata med det formål, at bestemme brugerens sindstilstand ud fra disse. Denne applikation bliver præsenteret som en case for afhandlingen. Design- og implementeringsprocessen for applikationen beskrives, opfulgt af en analyse af de indsamlede data, baseret på den simpleste analysemodel, der er blevet foretaget løbende på mobiltelefonen under indsamlingen.

Afsluttende vil de fire analysemodellers anvendelse blive demonstreret samt diskuteret. Modellerne vil blive anvendt på forskellige former for brugergenereret data som filmanmeldelser, indsamlede SMS beskeder og opgaver og afhandlinger udført af britiske studerende. Derudover vil der blive givet eksempler på enkelte metoders anvendelse på Wikipedia og Twitter.

På baggrund af diskussionen af de fire analysemodeller konkluderes der på deres anvendelse generelt såvel som for casen.

Forord

Denne afhandling er udarbejdet ved Institut for Informatik og Matematisk Modellering (IMM) på Danmarks Tekniske Universitet Danmark (DTU) i en delvis opfyldelse af kravene for at opnå graden af Master of Science in Engineering (M.Sc.).
Denne afhandling er resultatet af arbejde udført i perioden fra november 2010 til april 2010, med en arbejdsbyrde svarende til 30 ECTS-point. Vejlederne er Michael Kai Petersen og Jakob Eg Larsen, Institut for Informatik og Matematisk Modellering, Danmarks Tekniske Universitet i Danmark.

Jens Christian Lange (s052804)

Kongens Lyngby, april 2011

Indholdsfortegnelse

Kapitel 1

1 Introduktion

1.1 Motivation

Mængden af digital viden der er tilgængelig i dag er enorm og på internettet er der samlet flere informationer, end et enkelt menneske ville kunne opsamle på en livstid. Det er blevet dagligdag for os at have al denne informationen ved hånden både i hjemmet, på jobbet og på farten.

En stor kilde til den kumulative viden som findes på internettet er brugergeneret indhold. Hver dag bliver mere og mere information lagret og der findes snart ikke voksent menneske der ikke har brugt Google eller Wikipedia til at finde frem til ny viden.

Problemet med at der kommer så meget ny viden til, er at det er umuligt at skulle håndtere og sortere alle disse informationer manuelt. Derfor ville det være hensigtsmæssigt, hvis computere kunne læse data for os og indeksere det, så vi kan finde informationer når vi skal bruge dem. Dette er bare ikke lige så let som det lyder, da computere bestående af kredsløb og ledninger naturligvis ikke har den samme intelligens og opfattelsesevne som mennesker har.

I årtier har blandt andet denne problematik dannet grundlaget for en forskningsfelt kaldet *Natural Language Processing*, der netop har til formål at forske indenfor interaktionen mellem naturligt, menneskeligt, sprog og computere. Man taler i disse kredse om semantisk analyse, når man ønsker at simulerer en bevidsthed om betydning af digitale tekster hos computere.

På baggrund af mange års forskning indenfor feltet, er der blevet opfundet en række semantiske analysemodeller, der har til formål at vurdere indholdet af elektronisk tekst på forskellige måder. Nogle teknikker stræber mod at opdele tekster i forskellige kategorier, mens andre benyttes til at finde ud af, hvor forskellige tekster eller ord er i forhold til hinanden.

I denne afhandling vil nogle af disse semantiske analysemodeller blive diskuteret og deres effektivitet vil blive undersøgt. Der vil blive taget udgangspunkt i analyse af brugergeneret data, da disse har skabt motivation for udvælgelsen af semantiske analysemodeller som emne.

1.2 Grundlæggende principper

I dette afsnit beskrives to af de grundlæggende principper indenfor semantisk analyse. Først forklares, hvordan ordoptælling foretages og hvad det kan anvendes til. Derefter introduceres betinget frekvens distribution, hvor det ligeledes vil blive forklaret, hvordan man kan udregne betinget frekvens distribution og hvad det kan bruges til i praksis.

1.2.1 Ordoptælling

Det at tælle ord i en tekst virker umiddelbart trivielt og ligegyldigt, men ikke desto mindre danner ordoptælling basis for de fleste semantisk analysemodeller. Der vil her blive gennemgået et eksempel der illustrerer, hvordan ordoptælling kan afsløre betydelig informationer om en tekst. Eksemplet er fra *Natural Language Processing with Python s. 40-41* [**1**].

Eksemplet benytter et samling Python moduler der er en del af Natural Language Toolkit, NLTK, som vil blive beskrevet senere.

Først importeres gutenberg, der er en udvalgt samling af tekster der stammer fra Project Gutenberg [**2**]. Project Gutenberg er en enorm samling e-bøger der kan hentes gratis fra internettet.

```
>>> from nltk.corpus import gutenberg
>>> for fileid in gutenberg.fileids():
           num_chars = len(gutenberg.raw(fileid))
           num_words = len(gutenberg.words(fileid))
           num_sents = len(gutenberg.sents(fileid))
           num_vocab = len(set([w.lower() for w in
           gutenberg.words(fileid)]))
           print int(num_chars/num_words),
           int(num_words/num_sents),
           int(num_words/num_vocab),fileid
```

For hver tekst i gutenberg beregnes antallet af tegn med mellemrum, antallet af ord, antallet af sætninger og til sidst antallet af unikke ord per tekst. Når disse fakta om teksten er optalt, beregnes nogle forklarende statistikker

- Den gennemsnitlig ordlængde
- Det gennemsnitlige antal ord per sætning
- Antallet af gange hvert unikt ord optræder i teksten i gennemsnit, hvilket kaldes teksten leksikale diversitets score. Den leksikale diversitets score beskriver, hvor ofte et ord genbruges, altså grundlæggende hvor varieret teksten er skrevet.

For de 18 tekster i gutenberg ses de tre statistikker herunder, de er afrundet til nærmeste heltal for større læselighed.

```
4 21 26 austen-emma.txt
4 23 16 austen-persuasion.txt
4 23 22 austen-sense.txt
4 33 79 bible-kjv.txt
4 18 5 blake-poems.txt
4 17 14 bryant-stories.txt
4 17 12 burgess-busterbrown.txt
4 16 12 carroll-alice.txt
4 17 11 chesterton-ball.txt
4 19 11 chesterton-brown.txt
4 16 10 chesterton-thursday.txt
4 17 24 edgeworth-parents.txt
4 24 15 melville-moby_dick.txt
4 52 10 milton-paradise.txt
4 11 8 shakespeare-caesar.txt
4 12 7 shakespeare-hamlet.txt
4 12 6 shakespeare-macbeth.txt
4 35 12 whitman-leaves.txt
```

Fra den første af de tre udregnede statistikker er det interessant at se, at den gennemsnitlige ordlængde er ens for samtlige af de analyserede tekster. Det skal dog nævnes, at den i virkeligheden er 3 for teksterne, da mellemrum er medregnet.

Ud fra den gennemsnitlige sætningslængde og den leksikale diversitet kan man sige meget om tekstens form.

Tag for eksempel den første tekst af Jane Austen, hvilken har en gennemsnitlige sætninglængde for en roman og med en relativt høj leksikal diversitet kan det siges, at sproget ikke er meget avanceret, altså en let læselig roman. På samme måde er Herman Melville's Moby-Dick også en roman, men som det kan ses på den leksikale diversitet er sproget mere nuanceret og stiliseret.

I shakespeare's værker ses det tydeligt af de gennemsnitligt meget korte sætninger, at det har en egnet form til teater og skuespil, samtidig er sproget avanceret.

I Biblen benyttes der lange sætninger og samtidig findes et meget begrænset ordforråd. At visse ord gentages ofte er effektivt når der skal spredes et budskab. Inden for kognitiv psykologi kaldes fænomenet *Repitition priming*, når gentagen udsættelse for et givent stimulus får hjernen til at huske det bedre og finde det mere troværdigt. Det er det samme fænomen der udnyttes i reklamer og propaganda.

Det kan ses her, at man ved at føre simpel statistik på antallet af tegn, ord og sætninger i en tekst kan udlede meget om dennes indhold og stil. Under *Analyse* vil blive gennemgået nogle avancerede analysemodeller, der har basis i den simple ordoptælling.

1.2.2 Betinget Frekvens Distribution

I dette afsnit vil der ved hjælp af Betinget Frekvens Distribution og NLTK blive vist et eksempel på hvordan optælling af verber i en tekst kan indikere, hvilken genre teksten har.

Til dette benyttes Brown korpusset, som blev oprettet på Brown University i 1961. Korpusset består af tekst fra 500 forskellige kilder, som alle er opdelt efter genre. Dette korpus er godt til, at analysere systematiske forskelle på genre, da genreinddelingen gør det nemt at sammenholde udregnet statistik med en prædefineret genre.

Med NLTK foretages først Betinget Frekvens Distribution beregning på hele korpusset for samtlige genre. Dette vil sige, at antallet af forekomster for hvert ord i korpusset sammenkædes med hvor mange gange det optræder indenfor hver genre.

```
>>> import nltk
>>> from nltk.corpus import brown
>>> cfd = nltk.ConditionalFreqDist((genre, word) for genre
in brown.categories() for word in
brown.words(categories=genre))
```

Da vi ikke er interesserede i at se antallet af hvert ord i samtlige genre vælges, hvilke genre og ord der skal vises. Af genrerne vælges *news*, *religion*, *hobbies*, *science_fiction*, *romance*, *humor* samt ordene *can*, *could*, *may*, *might, must*, *will*.

Optællingen af de valgte ord for de specifikke genre er angivet herunder.

```
>>> genres = ['news', 'religion', 'hobbies', 'science_fiction',
'romance', 'humor']
>>> modals = ['can', 'could', 'may', 'might', 'must', ,'will']
>>>cfd.tabulate(conditions=genres, samples=modals)

                 can could  may might must will
           news   93   86   66   38   50  389
       religion   82   59   78   12   54   71
        hobbies  268   58  131   22   83  264
science_fiction   16   49    4   12    8   16
        romance   74  193   11   51   45   43
          humor   16   30    8    8    9   13
```

Det kan ses, at der er nogle af genrerne der hurtigt kan identificeres ved hjælp af forekomsten af verberne i de analyserede tekster. I *news* bruges ordet *will* og *has* betydeligt mere end nogle af de andre ord. For *hobbies* bruges *can* og *will* ofte, mens for *romance* benyttes *could* tit. Ud fra dette er det altså muligt at identificere en specifik genre baseret på hyppigheden af nogle få verber

Disse observationer siger også noget om indholdet i det forskellige genre. *News* fortæller om hvad der er sket og hvad der vil ske, hvilket afspejles i de ord der benyttes. For *hobbies* fortæller teksterne om hvad man kan opnå ved at arbejde for at det, for eksempel et flot modelskib, og hvordan det vil blive, hvis man følger instrukserne i litteraturen. I tekster fra genren *romance* går ordet *could* ofte igen, hvilket er åbenlyst fordi det appellere til romantikeren , der drømmer om hvordan deres kærlighedsliv kunne blive.

I det næste kapitel vil der blive set på nogle mere avancerede analysemetoder, hvis grundpiller alle findes i den simple ordoptælling.

Kapitel 2

2 Analyse

Dette afsnit indeholder en gennemgang af fire forskellige analysemodeller baseret på forskellige teknikker og med forskellige anvendelsesmuligheder.

Der startes med den simpleste model, hvilket er undersøgelse af en tekst med ved sammenligning med ord fra ANEW samlingen, derefter vil probabilistisk naiv Bayes dokument klassifikation gennemgås efterfulgt af den ligeledes probabilistiske generative model, Latent Dirichlet allokation. Afslutningsvis beskrives Vektormodellen Latent Semantisk Analyse, der vurderer ord og dokumenters betydning efter deres placering i det geometriske rum.

De fire modeller er valgt på baggrund af deres forskellige tekniske karakter, samt indbyrdes komplementerende anvendelsesmuligheder der vil blive illustreret i *4 Anvendelse af analysemetoder*.

Der vil der blive vekslet mellem brugen af ordene "ord" og "termer", der har den samme betydning, men brugen afspejler den originale litteratur.

2.1 ANEW

Affective Norms for English Words [**3**], ANEW, er et projekt der blev oprettet i 1999 ved Center for the Study of Emotion and Attention, CSEA, på University of Florida.

Formålet med projektet, er at give forskningsverdenen en standard metode at vurdere følelser på og producere et sæt af vurderede ord der kan benyttes indenfor studiet af følelser og opmærksomhed. ANEW samlingen består af lidt over tusind engelske ord, som alle er blevet følelsesmæssigt bedømt af en række kvindelige og mandlige testpersoner.

Hvert ord er blevet vurderet indenfor tre forudbestemte dimensioner, valens, også kaldet glæde, ophidselse og dominans. Skalaen for de to primære dimensioner valens og ophidselse går fra henholdsvis trist til glad og afslappet til ophidset. Den tredje og mindre håndgribelige dimension dominans går fra lidt dominerende til meget dominerende.

ANEW samlingen egner sig altså til en overfladisk vurdering af stemningen i et stykke tekst, hvor stemningen er vurderet ud fra den gennemsnitlige valens- og ophidselsesværdi der er tildelt ordene i samlingen.

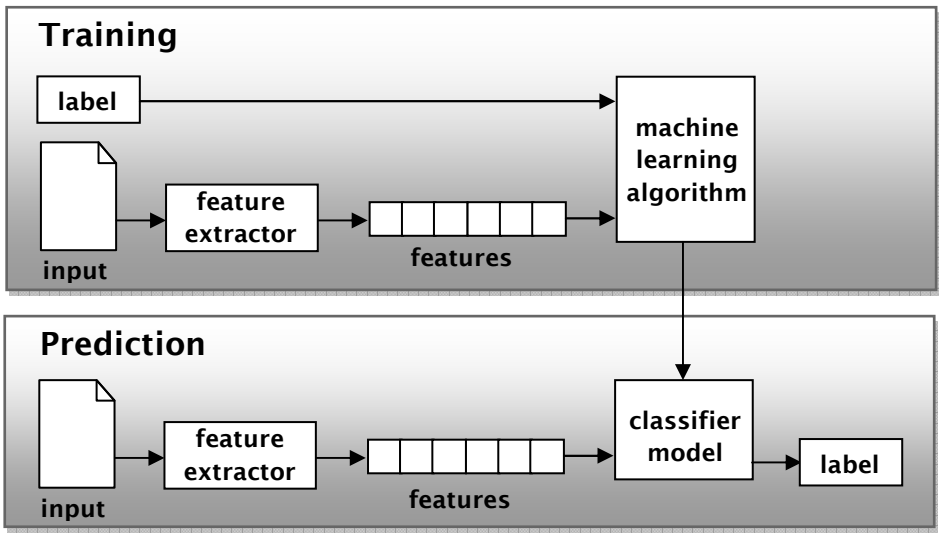
2.2 Dokument klassifikation

Dokument klassifikation er en metode og en vedvarende udfordring indenfor *Natural Language Processing*, NLP , altså behandling af naturligt eller menneskelig sprog. NLP er et forskningsfelt der beskæftiger sig med samspillet mellem computere og menneskeligt sprog. Herunder går dokument klassifikation ud på at få en algoritme til, at automatisk inddele elektroniske dokumenter i forskellige klasser eller kategorier.

Eksempler på dokument klassifikation kan være at bestemme om en given tekst er en anmeldelse af en film eller af køkkenudstyr, om det er en kriminalroman eller en sportsannonce eller om den er positiv eller negativt stemt.

Dokument klassifikation kan opdeles i to forskellige typer, nemlig overvåget klassifikation og uovervåget klassifikation.

Overvåget klassifikation er en metode til dokument klassifikation, hvor algoritmen der skal klassificere dokumenterne bliver trænet i at klassificere korrekt. Dette sker ved, at klassifikatoren bliver trænet, med nogle tekster der er givet forudbestemte labels, før den tages i brug. På den måde ved klassifikatoren også, hvilke labels der skal benyttes Selve trænings- og klassifikationsprocessen ses på *Figur 2-1*



Figur 2-1: Overvåget dokument klassificering. Under træningen benyttes en feature extractor til at udtrække alle relevante features som kan bruges til at klassificere inputtet. De relevante features bliver sammen med et passende label givet videre, til en machine learning algoritme, som feature sæt. Algoritmen benytter så de givne feature sæt til at skabe en klassifikator model. Under klassificeringen benyttes den samme feature extractor til at udtrække features som klassifikator modellen benytter til at bestemme et label.

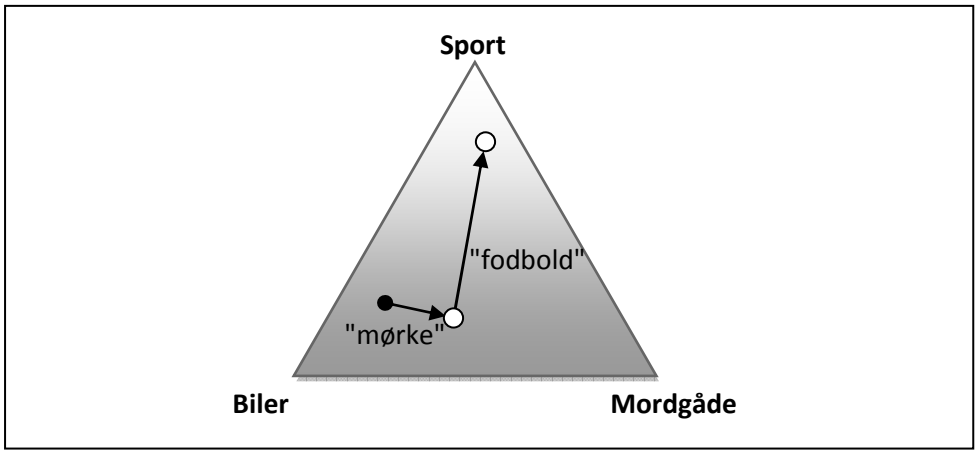
Uovervåget klassifikation er modsat overvåget klassifikation, en metode, hvor algoritmen ikke får nogen udefrakommende hjælp til bestemme, hvilke labels der er tale om eller træning i at inddele de elektroniske dokumenter i de korrekte

labels. Ved uovervåget klassifikation dannes automatisk labels baseret på det givne input og på den måde er det ikke sikkert, at der opnås de ønskede labels.

En klassisk teknik til at foretage dokument klassifikation er at bruge en Naive Bayes klassifikator.

2.2.1 Naiv Bayes klassifikator

Ved brug af naiv Bayes klassifikatoren benyttes alle features til at bestemme, hvilket label et givet input skal tildeles. Den første del af klassifikationsprocessen består i at bestemme a priori sandsynligheden for hvert label. Denne sandsynlighed er baseret på frekvensen af hvert label i det træningssæt som er benyttet til at træne klassifikatoren. Derefter tilpasses a priori sandsynligheden for hvert label, med bidraget fra hver feature der er fundet i input, for at nå frem til en a posteriori sandsynlighed for hvert label. *Figur 2-2* herunder illustrerer processen.



Figur 2-2: Dette er en simplificeret udgave af den proces der benyttes af naiv Bayes klassifikatoren. I det træningssæt der er brugt i eksemplet, handler flest dokumenter om biler, derfor starter klassifikatoren med at være tættere på Biler end de to andre labels. Så findes features i input og ordet "mørke" findes, hvilket er en svag indikator for Mordgåde, men ordet "fodbold" findes også og dette er en stærk indikator for Sport. Når alle features er behandler undersøger klassifikatoren, hvilket label den er nået tættest på og tildeler dette til input.

Hver feature der bliver fundet i input bidrager til dokument klassifikationen ved at føre klassifikatoren væk fra labels der ikke ses så hyppigt sammen med den givne feature. Dette sker ved at den samlede sandsynlighed der er udregnet for hvert label bliver ganget med en per-input sandsynligheden for at et input med samme label vil have præcis den pågældende feature. Altså hvis en given feature optræder meget få gange i sammenhæng med et label, vil sandsynligheden for at input har det label blive ganget med en lav sandsynlighed, hvilket vil reducere den samlede sandsynlighed meget. Optræder denne feature tværtimod ofte med dette label, vil den samlede sandsynlighed kun blive reduceret lidt.

2.2.2 Naiv Bayes algoritme

Fundamentet i en naiv Bayes algoritme er Bayes' Teorem som er givet ved

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{A}) \cdot P(\mathrm{B}|\mathrm{A})}{P(\mathrm{B})}$$

Der beskriver a posteriori sandsynligheden for A efter at hændelsen B er observeret, bestemt ved a priori sandsynlighederne A , B og sandsynligheden for B givet A.

Som i tilfælde af dokument klassifikation kommer til at se således ud

$$P(\mathrm{label}|\mathrm{features}) = \frac{P(\mathrm{label}) \cdot P(\mathrm{features}|\mathrm{label})}{P(\mathrm{features})}$$

Der altså giver sandsynligheden for et label givet observationen af specifikke features i et dokument. Algoritmen laver så den naive antagelse, som er årsag til algoritmens navn, at alle features forekommer uafhængigt givet et label:

$$P(\mathrm{label}|\mathrm{features}) = \frac{P(\mathrm{label}) \cdot P(\mathrm{f1}|\mathrm{label}) \cdot \ldots \cdot P(\mathrm{fn}|\mathrm{label})}{P(\mathrm{features})}$$

Denne antagelse er naiv, da det selvfølgelig ikke er realistisk, at features forekommer uafhængigt af hinanden, da de netop udgøre en del af en kontekst.

I stedet for at beregne *P*(features) eksplicit, udregnes nævneren for hvert label og derefter normaliseres de så de summere til ét:

$$P(\mathrm{label}|\mathrm{features}) = \frac{P(\mathrm{label}) \cdot P(\mathrm{f1}|\mathrm{label}) \cdot \ldots \cdot P(\mathrm{fn}|\mathrm{label})}{SUM[1](P(\mathrm{l}) \cdot P(\mathrm{f1}|\mathrm{l}) \cdot \ldots \cdot P(\mathrm{fn}|\mathrm{l})}$$

Dette er den Naive Bayes algoritme som benyttes af *Natural Language Toolkit*, NLTK, som vil blive beskrevet *i 4.2.3 Natural Language Toolkit*.

2.2.3 Konklusion på dokument klassifikation

Dokument klassifikation er en metode der effektivt kan vurdere en given tekst ud fra de features teksten indeholder. Metoden benytter sig af probabilistisk fremgangsmåde og kræver, at klassifikatoren er trænet på tilstrækkelige mængder af i forvejen klassificerede dokumenter, hvis resultatet af klassifikationen ønskes at have nogle forud specificerede labels.

2.3 Latent Dirichlet Allokering

Latent Dirichlet Allokering, LDA, er en generativ emne-model [**4**], som ved hjælp af probabilistiske antagelser gør det muligt at beskrive essensen af et dokument mere kortfattet, med et passende antal emner, deraf navnet emne-model. Hvert emne er beskrevet ved de ord fra teksten, hvis sandsynlighedsfordeling bedst dækker det givne emne.

LDA er baseret på den generative proces, altså at forekomsten af ord i et dokument er et resultat af den latente struktur, som er essensen af det ordene beskriver, som illustreret på Figur 2-3. Denne proces afspejler ikke virkeligheden, hvor det er ordene i et dokument der danner den latente struktur. Det gode ved at antage processen som generative er dog, at man ved opstille den generative proces som et Bayesiske inferens problem, kan vende pilen på figuren og udlede de latente betydninger i dokumentet ud fra de ordene.

En algoritme der benytter Latent Dirichlet Allokation foretager en probabilistisk analyse af en simpel term-dokument matrice, der indeholder frekvensen af hvert ord per dokument. Analysen er, som beskrevet herover, baseret på antagelsen om den generative proces og ud fra denne opnås en emnefordeling over termer som antages at have en Dirichlet prior, samt en dokumentfordeling over emner. Disse to fordelinger kan benyttes til at beskrive essensen ved hjælp af de fundne emner.

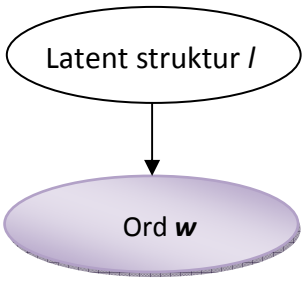
Grundkonceptet ved Latent Dirichlet Allokation som netop er blevet beskrevet beskrives mere detaljeret i det følgende afsnit.

2.3.1 Tekniske detaljer for LDA

Dette afsnit indeholder en beskrivelse af funktionaliteten bag LDA. Først beskrives et generelt eksempel på behandling af probabilistisk analyse af en generativ model og derefter undersøges emne-modellen, Latent Dirichlet Allokation.

2.3.1.1 Anskuelse af latent semantik som et statistisk problem

For at lette forståelsen af metoden der benyttes ved LDA startes der med et generelt eksempel for bestemmelse af essensen af et dokument, g, samt meningen af de ord dokumentet består af, $\boldsymbol{z} = \{z_1, z_2 \dots z_n\}$. Herunder på *Figur 2-3* ses en skematisk fremstilling af den generative model for sprog, hvor den latente struktur, l, af et dokument generer den samlingen af ord $\boldsymbol{w} = \{w_1, w_2 \dots w_n\}$ ord der udgøre dokumentet. Den latente struktur, *l*, består af essensen af dokumentet, g, og meningen med hvert ord, $\boldsymbol{z}$, så $l = (g, \boldsymbol{z})$



Figur 2-3: Skematisk repræsenteret generativ model for sprog

Den generative model fremstiller en hypotetisk årsags proces efter hvilken den givne data er blevet dannet. Den proces der er afbilledet herover, indeholder uobserveret data i form af den latente struktur, l, hvorfra dannelsen af de observerede ord, $\boldsymbol{w}$, foregår probabilistisk. Pilene indikerer retning for forhold mellem variable og disse går fra *forældre* mod deres *børn*. Pilene viser også rækkefølgen for hvilken data er genereret, så en værdi bliver valgt for hver variabel, ved at sample fra en fordeling der er betinget på *forælderen* til den variabel på grafen.

Det ses, at det modellen viser er omvendt af virkeligheden, da det er ordene i et dokument der danner den semantiske betydning og dermed essensen af dokumentet. Det gode ved denne fremstilling er, at man kan udlede sandsynlighedsfordelinger for, hvordan ordene er dannet probabilistisk ud fra den latente struktur og derefter ved at benytte statistisk inferens i form af Bayes' Teorem, kan man vende pilen på modellen. Det som sker når pilen vendes symbolsk er konkret, at man kan aflede de latente variable i strukturen som udgøre essensen af dokument, på baggrund af ordene dokumentet består af.

Først udledes sandsynlighedsfordelingerne fra den generative model, som skildrer at ord er genereret probabilistisk ved at sample en latent struktur, l, fra en fordeling over latente strukturer $P(l)$ og derefter samples en mængde ord, $\boldsymbol{w}$, betinget på denne latente struktur fra en fordelingen $P(\boldsymbol{w}|l)$.

Processen hvor hver variabel vælges baseret på en fordeling betinget på dennes *forældre* definerer en fællesfordeling over observeret data og latente strukturer, der ud fra den generative model er givet

$$P(\boldsymbol{w}, l) = P(\boldsymbol{w}|l)P(l)$$

Ved passende valg af l, kan denne fællesfordeling benyttes til at udregne sandsynligheden for l, fordelingen over meningen af ord $\boldsymbol{z}$, samt deres essens. Dette gøres ved først at benytte Bayes' Theorem, hvorfra der opnås

$$P(l|\boldsymbol{w}) = \frac{P(\boldsymbol{w}|l)P(l)}{P(\boldsymbol{w})}$$

hvor

$$P(\boldsymbol{w}) = \sum_{l} P(\boldsymbol{w}|l)P(l)$$

Ved hjælp af Bayesiske inferens, er retningen på den generative model, nu blevet vendt om og den latente struktur af et dokument kan findes ud fra de ord det indeholder. Fordelingen for meningen med ordene $\boldsymbol{z}$ og essensen, g, findes ved at summere det irrelevante aspekt af l ud

$$P(\boldsymbol{z}|\boldsymbol{w}) = \sum_{\boldsymbol{g}} P(l|\boldsymbol{w})$$

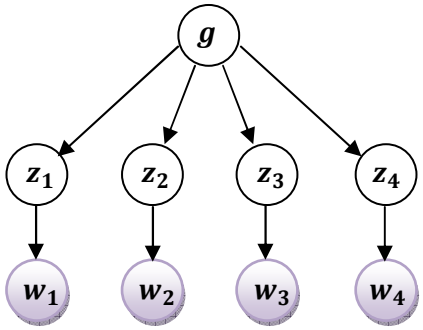
og

$$P(g|\boldsymbol{w}) = \sum_{\boldsymbol{z}} P(l|\boldsymbol{w})$$

Nu da konceptet bag probabilistisk behandlingen af latente strukturer er vist ved et generelt eksempel, vil emne-modellen Latent Dirichlet Allokation blive gennemgået.

2.3.1.2 LDA, en emne-model

En emne-model antager også en latente struktur som beskrevet i det tidligere afsnit, $l = (g, \boldsymbol{z})$. Her repræsenteres essensen, $\boldsymbol{g}$, af ordene i strukturen, som en fordeling over *T* emner, og tildeling af meningen brugt for det *i'te* ord, z_i, til et af disse emner. Dette ses afbilledet på *Figur 2-4*



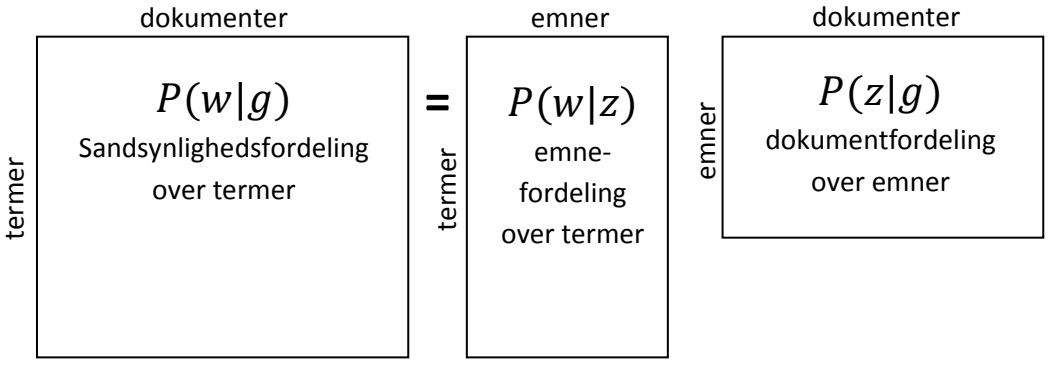
Figur 2-4: En generativ model for sprog beskrevet ved emne-modellen, Latent Dirichlet Allokation

Fra den generative model ses, hvert emne er en sandsynlighedsfordeling over ord. Ordene i et dokument bliver genereret ved at vælge den fordelingen over emner som afspejler dokumentets essens. Denne fordeling benyttes så til at vælge et emne, $\boldsymbol{z_i}$, for hvert ord, $\boldsymbol{w_i}$, og derefter generere selve ordet fra fordelingen over ord forbundet med det emne. Altså dannes emnerne ud fra essensen af dokumentet og ud fra disse emner dannes så de ord der udgøre dokumentet. Altså kan det sluttes af den generative proces, at sandsynligheden for det *i*'te ord betinget af en givet essens af dokumentet, er bestemt ved

$$P(w_i|g) = \sum_{z_i}^{T} P(w_i|z_i)P(z_i|g)$$

hvor emner bestemt af $P(w|z)$ er blandet med vægte af $P(z|g)$ som varier fra dokument til dokument. Dette er illustreret på Figur 2-5. Figuren viser også den dimensional reduktion der sker ved den Bayesiske statistiske inferens, altså når sandsynlighedsfordelingen over ord for hvert dokument betinget af essensen af dokumentet, $P(w|g)$, bliver approksimeret af en vægtet sum over en række probabilistiske emner, givet ved sandsynlighedsfordelinger over ord, $P(w|z)$, hvor vægten for hvert dokument er sandsynlighedsfordelingen over emner, $P(z|g)$, bestemt af dokumentets essens. $P(w|z)$ viser hvilke ord der er vigtige for et emne, hvor $P(z|g)$ viser forekomsten af emnerne i et dokument.

Modellen benytter en multinomial fordeling udvundet fra en Dirichlet fordeling til at bestemme essensen af dokumentet og grundet emnefordelingen antages at have en Dirichlet prior har denne emnemodel navnet Latent Dirichlet Allokation.



Figur 2-5: Emnemodellen, Latent Dirichlet Allokation. Her ses hvordan sandsynlighedsfordelingen over termer er approksimeret af emnefordelingen over termer vægtet af dokumentfordelingen over emner.

Senere vil funktionaliteten af Latent Dirichlet Allokation blive efterprøvet, for at give et konkret eksempel på metodens brug. I det følgende afsnit vil Latent Semantisk Analyse, LSA, blive beskrevet. LSA er også en analysemetode der forsøger at fange essensen af en tekst, i denne benyttes der dog ikke

probabilistiske metoder, men i stedet anvendes en vektor model for at beskrive betydningerne af ord og dokumenter i et geometrisk rum.

2.3.2 Konklusion på Latent Dirichlet Allokering

LDA er en generativ probabilistisk metode, der ved at analysere en term-dokument matrice kan bestemme de underliggende latente emner, for det korpus som er repræsenteret ved matricen. Dette gøres ved at antage en Dirichlet prior for emnefordelingen og fremstille en Bayesiske fortolkning af de betingede sandsynligheder, som er defineret ved den generative anskuelse.

LDA kan som redskab anvendes til, at simplificere store korpora ved at repræsentere dem på en let overskuelig form.

2.4 Latent Semantisk Analyse

Indenfor NLP benyttes Latent Semantisk Analyse [**5**], LSA, til at analysere dokumenter eller stykker af ustruktureret tekst for at finde latente mønstre i sammenhængen mellem forskellige termer og begreber. Ved at gøre dette kan dokumenter med fælles begrebsmæssigt indhold findes såvel som ord med fælles betydning.

LSA er baseret på princippet om at ord der ofte benyttes i den samme sammenhæng har en tendens til at have lignende betydning. Dette gør, at de termer der findes ikke altid er af den samme betydning, men også kan være af den modsatte betydning.

Det næste afsnit vil introducere Gensim, hvilket er et Python framework, der kan benyttes til at foretage Latent Semantisk Analyse på en hukommelsesvenlig måde. Med Gensim vil blive gennemgået en tutorial, som på en simpel måde viser anvendelsen af LSA.

Til sidst vil de tekniske detaljer for Latent semantisk analyse blive gennemgået.

2.4.1 Gensim

Gensim er et ikke-kommercielt Python framework, hvis formål er at gøre det nemt og ligetil at konverter tekster i naturligt sprog til Vector Space Model [**6**].

I tutorialen til Gensim gennemgås det eksempel der blev brugt i den originale artikel, hvor LSA bliver defineret [**5**]. For at give et indblik i hvordan LSA fungerer udføres tutorialen herunder med forklaring.

2.4.1.1 Gensim tutorial

For at give en praktisk introduktion til anvendelsesmulighederne for Latent Semantisk Analyse gennemføres nu et let forståeligt eksempel, hvor Gensim benyttes til at analysere ni dokumenttitler. I tutorialen gennemgås tre hovedemner

- Korpora og vektorrum
- Transformationer
- Lighedsforespørgelser

LSA går også tit under betegnelsen Latent Semantisk Indeksering, LSI, og da udvikleren af Gensim har valgt at kalde sine model der håndterer LSA for LSI vil det sidstnævnte blive benyttet gennem denne tutorial.

Først importeres gensim modulet og objekterne *corpora*, *models* og *similarities* gøres klar til brug. Derefter oprettes en liste med de ni videnskabelig titler der blev anvendt til at påvise funktionaliteten af LSA i original litteraturen. Det ses at de første fem titler er relaterede til menneske-maskine interaktion og de sidste 4 omhandler grafer.

```
>>> from gensim import corpora, models, similarities
>>> documents = ["Human machine interface for lab abc computer
applications",
"A survey of user opinion of computer system response time",
"The EPS user interface management system",
"System and human system engineering testing of EPS",
"Relation of user perceived response time to error measurement",
"The generation of random binary unordered trees",
"The intersection graph of paths in trees",
"Graph minors IV Widths of trees and well quasi ordering",
"Graph minors A survey"]
```

Disse ni titler skal nu opdeles i ord og dette gøres ved at fjerne de ord fra titlerne som kun forekommer en gang og ved at fjerne stop ord der ikke tilføjer mening til titlerne, her i eksemplet fjernes "til", "en" og "og" og så videre.[1]

Først oprettes en liste af de normale ord der skal fjernes, derefter deles titlerne op i enkelte ord og de normale ord udelades.

```
>>> stoplist = set('for a of the and to in'.split())
>>> texts = [[word for word in document.lower().split() if word
not in stoplist] for document in documents]
```

[1] "for", "a", "of", "the", "and", "to", "in"

Så oprettes der en liste af de ord der kun kan findes én gang i teksterne og ordene i denne liste fjernes så fra titlerne. Resultatet ses printet herunder.

```
>>> allTokens = sum(texts, [])
>>> tokensOnce = set(word for word in set(allTokens) if
allTokens.count(word) == 1)
>>> texts = [[word for word in text if word not in tokensOnce]
for text in texts]
>>> print texts
[['human', 'interface', 'computer'],
['survey', 'user', 'computer', 'system', 'response', 'time'],
['eps', 'user', 'interface', 'system'],
['system', 'human', 'system', 'eps'],
['user', 'response', 'time'],
['trees'],
['graph', 'trees'],
['graph', 'minors', 'trees'],
['graph', 'minors', 'survey']]
```

Nu skal samtlige af titlerne konverteres til vektorer og dette bliver muligt ved først at benyttes metoden Bag-Of-Words, BOW, til at repræsentere ordene. BOW metoden gør at hvert ord vil blive repræsenteret ved et fast ID, der er et heltal som henviser til ordet.

For at gøre dette oprettes først en ordbog, eller *dictionary*, der indeholder de 12 unikke tilbageblivende ord. Denne ordbog indeholder så kortlægningen af ordene til deres ID.

```
>>> dictionary = corpora.Dictionary.fromDocuments(texts)
>>> print dictionary
Dictionary(12 unique tokens)
>>> print dictionary.token2id
{'minors': 11, 'graph': 10, 'system': 5, 'trees': 9, 'eps': 8,
'computer': 0, 'survey': 4, 'user': 7, 'human': 1, 'time': 6,
'interface': 2, 'response': 3}
```

Når ordene i dette eksempel er blevet kortlagt til et heltal kan samtlige af titlerne konverteres til 12 dimensionale vektorer. Disse vektorer beskriver hvor mange gange hvert af de 12 ord forekommer i hver titel. I tilfælde af at et ord ikke forekommer i titlen vises dette implicit ved at udelade ordets heltal fra vektoren.

Ved at bruge den oprettede ordbog laves et corpus bestående udelukkende af de før omtalte vektorer.

```
>>> corpus = [dictionary.doc2bow(text) for text in texts]
>>> print corpus
[[(0, 1), (1, 1), (2, 1)],
[(0, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)],
[(2, 1), (5, 1), (7, 1), (8, 1)],
[(1, 1), (5, 2), (8, 1)],
[(3, 1), (6, 1), (7, 1)],
[(9, 1)],
[(9, 1), (10, 1)],
[(9, 1), (10, 1), (11, 1)],
[(4, 1), (10, 1), (11, 1)]]
```

Nu er vores corpus blevet lavet om til en Bag-Of-Words repræsentation, men dette giver ingen ekstra nyttevirkning i sig selv. For at kunne lave meningsfulde beregninger ud fra dette corpus skal det først transformeres.

Først oprettes en Term frekvens-invers dokument frekvens, tf-idf, model med vores corpus som træningskorpus og derefter benyttes tf-idf-modellen til, at transformere korpusset så det er repræsenteret ved vægtede værdier. I dette tilfælde transformeres det samme corpus som benyttes til træning, men når først tfidf-modellen er oprettet kan den bruges til at transformere alle vektorer fra det samme vektorrum.

```
>>> tfidf = models.TfidfModel(corpus)
>>> corpus_tfidf = tfidf[corpus]
>>> for doc in corpus_tfidf:
            print doc
 [(0, 0.57735026918962573), (1, 0.57735026918962573),
(2, 0.57735026918962573)]
[(0, 0.44424552527467476), (3, 0.44424552527467476),
(4, 0.44424552527467476), (5, 0.32448702061385548),
(6, 0.44424552527467476), (7, 0.32448702061385548)]
[(2, 0.5710059809418182), (5, 0.41707573620227772),
(7, 0.41707573620227772), (8, 0.5710059809418182)]
[(1, 0.49182558987264147), (5, 0.71848116070837686), (8,
0.49182558987264147)]
[(3, 0.62825804686700459), (6, 0.62825804686700459), (7,
0.45889394536615247)]
[(9, 1.0)]
[(9, 0.70710678118654746), (10, 0.70710678118654746)]
[(9, 0.50804290089167492), (10, 0.50804290089167492),
(11, 0.69554641952003704)]
[(4, 0.62825804686700459), (10, 0.45889394536615247), (11,
0.62825804686700459)]
```

Nu kan det tfidf transformerede korpus benyttes til foretage todimensionel Latent Semantic Indexing, LSI. Da korpus er indekseret i et 2-D rum, er det opdelt i to emner, som kan ses herunder. I følge LSI er ordene trees, graph og minors relaterede og bidrager mest til det første emne, hvilket er grafer. Hvorimod de overskydende ord næsten alle har relevans for det andet emne, menneske-maskine interaktion.

```
>>> lsi = models.LsiModel(corpus_tfidf, id2word =
dictionary.id2token, numTopics = 2)
>>> for topicNo in range(lsi.numTopics):
            print 'topic %i: %s' %
            (topicNo,lsi.printTopic(topicNo))

topic 0: 0.703*"trees" + 0.538*"graph" + 0.402*"minors" +
0.187*"survey" + 0.061*"system" + 0.060*"response" + 0.060*"time"
+ 0.058*"user" + 0.049*"computer" + 0.035*"interface"

topic 1: -0.460*"system" + -0.373*"user" + -0.332*"eps" + -
0.328*"interface" + -0.320*"time" + -0.320*"response" + -
0.293*"computer" + -0.280*"human" + -0.171*"survey" +
0.161*"trees"
```

Ordenes relation til de to emner kan også ses ved transformere vores korpus til det todimensionale latente rum. Her kan dokumenternes afstand til de to emner ses således

```
>>> corpus_lsi = lsi[corpus_tfidf]
>>> for doc in corpus_lsi:
            print doc
[(0, 0.06600783277618083), (1, -0.52007032130184239)]
[(0, 0.19667592692418112), (1, -0.76095631615039461)]
[(0, 0.089926399534091706), (1, -0.72418606462353752)]
[(0, 0.075858477293470852), (1, -0.63205516320005606)]
[(0, 0.10150299270848853), (1, -0.57373085347902797)]
[(0, 0.70321089393783098), (1, 0.16115180214025895)]
[(0, 0.87747875229457351), (1, 0.16758906577843058)]
[(0, 0.90986244430294261), (1, 0.14086553027268578)]
[(0, 0.61658254164787762), (1, -0.05392907532200053)]
```

Her ses det også at de fem første ord relaterer til det andet emne, mens de fire sidste relaterer mest til det første. Til sammenligning af dokumenterne benyttes cosinusligheden. Cosinusligheden er afstanden mellem to vektorer i vektorrummet. Afstanden går fra -1 til 1, hvor 1 betyder de er ens.

Nu hvor korpusset er blevet transformeret over i det todimensionelle latente semantiske rum kan der foretages lighedsforespørgsler på korpusset. Først indføres navnet på det dokument hvis lighed til de dokumenterne is korpusset

ønskes fundet. Dokumenttitlen transformeres derefter først over til en BOW vektor og herefter over til en vektor i det todimensionelle latente rum.

```
>>> doc = "Human computer interaction"

>>> vec_bow = dictionary.doc2bow(doc.lower().split())
>>> vec_lsi = lsi[vec_bow]
>>> print vec_lsi
[(0, 0.4618210045327148), (1, -0.07002766527899766 1)]
```

Så transformeres korpusset til LSI rummet og det indekseres. Når dette er gjort kan der foretages en lighedsforespørgsel med vores dokumentvektor på korpusset.

```
>>> index = similarities.MatrixSimilarity(lsi[corpus])
>>> sims = index[vec_lsi]
>>> print list(enumerate(sims))
[(0, 0.99809301), (1, 0.93748635), (2, 0.99844527), (3,
0.9865886), (4, 0.90755945), (5, -0.12416792), (6, -0.10639259),
(7, -0.098794639), (8, 0.050041765)]
>>> sims = sorted(enumerate(sims), key = lambda item: -item[1])
>>> print sims
[(2, 0.99844527), "The EPS user interface management system"
(0, 0.99809301), "Human machine interface for lab abc computer
                  applications"
(3, 0.9865886), "System and human system engineering testing of
                  EPS"
(1, 0.93748635), "A survey of user opinion of computer system
                  response time"
(4, 0.90755945), "Relation of user perceived response time to
                  error measurement"
(8, 0.050041765), "Graph minors A survey"
(7, -0.098794639), "Graph minors IV Widths of trees and well
                  quasi ordering"
(6, -0.10639259), "The intersection graph of paths in trees"
(5, -0.12416792), "The generation of random binary unordered
                  trees"]
```

Herover ses de værdier, som er udregnet på baggrund af indeksering der er blevet foretaget på korpusset. Først udskrives de i rækkefølge af titler og derefter sorteres de og udskrives i orden efter faldende lighed.

Det ses, at de fem første titler sorteret efter størst lighed hører under emnet "human computer interaction" og derefter kommer de titler der tilhører emnet grafer.

Det skal her gøres opmærksom på, at titel nummer to, *"The EPS user interface management system"*, og titel nummer 4, *"Relation of user perceived response*

time to error measurement", kommer op med en meget stor lighed, selv om titlerne ikke har et eneste ord tilfældes med forespørgslen. At de to titler kommer ud med en stor lighed er noget der er nemt at gennemskue når et menneske skal vurdere situationen, men det er ikke desto mindre to titler, som ikke ville dukke op ved en standard boolsk fuldtekstsøgning.

Det er netop dette aspekt af LSI, eller LSA, der gør det så ekstremt interessant indenfor området *Natural Language Processing*. LSI giver nemlig computeren en mulighed for at rejse sig fra den objektive verden bestående af sandt eller falsk, lig eller ikke lig, ved at finde de latente sammenhænge mellem ordene og benytte avancerede sammenligninger, når der for eksempel skal fortages søgninger. Med LSI kan computeren foretage simulerede subjektive beslutninger baseret på en teksts sammensætning, som i tutorialen, hvor en artikel blevet fundet frem ved en søgning, hvor ingen af søgeordene var tilstede.

2.4.2 Tekniske detaljer for LSA

I dette afsnit vil de tekniske detaljer involveret i Latent Semantisk Analyse, blive gennemgået i den rækkefølge analyseprocessen udføres i.

Først oprettes en term-dokument matrice der vægtes ved hjælp af term-frekvens-invers dokument frekvens. Den vægtede matrice opdeles i underkomponenter ved hjælp af Singulær Værdi Dekomposition og derefter reduceres rangen af systemet. Til sidst i afsnittet forklares, hvordan forskellige forespørgsler finder sted.

2.4.2.1 Term-dokument matricen

Det første skridt når der skal foretages latent semantisk analyse af en given tekst, er at producere en term-dokument matrice ud fra tekstens indhold. I en term-dokument matrice repræsenterer hver række et term, mens hver kolonne repræsenterer et dokument. På den måde viser hver række antallet af forekomster af ét specifikt term i hvert dokumenter og modsat viser hver kolonne, hvor mange forekomster af hvert term ét dokument indeholder. Grundlaget for LSA, term-dokument matricen er altså baseret på elementær ordoptælling. Matricens indhold er illustreret på *Figur 2-6*

$$\begin{matrix} & & \mathrm{d_j} \\ & & \downarrow \\ t_i^T \rightarrow & \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} & \end{matrix}$$

$$t_i^T = \begin{bmatrix} x_{i,1} & \cdots & x_{i,n} \end{bmatrix}$$

$$d_j = \begin{bmatrix} x_{1,j} \\ \vdots \\ x_{m,j} \end{bmatrix}$$

Figur 2-6: Term-dokument matrice. I matricen repræsenterer hver række et term, mens hver kolonne repræsenterer et dokument. Udtages en række vil den indeholde antallet af forekomster, af det term rækken repræsenterer, i hvert dokument. Udtages en kolonne vil denne indeholde antal af forekomster af hvert term i det dokument kolonnen repræsenterer.

Før et tekstkorpus indsættes i en term-dokument matrice foregår der som regel en del preprocessering. Dette inkluderer at fjerne ligegyldige stopord som ikke holder nogen værdifuld semantisk betydning og optræder hyppigt i hvert dokument. Derudover fjernes også ord som ikke forekommer mere end en gang.

Term-dokument matricen er en meget tom matrice, idet den hovedsageligt vil være udfyldt med nuller. Når matricen er oprettet vægtes elementerne med term frekvens-invers dokument frekvens, tf-idf.

2.4.2.2 Term frekvens-invers dokument frekvens

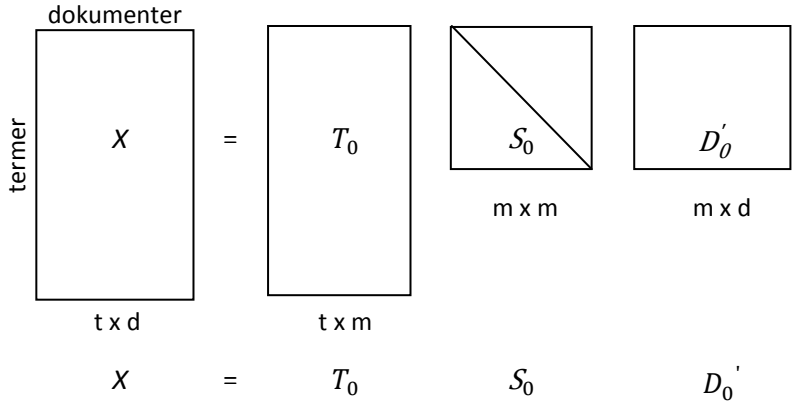
Term frekvensen vil sige, at vægten af et element i term-dokument matricen bliver større jo flere gange et term fremgår i et enkelt dokument. Da dette alene ville få ubetydelige ord, som fremgår ofte i alle tekster til at fremstå betydelige, bliver en invers dokument frekvens faktor inkluderet. Invers dokument frekvens mindsker vægten af termer som fremkommer meget hyppigt i hele korpusset og øger vægten af de termer der forekommer sjældent. Altså gør tf-idf at et element i matricen er proportionalt med antallet af gange det fremkommer i et korpus og sjældne termer bliver vægtet højere for at vise deres større relevans for konteksten. Når term-dokument matricen er vægtet benyttes Singulær Værdi Dekomposition, SVD, til at opdele matricen underkomponenter.

2.4.2.3 Singulær Værdi Dekomposition

Singulær Værdi Dekomposition er en matematisk teknik der benyttes til at analysere mønstre i sammenhængen mellem termer og dokumenter i term-dokument matricen. Annoteres vores term-dokument matrice, X, kan denne altså opdeles i produktet af tre matricer

$$X = T_0 S_0 D_0'$$

hvor T_0 og D_0' har ortonormale søjler bestående af de henholdsvis venstre og højre singulære vektorer[2] for *X*. S_0 er en diagonalmatrice der består af positive singulære værdier for *X*, i faldende orden. Dette er illustreret på *Figur 2-7*



Figur 2-7: Singulær Værdi Dekomposition af term-dokument matrix.
t er antallet af rækker eller termer i X.
d er antallet af søjler eller dokumenter i X.
m er rangen af X (≤min(t,d))

T_0 og D_0' indeholder henholdsvis term- og dokument vektorrummene. Når der er benyttet SVD på term-dokument matricen kan der i teorien allerede laves forskellige former for sammenligninger, men det er der mangler stadig et skridt der skal hjælpe med at optimere resultatet af sammenligningerne og simplificere modellen, nemlig reduktion af rang eller dimensioner.

2.4.2.4 Reduktion af rang

Dimensionerne af matricerne kan reduceres for at frembringe en optimal approksimering af den originale matrice, *X*. Samtidig reduceres effekten af statistisk støj og den latente sammenhæng mellem ord understreges [**4**].

Idet S_0 indeholder singulær værdier i faldende orden, kan man vælge at beholde de *k* øverste og dermed største værdier og ændre de resterende diagonalelementer til nul. Ved at gøre dette vil det resulterende produkt af matricerne, $\ddot{X}$, være tilnærmelsesvis lig med *X* og det vil være den rang *k* matrice der er tættest på *X* i forhold til mindste kvadraters metoden. For at simplificere matricerne kan de rækker og søjler fjernes fra S_0, hvor diagonalelementerne blev sat til nul og på samme måde kan de tilsvarende søjler fjernes T_0 og D_0'. Derved opnås

$$X \approx \ddot{X} = TSD'$$

Når *k* udvælges ønskes den valgt stor nok til at alle de rigtige strukturer i dataene bevares, men samtidig så lille, at uvæsentlige detaljer og sample fejl udelades.

[2] De venstre singulære vektorer svarer til egenvektorerne for XX', mens de højre singulære vektorer svarer til egenvektorerne for X'X. De singulære værdier i *S0* svarer til kvadratrødderne af egenværdierne for *XX'* eller *X'X*

Metoden hvorpå *k* udvælges er ikke standardiseret, men en god fremgangsmåde er at udfører forsøg med, hvilket antal dimensioner der giver de bedste resultater.

På dette tidspunkt i processen kan der foretages sammenligninger og de er fire sammenligningstyper

- Sammenligning af to termer
- Sammenligning af to dokumenter
- Sammenligning af term og dokument
- Sammenligning af term eller dokument og forespørgelse

Disse sammenligninger vil her blive forklaret i ovenstående rækkefølge.

2.4.2.5 Sammenligning af termer

For at foretage sammenligninger af to termer, udtages de to korresponderende rækker fra matricen TS og prikproduktet af de to rækker giver sammenfaldet mellem de to.

2.4.2.6 Sammenligning af to dokumenter

Teknikken til at sammenligne dokumenter fungerer på samme måde som med termer. Med dokumenter tages blot de rækker fra DS der udgøre de dokumenter der skal sammenlignes og prikproduktet af disse to rækker viser ligheden mellem dem.

2.4.2.7 Sammenligning af term og dokument

For at foretage sammenligninger mellem term og dokument tages prikproduktet af den række fra $TS^{½}$ der tilsvarer det ønskede term og den række fra $DS^{½}$ der tilsvarer det ønskede dokument.

2.4.2.8 Sammenligning af term eller dokument og forespørgelse

Dette er et af de rigtig vigtige aspekter af Latent Semantisk Analyse, da der kan findes termer og dokumenter der har den samme betydning eller konceptuelle indhold som forespørgslen.

Skal der foretages sammenligning med et dokument der ikke allerede er inkluderet term-dokument matricen, også kaldet et pseudo-dokument, kræver det at man først udregner en term vektor for dokumentet, X_p, der består af de vægtede termer som dokumentet indeholder. Ud fra X_p kan der så afledes en dokument vektor D_p, der sammen med S eller $S^{½}$ kan benyttes til at sammenligne med andre dokumenter eller med termer.

For et pseudo-term foregår processen på samme måde ved at udregne en dokument vektor for dokumentet, X_p, der indikerer hvilke dokumenter pseudo-termet fremgår i og i hvilken sammenhæng. Her afledes så term vektoren T_p, der

sammen med S eller $S^{½}$ så kan sammenlignes med andre termer eller med dokumenter i korpusset.

2.4.3 Konklusion på LSA

Vektor Modellen Latent Semantisk Analyse, har sit udspring i en simpel term-dokument matrice, der beskriver et terms forekomst på tværs af dokumenter, samt et dokuments fordeling på specifikke termer. Når denne matrice er vægtet med tf-idf, kan der foretages Singulær Værdi Dekomposition samt dimensional reduktion, hvilket gør det muligt, at uddrage de latente semantiske relationer, i form af afstande mellem vektorer i et vektor rum.

LSA er et værktøj til, at sammenligne ord og dokumenter internt i et korpus, men samtidig også til at sammenligne udefrakommende ord og dokumenter med korpussets indhold.

Kapitel 3

3 Case - Indsamling og analyse af mobil kontekstdata

I disse tider benyttes smartphones i stigende grad til, at generer store mængder tekstdata. Udover afsendelse af de traditionelle tekstbeskeder anvendes telefonerne også til e-mails, opdatere blogs og brug af forskellige sociale netværkstjenester, som Facebook og Twitter. Når brugeren er på farten anvendes smartphones til at udføre næsten samtlige af de funktioner MP3-afspilleren, telefonen og den bærbare computer stod for før i tiden. Det særlige ved at alle disse funktioner findes i en og samme enhed er, at det gør det muligt at opsamle alle mulige forskellige former for kontekstdata som brugeren producerer. Med henblik på dette er udviklet en mobilapplikation der kan indsamle kontekstdata med det formål, at analysere disse data i et forsøg på, at bestemme brugerens sindstilstand. Dertil er det interessant at undersøge hvordan de tidligere beskrevne semantiske analyse metoder kan benyttes på de indsamlede kontekstdata.

Ved at analyserer de indsamlede data bliver det forhåbentlig muligt at vurdere, hvorvidt det er realistisk at lave en applikation, der kan vurderer brugerens humør præcist.

Dette kapitel indeholder et afsnit om det ønskede design af indsamlingsapplikationen efterfulgt af et afsnit omhandlende den aktuelle implementering. Derefter beskrives metoden hvorpå der er blevet indsamlet kontekstdata, hvorefter den indsamlede data analyseres.

3.1 Design

I dette afsnit vil der først blive undersøgt, hvilke krav der er til indsamlingsapplikationen og på baggrund af dette vurderes applikations ønskede arkitektur. Herefter vil der blive foretaget en afgrænsning af hvilke sensorer applikationen skal benytte sig af efterfulgt af en begrundelse for de valg der er blevet taget.

3.1.1 Kravspecifikation

Den ønskede applikation skal kunne indsamle og gemme data for både fysiske og virtuelle sensorer[3]. Derudover skal det være muligt for brugeren at indtaste sit humør, så der ved den senere analyse af de indsamlede data kan foretages en

[3] Virtuelle sensorer måler ikke direkte på fysiske variable, men opsamler data internt i telefonen, eksempelvis SMS-beskeder.

sammenligning af det beregnede og det aktuelle humør. Applikationen skal nødvendigvis kunne udskrive de indsamlede data så de kan behandles og analyseres.

Af de mere praktiske funktionaliteter skal det helst være muligt for brugeren at tænde og slukke for applikationen direkte, i tilfælde af brugeren ønsker at beskytte personlige oplysninger der ellers ville være blevet opsamlet af applikationen. Med samme hensyn for øje ville det også være praktisk, hvis brugeren til alle tider kan se om der bliver indsamlet data.

Opsummeret skal den ønskede applikation kunne

- Indsamle og gemme data fra
  - fysiske sensorer
  - virtuelle sensorer.
- Modtage brugerens humør
- Aktiveres og deaktiveres eksplicit af brugeren
  - Tydeligt indikerer hvorvidt der indsamles data eller ej.
- Udskrive de indsamlede data til behandling

3.1.2 Design overvejelser

Herunder vil blive beskrevet nogle af de beslutning der er foretaget efter overvejelse af kravene i foregående afsnit.

3.1.2.1 Indsamling af data

Når der skal indsamles data på en smartphone er det i høj grad nødvendigt at der er tale om en smartphone med et styresystem der gør det muligt at multitaske. Dette vil sige, at applikationen vil kunne køre i systemets baggrund, usynligt for brugeren, så brugerens brug af telefonen samt adfærdsmønster ikke ændres.

Med hensyn til hvilke sensorer der skal indsamles fra er det klart at det skal være sensorer der kan sige noget konkret om brugerens sindstilstand. Når det er sagt ønskes data indsamlet fra så mange sensorer som muligt for, at få mest mulig data at kunne analysere og konkludere på.

Som beskrevet er det hensigtsmæssigt for brugerens følelse af privatliv, at det er nemt og ligetil at starte og ikke mindst stoppe dataindsamlingen. Det skal helst være en åbenlys mulighed for brugeren, så det ikke virker uønsket at indsamlingen stoppes et kort stykke tid i tilfælde af at private oplysninger skal formidles gennem telefonen.

3.1.2.2 Opbevaring af data

De data der indsamles løbende af applikationen skal nødvendigvis skrives direkte til en harddisk eller hukommelseskort når de indsamles. I modsatte tilfælde, hvor data eventuel holdes internt i applikation kan det hele gå tabt i tilfælde af fejl i applikationen, fejl i styresystemet eller at telefonen pludselig slukkes, ved for eksempel tomt batteri.

3.1.2.3 Indføring af brugerhumør

Generelt for en applikation som denne, der benyttes til passiv dataindsamling, er brugervenlighed ikke den primære bekymring. Dette er dog ikke tilfældet når det kommer til den del, hvor brugerens humør skal gemmes, tværtimod. Når brugeren skal indtaste sin sindstilstand er det af yderste vigtighed, at brugeren er fuldstændig fortrolig med den benyttede metode.

Udover at metoden skal være nem for brugeren at forstå skal det også helst være en anerkendt metode til at anføre humør med. Den skal altså gerne være testet tidligere og være beskrevet tilstrækkeligt i videnskabelig litteratur.

Afslutningsvis skal den benyttede metode, udover at være veldokumenteret og være intuitiv, heller ikke tage lang tid at benytte. Det skal gå hurtigt at indføre sit humør, så belastningen af brugeren er så lav som mulig og så brugeren forhåbentlig vil indfører sindstilstand så ofte som muligt.

3.1.2.4 Beskyttelse af personlige oplysninger

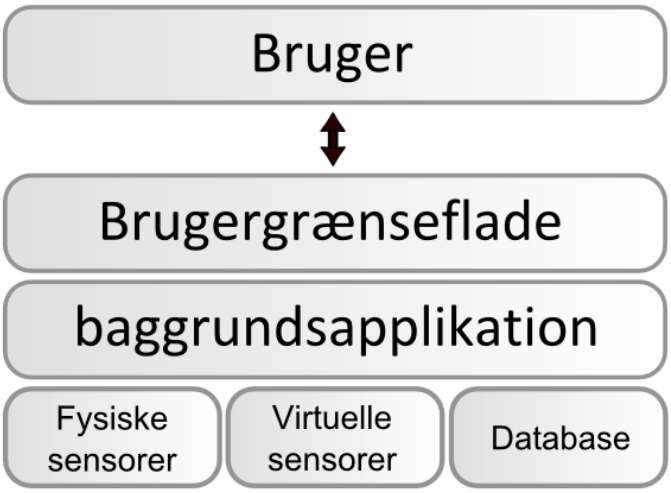
Grundet de personlige oplysninger som applikationen muligvis kan indsamle er det nødvendigt at udtænke en metode der kan gøre det mere trykt for brugerne af applikationen at have den aktiveret.

De oplysninger der hovedsageligt skal beskyttes, er SMS beskederne, da de indeholder meget direkte læseligt data.

Desuden vil det være hensigtsmæssigt, hvis telefonnumre ikke arkiveres, da disse kan være meget private for brugeren.

3.1.3 Applikationens arkitektur

På baggrund af de opstillede krav i de foregående afsnit er herunder fremstillet applikationens arkitektur.



Figur 3-1: Applikations arkitektur

Som det ses på figuren er det øverste lag applikationens brugergrænsefalde. Det er gennem dette lag brugeren kan starte og stoppe dataindsamlingen, samt angive sit eget humør, slukke applikationen og udskrive den indsamlede data. Desuden er det også her brugeren kan se om applikationen indsamler data eller ej. Under brugergrænsefladen er den del af applikationen der kører i baggrunden og står for at indsamle data fra de fysiske og virtuelle sensorer. Disse data skrives direkte til databasen.

3.1.4 Afgrænsning af sensorer

Herunder er en listen over de sensorer der er blevet valgt til indsamlingsapplikationen.

- SMS
- Opkald
- Musik
- Position

Andet input

- Bruger humør
- Tid

Sensorerne er blevet udvalgt fordi de menes at have betydelig indflydelse på eller afhængighed af brugerens humør. I de følgende underafsnit vil der blive

udspecificeret præcis, hvorfor hver enkel sensor er blevet valgt. De sidste to input kommer ikke direkte fra sensorer, men skal stadig indsamles og gemmes.

SMS beskeder og afspillet musik vil gennemgå en præliminær behandling ved indsamling som vil blive beskrevet under deres afsnit samt overvejelser for hvordan brugerens humør indføres.

3.1.4.1 Short Message Service, SMS

Indsamling af brugerens SMS beskeder er essentielt for beregning af brugerens humør, da en tekst der sendes i en SMS besked er en meget direkte form for kommunikation, der afspejler brugerens humør.

Når det kommer til hvad der præcis der skal indsamles, er der flere ting der skal overvejes. Om det både er indgående og udgående SMS beskeder der skal gemmes, samt hvilke af de mange informationer der skal gemmes.

I dette tilfælde vil input fra den virtuelle sensor blive begrænset til kun at omhandle de SMS beskeder der afsendes af brugeren selv. Dette skyldes af de afsendte SMS beskeder direkte afspejler brugerens humør, da det er brugerens egne tanker der er nedfældet i ord. Modtagne SMS beskeder er til gengæld et resultat af en anden persons tanker og ordenes påvirkning af brugerens humør er et langt mere abstrakt emne.

Med hensyn til hvilke data der gemmes er det dato for SMS beskedens afsender, indholdet af beskeden naturligvis, samt en måde hvorpå modtageren kan skildres fra andre modtagere.

Til at vurdere humøret af udgående SMS beskeder kan ANEW samlingen, oversat til dansk i dette tilfælde, benyttes til at kontrollere om nogle af ordene i SMS beskeden går igen på listen. Derved kan valence og arousal værdier for brugerens SMS beskeder bestemmes opsamles og lagres.

3.1.4.2 Opkald

Det er blevet besluttet også at indsamle brugerens opkaldshistorik, mens indsamlingsapplikationen er aktiveret. Selv om det ikke umiddelbart er muligt, at udvinde noget specifikt fra, hvilke numre brugeren har haft samtaler med på forskellige tidspunkter, menes der stadig at kunne findes nyttige oplysninger i disse informationer. Det gøres ud fra den antagelse om, at man ved hjælp de oplysninger der er udvundet fra en anden virtuel sensor, nemlig SMS beskederne, kan drage forbindelser til de foretagne opkald. Har brugeren for nyligt skrevet en SMS besked til et nummer, hvormed der kort tid efter føres en samtale, findes det

rimeligt at antage at stemningen i denne samtale befinder sig i nærheden af den fra SMS beskeden.

3.1.4.3 Musik

Valget af musik som virtuel sensor skyldes, at det er velkendt, at musik og følelser går hånd i hånd [**7**]. Når et stykke musik afspilles vil det frembringe følelser eller måske forstærke en allerede tilstedeværende følelse eller sindstilstand hos brugeren. Ligesådan må det antages, at brugerens valg af musik afhænger af humøret på det givne tidspunkt, hvor musikken vælges og afspilles. Selve musikvalget kan derfor næsten tolkes som brugerens egen vurdering af sit humør eller et ønske om at opnå en bestemt sindstilstand. Der er altså to aspekter af musikken, nemlig det at musikken frembringer følelser hos brugeren og det at musikken er valgt på baggrund af følelser hos brugeren [**7**]. I daglig brug skal det dog medtages i betragtningen af måleresultaterne, at mange brugere nok vil have deres musik i en afspilningsliste, hvor de afspillede sange så vil blive tilfældigt afspillet fra.

For at vurdere humøret af et stykke musik benyttes last.fm's offentlige API, der gør det muligt hente de tags som brugerne af last.fm har tildelt sangen. Når de respektive tags er hentet kan de gennemsøges for sammenfald med den engelske ANEW samling for at estimere stemningen af musikken.

3.1.4.4 Position/sted

Brugerens position har meget at sige om, hvilken sindstilstand brugeren er i. Det kan være svært at vurdere humør blot efter en position, men derimod kan position i sammenhold med andre kontekstdata benyttes til at oprette steder, hvor brugerens humør ofte er det samme.

Er det muligt, at inddele positionerne i grupper som arbejde, fritid, hjem eller rejse, kan der muligvis også være muligt at benytte disse som delfaktorer for brugerens humør.

I forhold til implementering er det igen relativt nemt at indsamle den nødvendige data, men også tilsvarende svært at udføre en meningsfuld fortolkning. Det vil være nødvendigt at indsamle fra GPS samt Wi-Fi for at opnå tilstrækkelig præcision. Positionering fra mobilmasternes cell-id kan muligvis ikke levere tilstrækkelig gode positionsdata.

3.1.4.5 Bruger humør

Som nævnt tidligere skal applikationen kunne modtage brugerens vurdering af eget humør, for at have et sammenligningsgrundlag for de udregnede værdier.

Med hensyn til metoden hvorpå brugeren skal indføre sit humør i applikationen er der udtænkt flere forskellige muligheder. De skal alle relateres til valens og ophidselse skalaen, for at kunne sammenlignes med det input der fås fra de andre sensorer. De udtænkte muligheder er

- En skala med forskellige former for smileys der henviser til ophidselses og valens værdier.
- To skalaer hvorpå brugeren skal vælge sin valens og ophidselse værdi separat.
- En graf med valens og ophidselse ud af x og y akserne, med ord fra ANEW listen påført som guide.

Ud af de tre muligheder blev den sidste originalt valgt til indsamlingsapplikationen. Det blev antaget på det tidspunkt, at grafen gjorde det intuitivt klart for brugeren hvordan valens og ophidselse skalaen virker, samtidigt med de påførte ord fra ANEW samlingen kunne hjælpe brugeren til et præcist humør estimat. Desuden går det hurtigt at indføre humør på denne måde, så brugeren ikke føler sig besværet unødigt.

Pilotundersøgelser med applikationen viste imidlertid, at der ved brug af grafen opstod et fundamentalt problem. Problemet bestod i, at brugerne etablerede vaner for, hvordan de afgav deres humør og viste tendenser til blot at trykke på det ord de syntes bedst beskrev deres nuværende tilstand. Dette gjorde også at de ofte benyttede de samme værdier, da det var ved de samme ord på grafen de ofte trykkede.

Konsekvenserne af dette var at skalaen ikke blev brugt ordentligt, da den skal bruges flydende til at kunne beskrive brugerens humør nuanceret, i stedet for blot at vælge nogle faste på forhånd indsatte værdier. Ved uddeling af den originale applikation mentes det, at brugeren ville benytte ordene på grafen vejledende og indfører sit humør et sted mellem ordene eller på ordene alt efter hvad der passede bedst.

Da grafen ikke fungerede optimalt ved pilotundersøgelsen blev metoden lavet om til en modificeret version af den anden metode, nemlig to skalaer med valens og ophidselse. Indførelsesmetoden vil være en efterligning af *Self Assesment Manikin*, SAM, som består af to rækker af mænd, den ene med ni mænd, hvor de går fra sur til glad indikeret ved deres smil. I den anden række går de fra at være afslappet til ophidset indikeret ved at de har en prik der står stille i maven på dem, som ændrer sig til at bevæge sig helt vildt på den sidste.

De to forskellige input metode vil være at finde under *3.2.3 Opbygning*

3.1.4.6 Tid

Da alle mennesker har mere eller mindre faste rutiner og rytmer der gentages hvert døgn, hver uge, hver måned eller hvert år indgår tiden som en naturlig måde at vurderer brugerens humør på. De rutiner som gennemgås med jævne intervaller vil utvivlsomt påvirke et individs sindstilstand mærkbart. Tid og dato som sensor input er i sig selv meget simpelt at indsamle, udfordringen her ligger i at foretage en meningsfuld fortolkning af de indsamlede data. For at tid og data skal give nogen eksplicit mening kræves det først, at der i sammenhold med input fra de andre sensorer bliver bestemt, hvilke tidspunkter der påvirker brugeren forskelligt.

3.2 Implementering

I dette afsnit vil den aktuelle implementering blive gennemgået med begrundelse for de valg der er taget. Implementeringen er foretaget på baggrund af de overvejelser der er gennemgået i foregående afsnit.

3.2.1 Valg af teknologi

Da indsamlingsapplikationen skal udvikles til en smartphone, grundet dennes unikke spændvidde i brug, står valget af teknologi på, hvilken smartphone og primært hvilket mobile operativ system der skal bruges. Grunden til operativsystemet er det vigtigste valg skyldes, at de fleste smartphones har nogenlunde samme egenskaber og sensorer, mens det er operativsystemet som sætter begrænsninger for hvordan telefonens hardware kan benyttes. De egenskaber der skal være opfyldt for et operativsystem er

- Multitasking, der tillader applikationer at køre videre i baggrunden. Dette skal være muligt, da applikationen skal kunne køre i baggrund uden at forstyrre brugeren.
- Operativsystemet skal være vidt benyttet for at have mange potentielle brugere.
- Gode distributionsmuligheder for at gøre det muligt at få applikationen ud til brugerne.

Ud fra disse tre hovedpunkter vælges Android som udviklingssystem. Dette skyldes hovedsageligt Androids multitasking egenskaber, som tillader udviklingen af programmer der kan køre i baggrunden så længe det er nødvendigt eller muligt.

Da dette valg blev foretaget stod det mellem iOS på Apples iPhone og Android. Valget var på tidspunktet dog ikke svært at foretage, da det blev gjort inden udgivelsen af Apples iPhone 4, som tillader ægte multitasking, hvilket de forrige iPhones ikke gjorde. De tidligere iPhone tillod simuleret multitasking ved at gemme den tilstand en applikation var i og derved bringe brugeren til en applikation der lignede den havde været åben hele tiden, men i virkeligheden ikke kørte i baggrunden.

Udover multitasking har begge operativsystemer en stor mængde brugere og meget lig distributionsmuligheder med iTunes App Store og Android Market.

Med valget af Android kommer også nogle fordele i form af

- Nem og billig distribution via Android Market ***uden*** forundersøgelse af applikationen. Tilmeldingsgebyret er cirka 130kr ($25 US dollars) og derefter er der ingen omkostninger. For at have sin app på Apple's App store koster det cirka 515kr ($99 US dollars) årligt, samt 30% af salget af applikationen.
- Effektiv udvikling via Android Development Tool til Eclipse.
- Mulighed for distribution af Android baserede HTC Desire fra IMM til frivillige studerende.

Da det er besluttet at valget står på Android skal det så bestemmes, hvilken version af styresystemet der skal benyttes. Det bliver Android 2.1, også kaldet Eclair, da dette er det styresystem som HTC Desire sælges med. Derudover laves applikationen kompatibel med Android 2.2, Froyo, da styresystemet på HTC Desire kan opgraderes til denne version.

3.2.2 Om Android

To af de grundlæggende begreber i Android applikationer som benyttes i indsamlingsapplikationen er Activity og Service.

En Activity er den del af en applikation der står for bruger interaktion med applikationen. En Activity indeholder en beskrivelse af den grafiske brugergrænseflade[4] og funktioner til at håndtere de input brugeren kan give brugergrænsefladen. Flere Activities kan derfor sættes sammen til at give applikationen det ønskede udseende og de ønskede funktionaliteter.

En Service er en del af en applikation der benyttes, hvis en applikation ønskes at udfører funktioner i baggrunden uden bruger interaktion og uden at fastlåse brugergrænsefladen mens applikationens funktionaliteter udføres. En Service kan

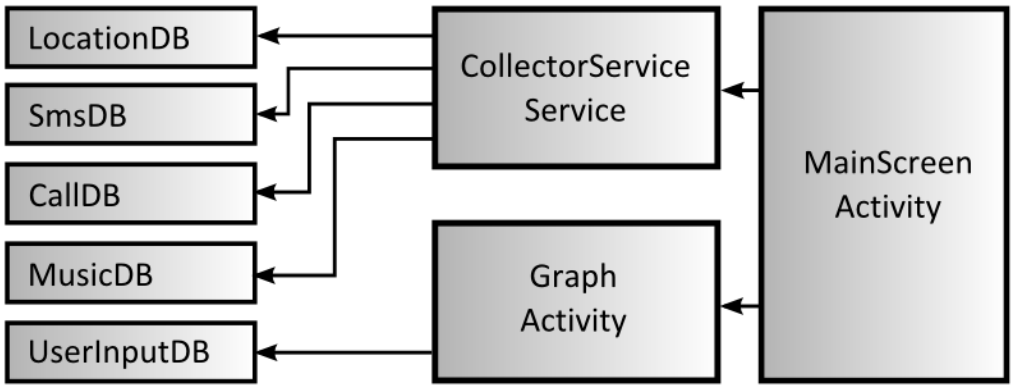
[4] ofte kalde GUI eller UI fra graphical user interface

altså kører i baggrunden samtidig med brugeren benytter applikationen eller når brugeren har afsluttet applikationen og benytter andre af telefonens egenskaber.

3.2.3 Opbygning

Dette afsnit indeholder den aktuelle opbygning af indsamlingsapplikationen og da den er navngivet MoodDetective vil der blive refereret til den ved dette navn. MoodDetective er opdelt i fire hoved elementer som er angivet herunder

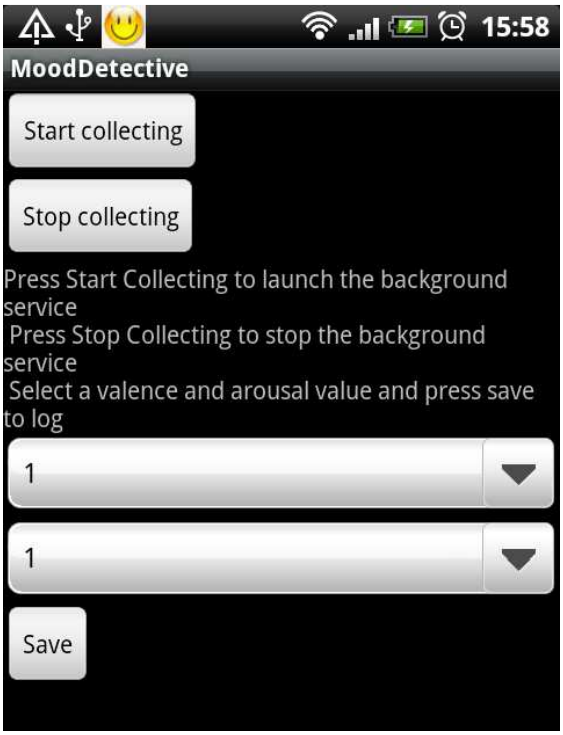
- MainScreen - den primære Activity.
- Graph - en Activity til at indfører brugerens humør.
- CollectorService - den Service der indsamler data.
- Databaser - fem databaser der indeholder de indsamlede data.
    - LocationDB
    - SmsDB
    - CallDB
    - MusicDB
    - UserInputDB



Figur 3-2: Opbygning af indsamlingsapplikationen MoodDetective

Her forklares den fundamentale opbygning af applikationen som illustreret ovenfor på *Figur 3-2*

MainScreen er en Activity og samtidig applikationens main-fil, den er altså den første der køres når applikationen starter. Når applikationen startes vises brugergrænsefladen som ses på Figur *3-3*



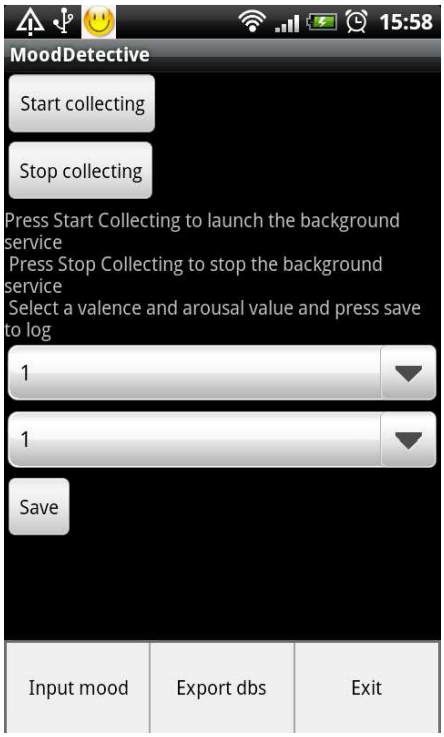
Figur 3-3: Startskærmen, MainScreen, i MoodDetective

Fra MainScreen kan brugeren vælge at starte og stoppe indsamling af data ved brug af knapperne "Start collecting" og "Stop collecting", som henholdsvis starter og stopper CollectorService. Når CollectorService er aktiv vises en notifikation i toppen af skærmen som kan ses på *Figur 3-4*.

Figur 3-4: Ikonet der vises i notifikationsområdet når der indsamles data til MoodDetective

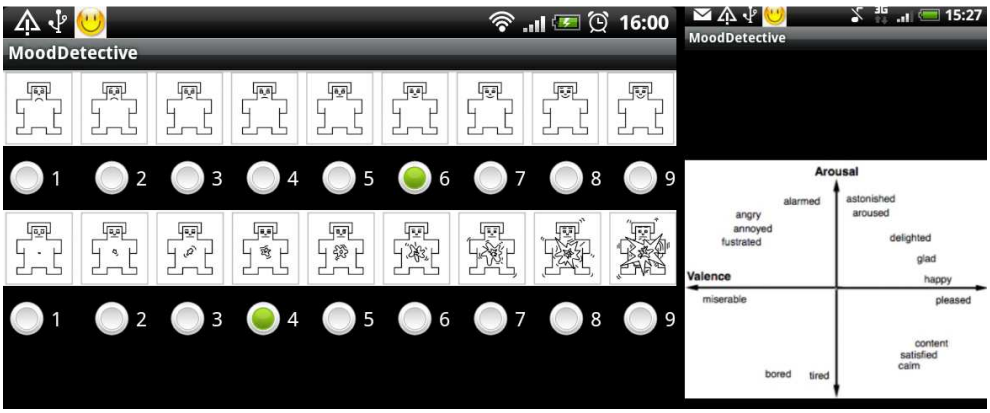
Derudover kan brugeren manuelt indfører sine valence og arousal værdier ved hjælp af to drop-down menuer og værdierne gemmes så i databasen når der trykkes save.

Fra denne brugergrænseflade kan desuden frembringes en menu i bunden af skærmen ved at trykke på Android telefonens menu-knap. Denne menu kan ses på *Figur 3-5*. Når menuen er fremme kan brugeren vælge mellem "Input mood", Export dbs" og "Exit".



Figur 3-5: Mainscreen menu

Ved valg af "Input mood" i menuen åbnes en ny Activity, der viser SAM som blev beskrevet under *2.1 ANEW*. Denne som vises til venstre Figur 3-6, hvor valens og ophidselse vælges separat. Til højre på figuren ses (Valens,Ophidselse)-grafen som blev fravalgt til fordel for SAM, hvilket forklares i *Afgrænsning af sensorer*.



Figur 3-6: Sammenligning af SAM(venstre) og (Valens,Ophidselse)-grafen(højre).

De to sidste menuvalg er meget selvforklarende, idet "Export dbs" eksporterer de data der er opbevaret i applikationens databaser til tekstfiler på SD-kortet, mens "Exit" blot lukker applikationen og CollectorService hvis denne er åben.

3.2.4 Detaljeret beskrivelse

Herunder vil funktionaliteten af applikationens avancerede funktioner blive beskrevet. Da CollectorService indeholder al koden der står for den automatiske indsamling af data vil der blive fokuseret på denne. Ydermere vil afsnittet blive underopdelt i de tre indsamlings områder

- Lokation
- SMS og Opkald
- Musik

CollectorService fungerer således at en timer er sat til at aktivere hvert 3. minut selv om der dog ikke indsamles data i alle de ovenstående kategorier hver gang timeren aktiveres. Den samlede indsamling foregår i cykler af 15 minutter, altså fem aktiveringer per cyklus. Forklaring om hvornår og hvordan data indsamles indenfor de forskellige kategorier kan findes i de følgende afsnit.

3.2.4.1 Lokation

Ved CollectorService's start oprettes en LocationManager, der er en klasse der giver adgang til androids lokations services. Denne anmodes så om at indsamle GPS og NETWORK data hvert 15 minut. Når der anmodes om NETWORK data menes der, at hvis Wi-Fi er tilgængeligt indsamles lokationsdata ved hjælp af omkringliggende Wi-Fi netværk og hvis Wi-Fi ikke er tilgængeligt indsamles via mobilmasterne.

De indsamlede lokationsdata læses fra LocationManager i hver fjerde aktivering, altså den første gang der læses fra LocationManager'en, er 12 minutter efter Servicens opstart og herefter hvert 15 minut. Forskydelsen i LocationManager'ens dataindsamling og læsning af disse data er indført for ikke at risikere at læse fra LocationManager'en samtidig med dennes lokationer opdateres, for at undgå eventuelle stabilitetsproblemer for programmet .

Grunden til der ikke indsamles lokationsdata oftere end hvert 15. minut er blandt andet for at reducere applikationens batteriforbrug, da især GPS dataindsamling bruger meget energi. Derudover er det ikke nødvendigt for applikationens brug at have detaljeret information om brugerens færden mellem opholdssteder, men blot på hvilke lokationer brugeren ofte opholder sig i længere tid .

For at opsamle så meget information som muligt og for at kunne vurdere hvilken datakilde der bør benyttes i senere versioner af applikationen gemmes begge kilder til lokationsbestemmelse i databasen. Udover længdegrad og breddegrad for GPS og NETWORK lokationer gemmes desuden præcisionen af lokationsbestemmelsen, tidspunkt af indsamling i LocationManager'en samt tidspunkt for indførelsen i databasen.

3.2.4.2 SMS

Indsamling og analyse af afsendte SMS finder sted i den femte og sidste 3 minutters cyklus efter CollectorService startes. Ved start af CollectorService gemmes Servicens starttidspunkt, dette er nødvendigt for at SMS beskeder der er afsendt før dette tidspunkt ikke blive analyseret og gemt i databasen. Når denne foranstaltning er indført, er det først og fremmest fordi SMS beskeder fra før anden data også er begyndt indsamlet ikke har relevans i forhold til den samlede vurdering, men også for at Servicen kan slukkes i tilfælde af brugeren eventuelt har brug for at holde informationen helt privat.

Når SMS indsamlingen startes hvert 15 minutter, i den femte cyklus, hentes alle SMS beskeder, der er afsendt efter CollectorService er startet, ind i en liste. Har en SMS indsamling fundet sted før hentes alle afsendte SMS tilbage til det tidspunkt hvor der sidst blev indsamlet.

SMS beskederne der er gemt i listen indføres så i en analyse-funktion en af gangen, hvor der søges efter ord fra den danske oversættelse af ANEW listen. De ord der findes i en SMS gemmes som features i en ny liste sammen med den samlede gennemsnitlige valens og ophidselse. Denne liste benyttes så sammen med listen indeholdende de rå SMS data og de relevante data gemmes i databasen. De data der gemmes i databasen er et ID der repræsenterer modtagerens nummer, den afsendte SMS indlæst bagfra, fundne features, valens og ophidselse, afsendelses tidspunkt samt tidspunktet hvorpå SMS beskederne er blevet gemt i databasen. Grunden til alle data gemmes selv om de er blevet analyseret, er fordi de så kan efteranalyseres, hvis dette ønskes.

Grunden til SMS beskederne indlæses baglæns, er for at gøre den ulæselig når databasen udskrives til tekstfil. Selvfølgelig kan omvendte SMS beskederne stadig læses bagfra, hvis man ønsker det, men når loggen skimmes er det svært at udlede noget af betydning.

For at gøre det mere sikkert for brugeren gemmes et ID nummer i databasen der kan eksporteres, i stedet for det rigtige telefonnummer. ID nummeret henviser til det rigtige telefonnummer der ligger i en intern database, som der ikke er adgang til.

3.2.4.3 Opkald

Opkald gemmes på præcis samme måde og i samme interval som SMS beskederne. Her benyttes også den samme telefonnummer database til at tildele ID-numre i stedet for telefonnumre.

3.2.4.4 Musik

Information fra musikafspilleren er den eneste indsamling som finder sted i hvert af CollectorServices tre minutters cyklusser. At der indsamles data fra musikafspilleren hvert tredje minut, er et forsøg på at indsamle hver gang en ny sang spilles. Denne fremgangsmåde har været nødvendig, da det ikke er muligt at få at vide, når et nyt nummer starter.

Før hver musik indsamling kontrolleres først om musikafspilleren kører i baggrunden er det tilfældet etableres en serviceforbindelse mellem CollectorService og musikafspilleren. Kunstnerens og sangens navn hentes så og indsættes i en URL, der indeholder last.fm [**8**] API-metoden Track.getTopTags[5]. Svaret fra URLen bliver så modtaget af en XML reader. Gav last.fm søgningen et resultat vil svaret have formen som ses på Figur 3-7

```
<toptags artist="Cher"
track="Believe">
  <tag>
    <name>pop</name>
    <count>97</count>
    <url>www.last.fm/tag/pop</url>
  </tag>
  <tag>
    <name>dance</name>
    <count>88</count>
    <url>www.last.fm/tag/dance</url>
  </tag>
  ...
</toptags>
```

Figur 3-7: Last.fm XML svar på Track.getTopTags

Den XML reader som modtager svaret er så sat op med en XML Handler der gemmer alle felterne med navnet "name" i en liste, hvilket vil sige den samler alle tags. Ud af de modtagne tags gemmes kun de første 25 for at begrænse mængden af tags med meget få "counts". Antallet af "counts" indikerer, hvor mange gange en sang er blevet tagget med et specielt tag. Ved at sætte en begrænsning undgås tags med få "counts", som altså kan være irrelevante eller forkerte.

De indsamlede tags køres herefter igennem en analyse-funktion på samme måde som SMS beskederne, hvor de fundne tags i stedet sammenlignes med den engelske ANEW liste. De tags der bliver fundet i listen gemmes sammen med den gennemsnitlige valens og ophidselse.

Når tags er blevet analyseret bliver kunstnerens og sangens navn, de fundne features, valens og ophidselse scoren samt tiden for indførslen i databasen gemt.

[5] http://ws.audioscrobbler.com/2.0/?method=track.gettoptags&artist=name&track=track&api_key=key

3.3 Indsamling af data

I dette afsnit vil det blive beskrevet hvordan dataindsamlingen fandt sted. Data indsamlingen fandt sted i tre faser

- Pilotforsøg på egen mobil og en medstuderende
- Uddeling af HTC Desire til frivillige
- Tilbagekald af smartphones

Applikationen blev først testet på en medstuderendes og egen HTC Desire. Dette blev gjort løbende, mens den blev udviklet, samt i tiden op til distributionen. Ved teste applikationen løbende kunne fejl og mangler udbedres når der opstod bevidsthed om dem. Især i den sidste periode, hvor applikationen var færdig, men ikke var blevet uddelt foregik en megen finjustering og fejlkorrektion for at undgå uforudsete nedbrud af applikation.

For at få applikationen delt ud, blev dens formål og funktionalitet beskrevet til de studerende i kurset *02817 Personalisering og metadata modeller F11*. Derefter blev de studerende spurgt om de ville benytte applikationen i en begrænset periode imod at låne en HTC Desire. Ved lejligheden blev det gjort de studerende klart, at indholdet af deres SMS beskeder ville forblive fortroligt og ikke ville blive læst. Ved denne lejlighed blev 10 smartphones udlånt og e-mail adresser blev indsamlet for at kunne give instruktioner om applikationens installation og brug. Udover at uddele applikationen til de 10 studerende blev den også uddelt til venner samt benyttet af undertegnede.

Telefonerne samt den indsamlede data blev tilbagekaldt efter tre ugers brug ved at sende en e-mail til de studerende om at medbringe den lånte HTC Desire til næstfølgende forelæsning. Ved forelæsningen blev nogle af telefonerne leveret tilbage, mens andre blev beholdt til brug i andre kurser. Den indsamlede data blev overdraget ved forelæsningen eller efterfølgende per e-mail.

Da telefonerne skulle tilbageleveres gik noget data tabt, da nogle af de studerende uden opfordring foretog en nulstilling til fabriksstandard.

3.4 Analyse af data

I dette afsnit vil de indsamlede data blive analyseret. Først vil de behandlede data blive holdt op imod brugernes input og derefter vil der blive brugernes adfærdsmønstre blive undersøgt. Grundet en begrænset mængde af musik og opkaldsdata, vil SMS beskederne som de eneste blive analyseret og disse vil blive holdt op mod brugernes indførte humør.

3.4.1 Kontrol af data

Da de datasæt der blev indsamlet af de studerende er mangelfulde og næsten udelukkende består af GPS placeringer og brugerhumør, vil mine egne indsamlede data blive benyttet til at kontrollere, hvorledes de beregnede humørværdier stemmer overens med de indførte. Alting tyder på, at de studerende ikke brugte telefonen til at foretage opkald og sende SMS beskeder, da der næsten ikke blev indsamlet nogen data fra de udlånte telefoner.

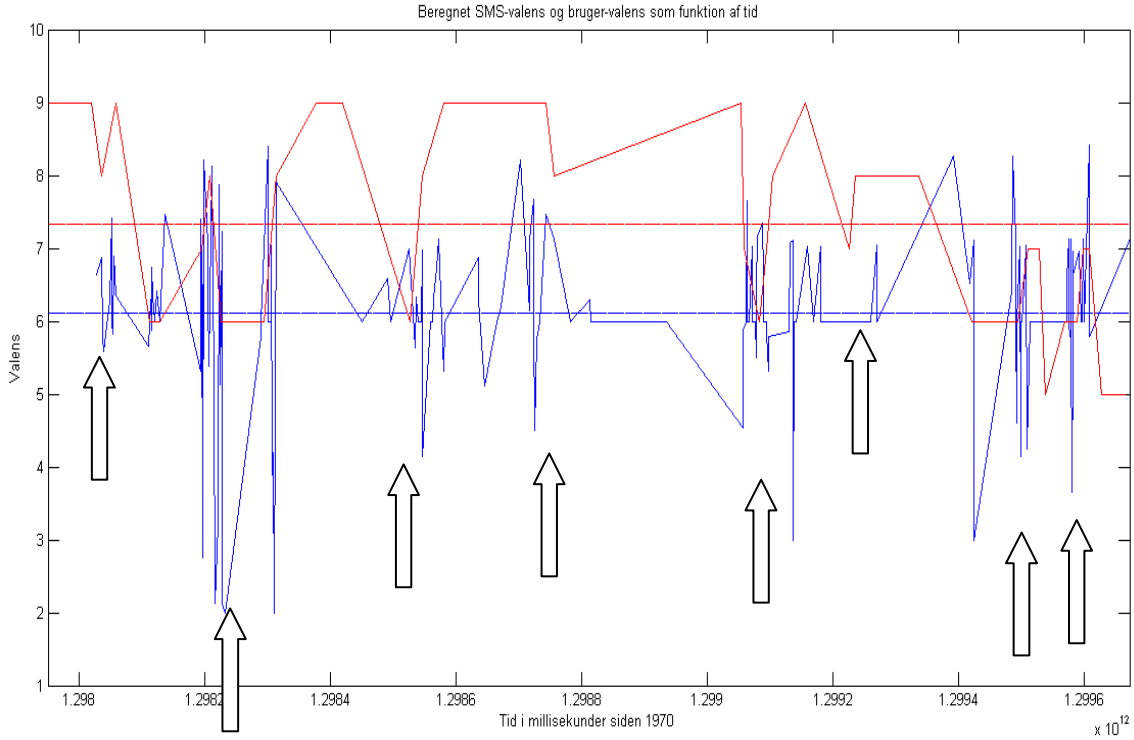
I det efterfølgende afsnit vil de beregnede værdier fra de indsamlede SMS beskeder blive evalueret mod de indtastede humørdata.

3.4.1.1 Humør fra SMS

Her sammenlignes det beregnede humør fra SMS beskeder, med det brugerindtastede humør. Grafen på Figur 3-8 viser den beregnede SMS-valens og bruger-valens som funktion af tiden. Den røde linje viser brugerens valens input og den røde stiplede linje viser gennemsnittet af brugerens valens. Den blå linje viser på samme måde SMS valens, hvor den blå stiplede er gennemsnittet.

På grafen kan det ses, at de to grafer er forskudt i valens og dette ses også på gennemsnittet af bruger valensen som er cirka 20 procent større end den fra SMS beskederne. Selv om de to valenser ikke falder sammen kan der dog stadig ses nogle sammenfald i udslagene på de to grafer. De interessante lokale minimum og maksima er markeret på grafen og det ses også, at udslag i brugerens egen vurdering ofte sker i sammenfald med at der er blevet sent mange SMS beskeder på kort tid.

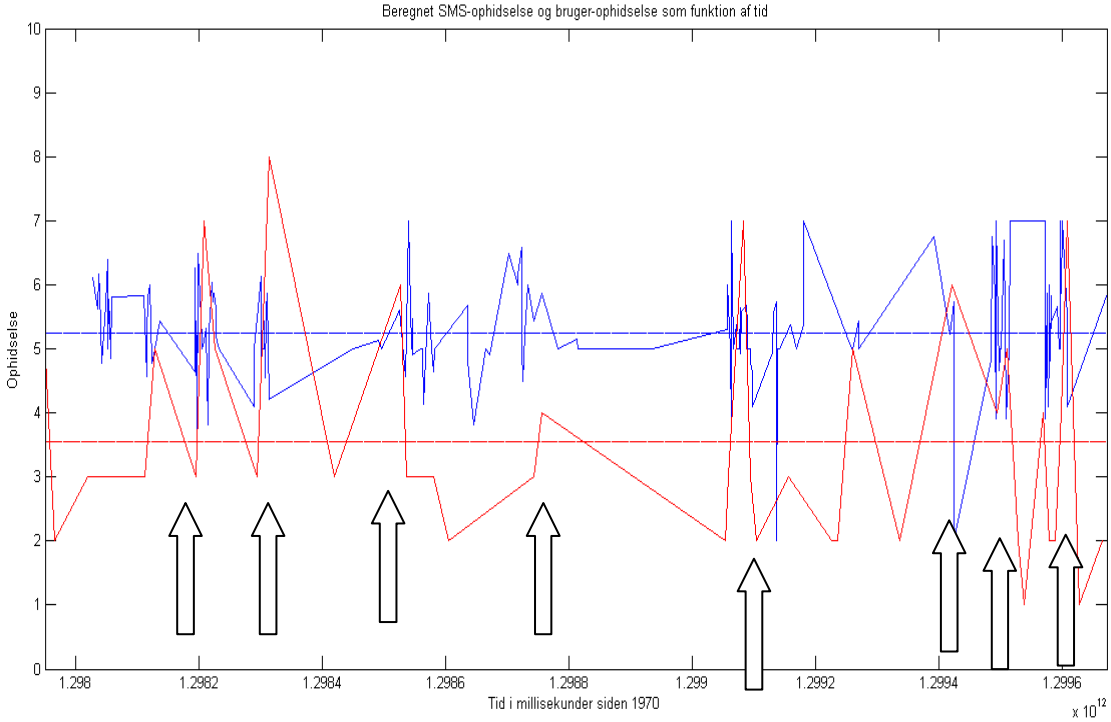
At udslagne i begge grafer følger hinanden til en vis grad, kan tyde på, at der ved justering af beregningsmetoden vil kunne opnås større sammenfald i de udregnede og de indførte værdier. Eventuelt vil sammensætningen af flere forskellige vurderingsmetoder kunne resulterer i en bedre overensstemmelse.



Figur 3-8: Beregnet SMS-valens og bruger-valens som funktion af tid.
Forklaring: Rød - bruger-valens, Rød stiplet - bruger-valens gennemsnit, Blå - SMS-valens, Blå stiplet SMS-valens gennemsnit.

På *Figur 3-9* ses den beregnede ophidselse ud fra SMS beskeder sammen med brugerens indtastede ophidselse. Den røde linje indikerer den beregnede SMS-ophidselse, mens den røde stiplede er gennemsnittet af SMS-ophidselsen. Den blå viser brugerens indtastede ophidselse, mens den stiplede er gennemsnittet.

Med ophidselsen er brugerens data og de udregende værdier fra SMS beskederne endnu mere forskudt end dem for valensen. Gennemsnittet for for SMS-ophidselsen er 1,7 større end bruger-ophidselsen, hvor bruger-valensen er 1,2 større end sms-valensen , til sammenligning. For ophidselsen er der altså større forskydelse i data, men de to grafer bevæger sig mere synkront end graferne for valensen. På samme måde som med valensen ses altså nogle klare tendenser i udsvingene for de to grafer samt.



Figur 3-9: Beregnet SMS-valens og bruger-ophidselse som funktion af tid.
Forklaring: Rød - bruger-ophidselse, Rød stiplet - bruger-ophidselse gennemsnit, Blå - SMS-ophidselse, Blå stiplet - SMS-ophidselse gennemsnit.

Desværre er der ikke indsamlet en tilfredsstillende mængde musik data til at analysere og sammenholde de indførte humørværdier. Derfor må der på baggrund af de indsamlede SMS data konkluderes, at der ved den benyttede metode ikke opnås tilstrækkelig nøjagtighed ved bedømmelse af brugeres humør til at afspejle virkeligheden. De værdier der opnås har dog vist sig forholdsvist at følge brugerens indtastede værdier med en vis forskydelse, hvilket giver anledning til at konkludere, at hvis der kunne opnås større præcision ved beregning af valens og ophidselse, ville graferne konvergere overlappe.

Ved en større dataindsamling vil det måske være muligt at udregne et samlet sæt af valens- og ophidelsesværdier ud fra musik, lokationer, SMS beskeder og tidspunkt, med øget præcision.

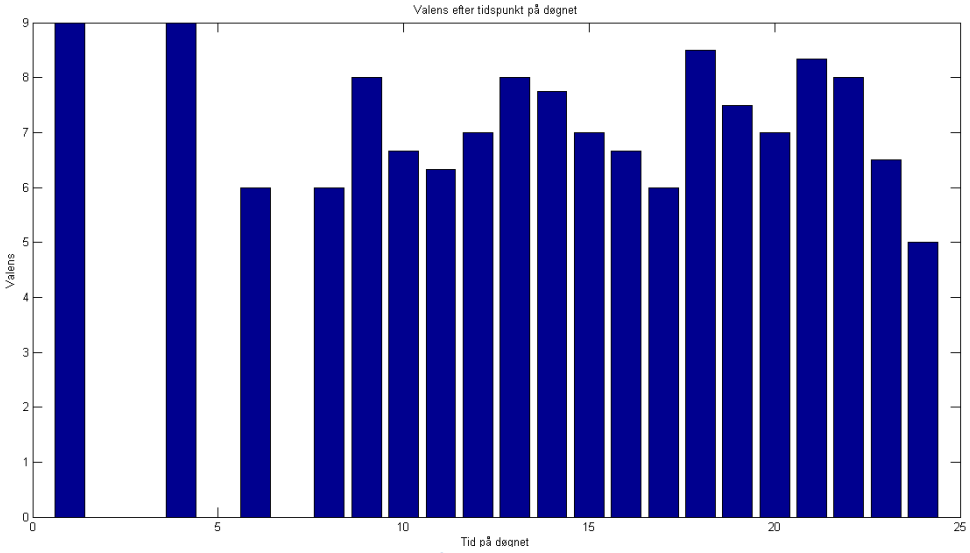
3.4.2 Adfærdsmønstre

I dette afsnit vil der blive kigget på brugernes adfærdsmønstre i forhold til hvilket humør de er i, afhængigt af tid og placering. Først vil humør- og ophidselsesfordelingen efter tid på døgnet blive gennemgået med mine egne indsamlede data, hvorefter de studerendes data vil blive brugt samlet. Grunden til de studerendes data benyttes som et samlet gennemsnit, er at deres indsamlede data, hver for sig er for mangelfulde. Derefter vil humør- og

ophidselsesfordelingen efter placering blive undersøgt efter samme fremgangsmåde som med tiden.

3.4.2.1 Stemning fordelt på døgnet - egne data

I dette afsnit benyttes mine egne indsamlede data til at vurdere stemningen i forhold til tiden på døgnet. Alle stemningsinput fra er sorteret efter hvad tid på døgnet de er blevet indført og derefter er gennemsnittet af summen for hver time i døgnet blevet udregnet. Om valensen skal det nævnes, at der i indsamlingsperioden af private årsager blev indført høje valensværdier, så gennemsnittet vil ligge lavere under normale omstændigheder.



Figur 3-10: Valens fordelt efter tidspunkt på døgnet.

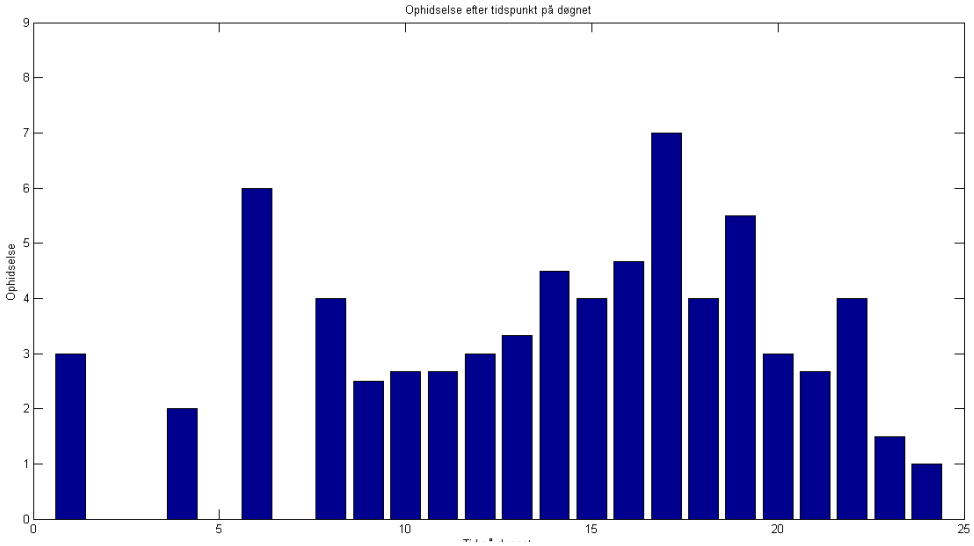
Her på *Figur 3-10* ses min gennemsnitlige valens fordelt ud på døgnets 24 timer. På grafen kan der ses en svingende valens i løbet af dagen, som er er lav tidligt om morgenen omkring kl. 6, hvorefter det gradvist stiger indtil kl. 13, falder igen til kl. 18, hvor det ligger nogenlunde stabilt til det falder igen efter kl. 21-22.

De eneste egentlige tendenser der kan ses ud fra den ovenstående figur er, at dagen startes med en lav valens, som stiger i løbet af eftermiddagen og i slutningen af dagen falder humøret stødt fra sen eftermiddag indtil midnat.

På *Figur 3-11* ses en stigende kurve fra kl. 9 til kl. 17, hvorefter den er stødt og roligt faldende til i løbet af aftenen.

Der er især to tidspunkter der skiller sig ud på grafen, nemlig kl. 6 og kl. 8, hvilket skyldes en arbejdsrutine, hvor der mødes på arbejde kl. 6 om morgenen og der er fysisk aktivt arbejde til omkring kl. 9. Da jeg normalt ikke er vågen før kl. 9, medmindre jeg er på arbejde, giver det disse afbrydelser i grafen.

Bortset fra de beskrevne anormaliteter ses af grafen en stigende ophidselse fra der stås op til omkring aftensmaden, hvorefter der igen slappes mere og mere af.



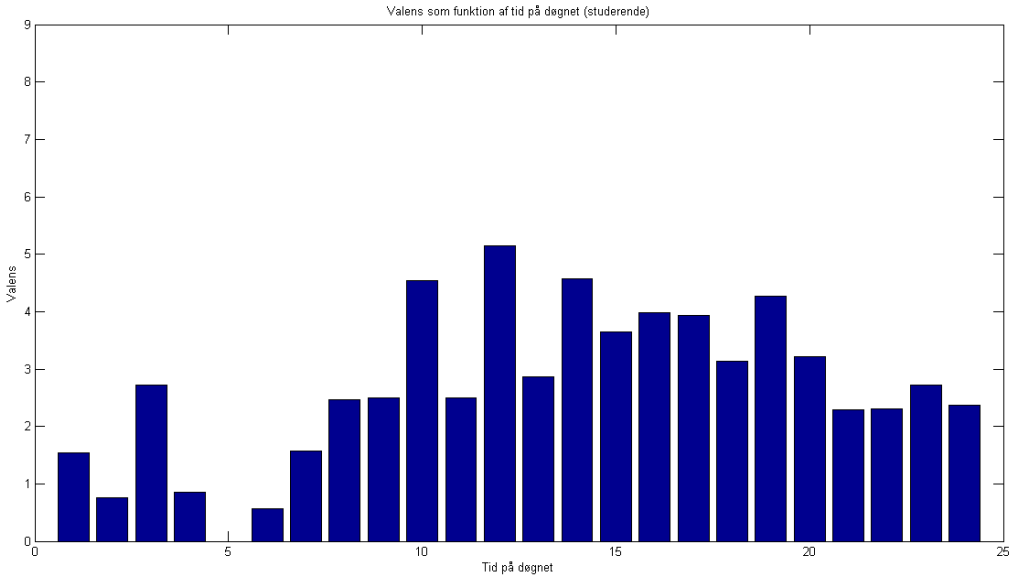
Figur 3-11: Ophidselse fordelt efter tidspunkt på døgnet.

Ud fra stemningen fordelt ud på døgnets 24 timer for mine egne data vurderes, at der kan observeres nogle klare stemningsmæssige tendenser. Disse tendenser afspejler meget hvad man kunne forestille sig for stemningen i løbet af dagens forløb og viser altså, at det vil være muligt at tilpasse de beregnede data fra for eksempel SMS beskeder, ved at justere de fundne værdier alt efter hvad tid de bliver beregnet.

3.4.2.2 Stemning fordelt på døgnet - studerendes data

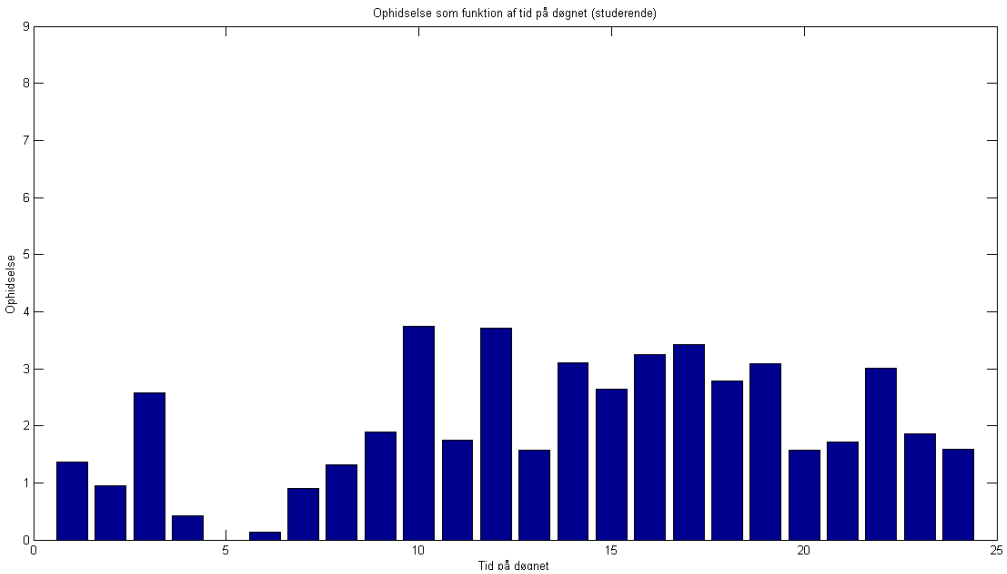
Her vil gennemsnittet af de indsamlede data fra de studerende blive benyttet til at vurdere stemningen i forhold til tid på døgnet. På ***Error! Reference source not found.*** ses fordelingen af de studerendes gennemsnitlige valens.

Bortset fra nogle enkelte høje udslag er kurven for de studerendes valens langt simplere end den for mine egne data. De studerendes valens starter meget lavt kl 6, hvorfra den stiger til kl. 19, hvor kurven brydes og valensen falder stødt igen indtil kl. 6. Dette tyder altså på at de studerendes humør er stødt stigende fra morgenstunden af og stiger indtil omkring aftensmaden, hvorefter den falder i løbet af natten. Dette viser tegn på, at humøret der er indsamlet er påvirket af de studerendes træthed.



Figur 3-13: Valens efter tid på døgnet (studerende)

De studerendes samlede ophidselse i løbet af døgnet ses på *Figur 3-12*. Det første der umiddelbart ses, er at de to grafer minder for valens og ophidselse minder meget om hinanden, med en tydelig kurve der har nogle enkelte afvigelser. Med ophidselsen stiger kurven også fra kl. 6, men er dog fladere end den for valensen. Det ses, at de studerendes ophidselse topper omkring kl. 16, hvorfor den falder gennem natten til kl. 6.



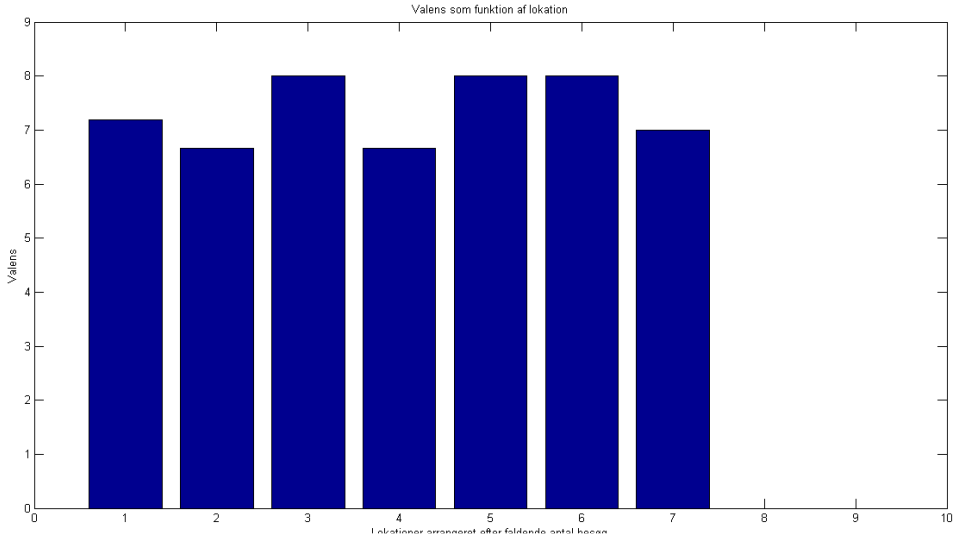
Figur 3-12: Ophidselse efter tid på døgnet (studerende)

På samme måde som med mine egne indsamlede data bekræftes der her, at der kan ses nogle specifikke tendenser i stemningsafgivelsen. Disse vil kunne benyttes til mere præcist, at vurdere brugerens sindstilstand på baggrund af analyse af kontekstdata.

3.4.2.3 Stemning på mest besøgte lokationer - egne data

Her bliver de indsamlede GPS-lokationer inddelt i klynger af de placeringer der ligger tæt på hinanden. Derefter optælles, hvor tit brugeren har befundet sig i hver af disse klynger og derefter sorteres de efter mest besøgte. Derefter sammenlignes tidspunkt for stemningsinput og GPS-lokationerne i de oprettede klynger. En gennemsnitlig valens og ophidselse udregnes så for hver af klyngerne.

Herunder på *Figur 3-14* ses de mest besøgte lokationer og den gennemsnitlige valens på disse lokationer. De er sorteret efter faldende antal besøg fra venstre mod højre.



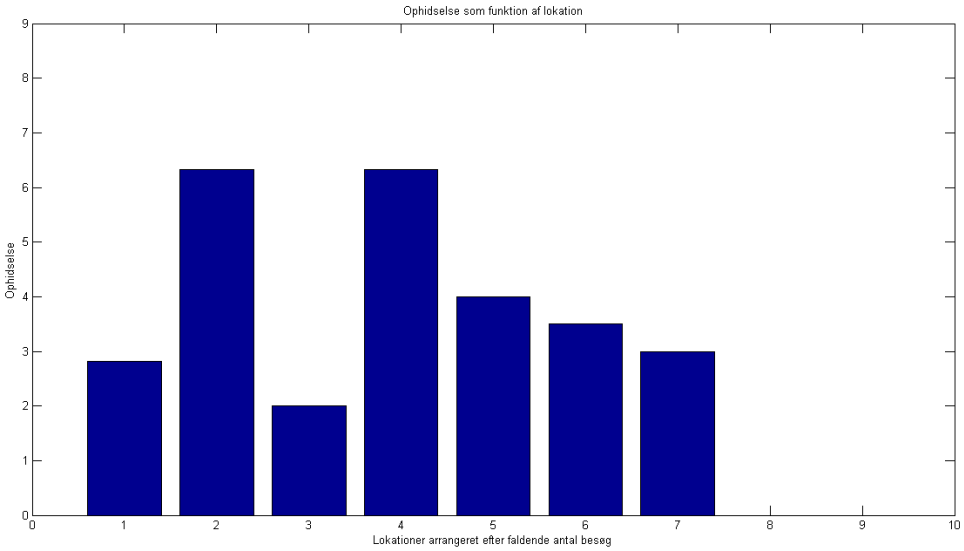
Figur 3-14: Valens på mest besøgte lokationer - egne data. Sorteret i faldende orden fra venstre mod højre

Der er ikke umiddelbart nogen tendens at spore ved at analysere grafen for valens efter mest besøgte lokationer. Selv om der ikke kan konkluderes nogen egentlig tendens fra følgende, kan der sættes placering på de fire første lokationer ud fra de tilhørende GPS-koordinater. Ud fra GPS-koordinaterne er lokationerne

1. Hjem
2. Arbejde
3. Forældres hjem
4. Træningscenter

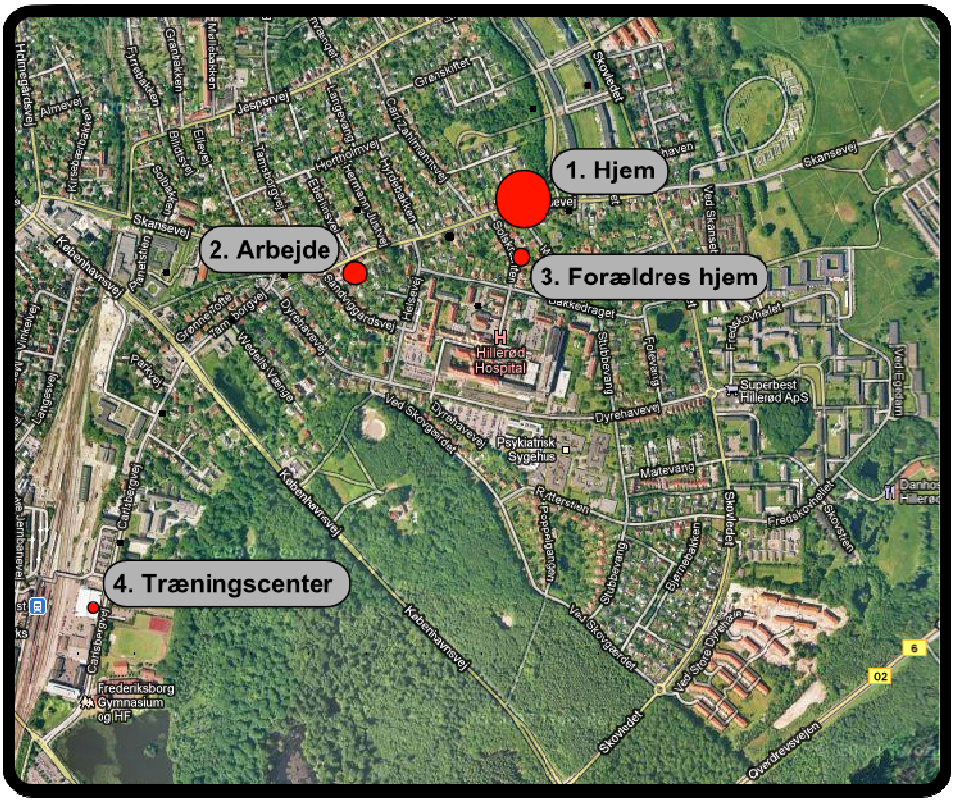
de efterfølgende 3 placeringer ikke har nogen særlig betydning. Det kan ses, at humøret er lavest på arbejdet og i træningscentret, samtidig med det er højest i forældrenes hjem, hvilket giver en form for mening.

Ved at se på *Figur 3-15*, med placeringernes personlige betydning for øje, ses at ophidselsen i forældrenes hjem er lavest, med en lidt højere ophidselse i eget hjem. På arbejdet og i træningscentret er humøret præcist lige højt.



Figur 3-15: Ophidselse på mest besøgte lokationer - egne data. Sorteret i faldende orden fra venstre mod højre

På *Figur 3-16* ses de fire mest besøgte lokationer i centrum af cirklerne, hvis radius er proportionel med antallet af besøg. Desværre var der ikke nogen tendens der ville kunne generaliseres ud fra mine egne indsamlede data.

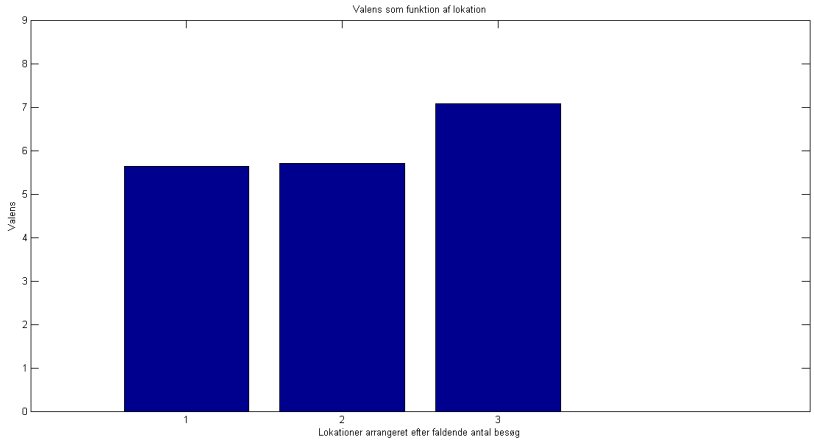


Figur 3-16: De fire mest besøgte lokationer angivet i størrelse efter antal besøg.

Selv om værdierne stemte godt overens med de forskellige placeringer. Nu vil den samme analyse blive foretaget på de studerendes data.

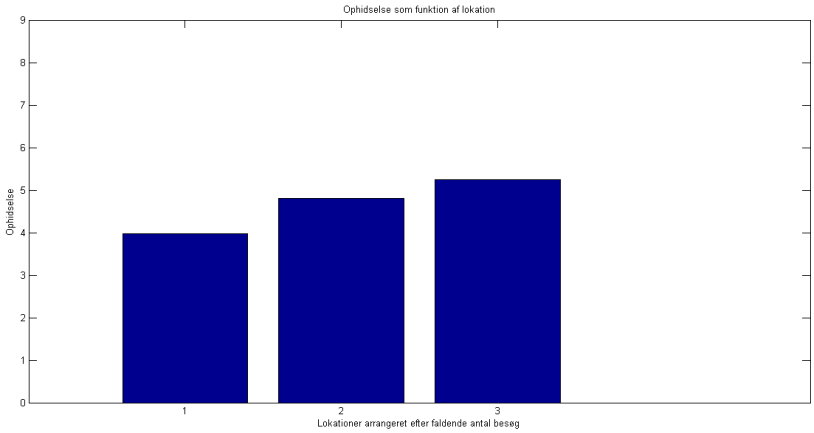
3.4.2.4 Stemning på mest besøgte lokationer - studerendes data

Da de studerendes data er sparsomme er der kun tilstrækkelig stemningsinput til, at vurdere de tre mest besøgte lokationer. På Figur 3-17 ses de studerendes valens sorteret ud på de tre mest besøgte lokationer. Det ses, at valensen stiger på de mindre besøgte lokationer, en smule fra den første til den anden og dereft er stiger valensen over 15% til den sidste lokation.



Figur 3-17: Valens på mest besøgte lokationer - studerendes data. Sorteret i faldende orden fra venstre mod højre.

Det er en interessant tendens, at humøret skulle stige på de lokationer, hvor de studerende er mindre. Selv om det ikke er nogen stor stigning fra den først til den anden mest besøgte lokation, stiger humøret mod de mindre besøgte lokationer



Figur 3-18: Ophidselse på mest besøgte lokationer - studerendes data. Sorteret i faldende orden fra venstre mod højre.

Her er det desværre ikke muligt, at sætte en titel på placeringer, men umiddelbart må man antage, at den første lokation er hjemmet, mens de to næste kan være studiet, arbejdet, kærestens hjem osv. Idet der er tale om studerende, kunne man forestille sig, at den anden lokation er studiet, hvor den tredje så kan være hvilken som helst af de resterende grupper.

Den samme tendens der ses for de studerendes valens, ses også på grafen for de studerendes ophidselse på Figur 3-18. Her stiger ophidselsen dog mere markant fra den første til den anden lokation og det er altså en mere ensformig stigning som der gås mod de mindre besøgte lokationer.

På de studerendes data, kunne der altså ses en tendens til at deres valens og ophidselse steg i takt med de kom ud til deres anden og tredje mest besøgte lokationer. Dette kunne give en god hjælp til at vurdere de beregnede stemningsværdier, hvis det kunne fastslås som en grundlæggende tendens, men det er der desværre ikke nok data til at kunne retfærdiggøre.

3.5 Konklusion på case

Med hensyn til kontrol af de valens og ophidselsesværdier, der blev beregnet ud fra de indsamlede SMS beskeder, var der en stor afvigelse fra de værdier som brugerne havde indtastet. Den udregnede valens ligger generelt for lavt, mens den udregnede ophidselse ligger for højt. Der kunne dog ses lignende udsving på kurverne for de udregnede og indførte valensværdier, hvilket giver anledning til at tro, at disse to kurver kunne komme tættere på at ligne hinanden med tilpasning på baggrund af generelle vaner og tilbøjeligheder. Det samme gælder for grafen for beregnet og indført ophidselse.

Ved at benytte mere avancerede analysemetoder af kontekstdata ville det formegentlig kunne lade sig gøre, at opnå en større præcision, hvis der samtidig blev inkluderet data fra flere sensorer i vurderingen.

Under analyse af adfærdsmønstre kunne det se, at der er en tendens til lav valens og ophidselse tidligt om morgenen og faldende valens og ophidselse hen af aftenen. I løbet af dagen er det svært at sige noget præcist grundet den sparsomme mængde indsamlede data.

Med hensyn til humør givet lokationer er det ikke til at se nogen specifik tendens, men en forbedring af dette kunne være, at applikationen i stedet for at bruge generelle grupper, brugte autogenerede grupper ud fra, hvilket humør brugeren har været i tidligere på forskellige lokationer.

Det kan altså konkluderes, at det angiveligt er muligt at vurdere brugerens humør ved at indsamle kontekstdata, men alt tyder på beregningen af stemningen skal raffineres og tilpasses brugerens adfærdsmønstre.

Kapitel 4

4 Anvendelse af analysemetoder

I dette afsnit vil de analysemodeller der blev beskrevet i *2 Analyse* blive benyttet til at analysere forskellige former for brugergeneret data, herunder noget af det data der er indsamlet i casen der er beskrevet i *Case - Indsamling og analyse af mobil kontekstdata*.

Der vil være et underafsnit for hver af de fire analysemodeller og disses anvendelsesmuligheder og begrænsninger vil blive diskuteret. Desuden vil modellerne løbende blive vurderet for deres funktionalitet og afkastet af deres analyser vil blive behandlet.

4.1 ANEW

ANEW samlingen som allerede er benyttet i casen, vil her blive anvendt brugergenererede filmanmeldelser i et forsøg på, at vurderer stemningen af disse. Derudover vil der blive introduceret et projekt som allerede har benyttet netop ANEW-samlinge til at vurderer store mænger af tekst.

4.1.1 Anvendelse på case

ANEW samlingen primære anvendelse er, at gennemsøge tekster for de cirka tusind ord der findes i listen. På baggrund af de ord der bliver genkend kan tekstens følelsesmæssige ladning vurderes ud fra den ANEW score der opnås. Det er på denne måde hvorpå ANEW samlingen allerede er blevet anvendt i casen til at vurdere stemningen i SMS beskeder og musik tags.

4.1.2 Begrænsning for case

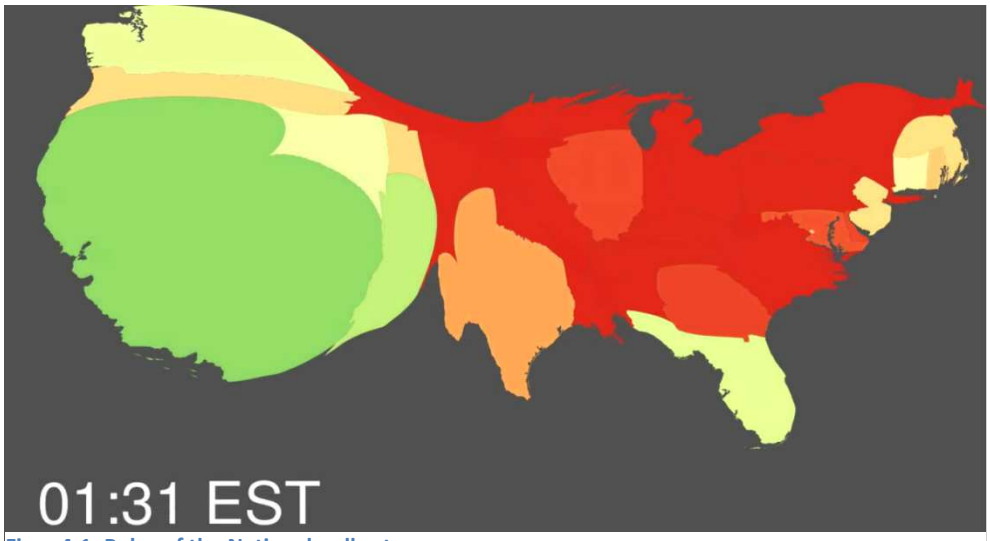
Brugen af ANEW samlingen er hovedsageligt begrænset af to faktorer, nemlig at tusind ord ikke er særlig mange og derfor ikke giver en særlig bred dækning når meget små tekster som sms beskeder, instant messaging beskeder eller tags gennemgås.

Derudover er metoden alt for simpel idet at en tekst der analyseres ved at finde ord fra ANEW samlingen blot vurderes ord for ord uden at tage højde for sammenhængen i teksten.

De begrænsninger der ligger i at benytte ANEW samlingen til tekst analyse, kan reduceres ved at tage nogle af de teknikker i brug der beskrives senere i dette afsnit.

4.1.3 Twitter analyseret med ANEW

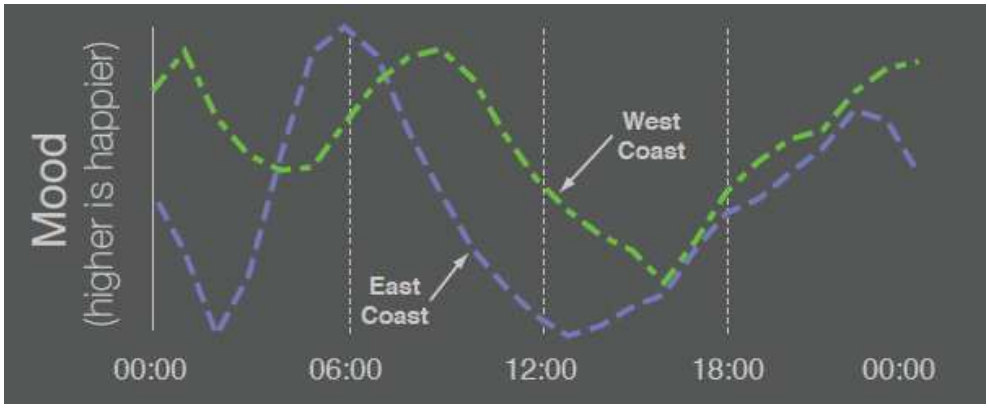
Et projekt gennemført af forskere for Northeastern University og Harvard, har indsamlet Twitter [**9**] beskeder kaldet *Tweets* fra USA og på den måde afbilledet forløbet i USA's humør afhængigt af tiden på døgnet [**10**]. Dette afbildedes på en helt genial måde ved, at farve USA's stater fra rød til grøn alt efter det afledte humør, fra trist til glad henholdsvis. Derudover varierer størrelsen af staterne på kortet alt efter, hvor stor en del af det samlede antal *tweets* der kommer fra den enkelte stat. Dette vises på en video der kan findes på projektets hjemmeside samt på YouTube [**11**]. Et billedet fra videoen ses herunder, der viserkortet for USA's humør afledt af Twitter beskeder, kl. 01:31 Eastern Standard Time, EST.



Figur 4-1: Pulse of the Nation, landkort.

Det kan ses, at der på dette tidspunkt kommer mange flere *Tweets* på vestkysten end på østkysten og i midten af landet. Ud fra farven kan man også tydeligt tolke at humøret på vestkysten er bedre end på østkysten, hvor det er i den glade ende af skalaen, mens det er i den triste ende i stort set resten af landet. Dette kunne skyldes, at klokken kun er 22.31 på vestkysten, mens det er midt om natten på østkysten, hvor de anvender EST.

På en graf der er fremstillet på den poster forskerne har udviklet til at forklare projektet, ses en meget interessant tendens. På Figur 4-2 ses denne graf, hvor humøret for vestkysten og østkysten følger hinandens svingninger, blot forskudt af cirka tre timer, hvilket udgøre tidsforskellen.



Figur 4-2: Pulse of the Nation, graf.

Beregningerne der er lavet i projektet, udført i samarbejde mellem Harvard og Northeastern University, er foretaget på 300 millioner Twitter-opdateringer, der er indsamlet i tidsrummet 2006-2009.

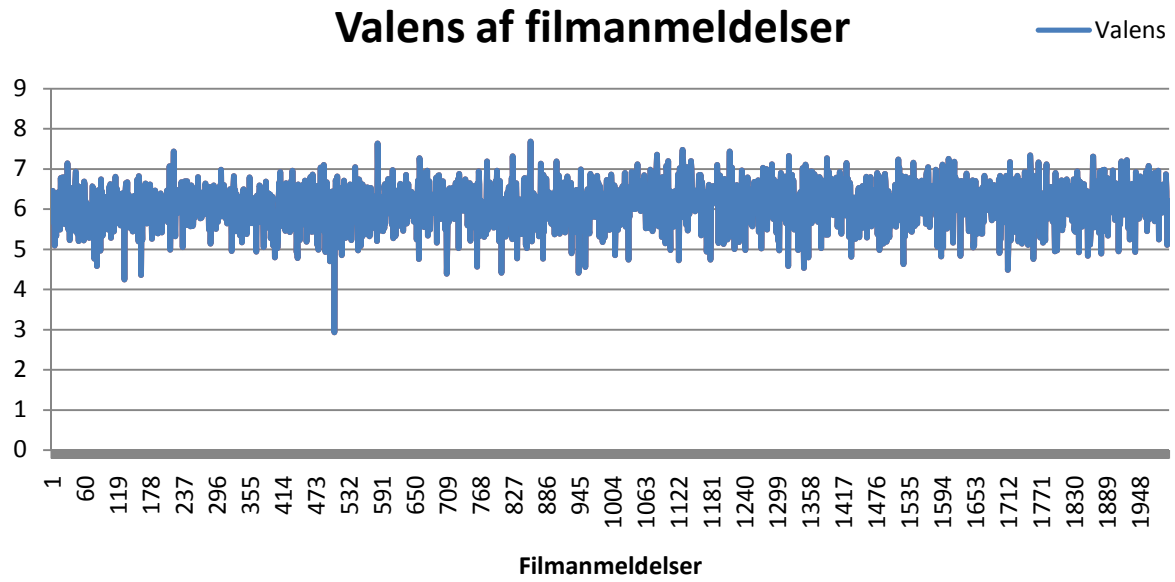
Projektet er et glimrende eksempel på, hvordan man kan udvinde nogle meget interessante oplysninger, ved at benytte et simpelt værktøj som ANEW, når man har tilstrækkelig med data og en god idé. En del af det der er med til at gøre projektet storartet, er den geniale visuelle måde, hvorpå det svingende humør er vist på videoen, der på en flot og letforståelig måde illustrer de beregnede data.

4.1.4 ANEW undersøgelse af filmanmeldelser

I dette afsnit vil der blive foretaget en analyse af 2000 filmanmeldelser, der er inkluderet i *Natural Language Toolkit*, som vil blive forklaret i større detaljer i *4.2.3 Natural Language Toolkit*. De 2000 anmeldelser inddelt sådan, at de 1000 første er negative, mens de 1000 sidste er positive. Det skal nu efterprøves, om søgning efter ord i ANEW samlingen samt tildeling af scorer efter fundne ord vil være i stand til at inddele anmeldelserne korrekt. Udregning af valensscore er lavet på baggrund af anbefalingerne i [**12**].En filmanmeldelse antages at være negativt ladet, hvis den gennemsnitlige ANEW valens ligger under 5, da skalaen går fra 1 til 9. Herunder ses en de statistiske tal for optællingen, koden kan findes vedlagt efter *7 Appendix*, som *movieMood.py*

Negative folder
Correct count: 26
Found in files: 1000
Average: 6,02363453734
Positive folder
Correct count: 999
Found in files: 1000
Average: 6,15723766729

Det kan desværre ses, at ud af de 1000 negative filmanmeldelser, vurderes kun 26 til at være negative, ved at have en samlet valens under 5. Med hensyn til de positive filmanmeldelser er 999 vurderet positivt, hvilket tyder på, at vurderingen der er givet med ANEW samlingen ligger for højt generelt. På grafen herunder ses den udregnede valens per filmanmeldelse, hvor de første 1000 er negative og de sidste 1000 er positive



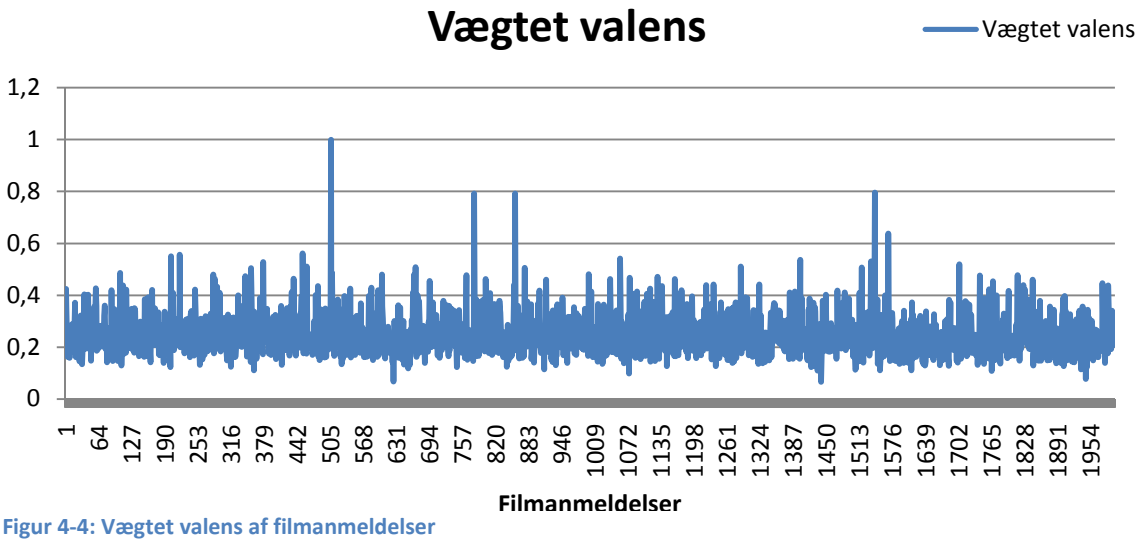
Figur 4-3: Valens af filmanmeldelser.

Det fremgår tydeligt af grafen, at den gennemsnitlige vurdering af de negative og positive filmanmeldelser ligger meget tæt på hinanden, hvilket også kan se på de udregnede statistikker. Ud fra ANEW analysen her kan det altså ikke påvises at ANEW modellen kan estimere humøret, af i hvert fald denne form for tekster, korrekt.

Fra en antagelse om, at forekomster af enkelte ord kunne gå igen over hele samlingen af anmeldelser, og på den måde ensforme resultatet, er der lavet en tf-idf vægtning af filmanmeldeserne som korpus. Tf-idf er forklaret under *2.4.2.2 Term frekvens-invers dokument frekvens*. Dette blev gjort i forhåbningen om, at en vægtning af ordene kunne give en mere nuanceret afbildning af teksterne. De nye værdier er således den fundne valens ganget med ordets vægtning. Med tf-idf vægtningen går, hvert ord kun igen én gang i en tekst, da ordets vigtighed fremgår af dets vægt. Koden hvormed vægtningen og udregningen er lavet findes i appendix.

På *Figur 4-4* ses det, at vægtningen med tf-idf desværre ikke har givet et bedre billede af filmanmeldelsernes indhold. Dette betyder, at den ensartede gennemgående valens ikke skyldes ord der genbruges ofte gennem hele korpusset, men nærmere de ord der er benyttet i hver anmeldelse har nogenlunde samme vigtighed i forhold til hele korpusset.

Det er dog interessant at se, hvordan det store negative udslag på *Figur 4-3* i filmanmeldelse 507 (set på de originale data) giver et stort positivt udslag på *Figur 4-4* i anmeldelse nummer 507. Dette skyldes, at ordet der har givet udslaget i de ikke vægtede data, åbenlyst har været sjældent i forhold til korpusset, hvilket derved har givet det en meget stor vægtning. Derved har den ellers lave valens ikke blevet skaleret nær så meget ned som de andre ord i korpusset og kommer derfor til at fremstå som positiv.



Figur 4-4: Vægtet valens af filmanmeldelser

Det kan desværre konkluderes, at det ikke var muligt, at vurdere den semantiske betydning af de 2000 filmanmeldelser ved hjælp af ANEW samlingen.

Heldigvis findes der en anden analysemodel der er meget velegnet til dette, nemlig dokument klassifikation. Denne er beskrevet i næste afsnit.

4.2 Dokument klassifikation

Dokument klassifikation baseret på en naiv Bayes klassifikator er meget velegnet til, at vurdere input af en kendt struktur og tildele det et label, på baggrund af de labels klassifikatoren er blevet trænet på.

4.2.1 Anvendelse på case

I forbindelse med casen i dette projekt er dokument klassifikation en mulighed for at hjælpe med at vurdere den følelsesmæssige ladning af en tekst. Dokument klassifikation kan ikke give en nuanceret beskrivelse af et dokuments humør, men blot indikere om det er positivt eller negativt. Siden det på forhånd er bestemt, hvilke klasser vi ønsker vores tekster inddelt i, vil en overvåget klassifikation åbenlyst egne sig bedst.

For at træne en klassifikator til at vurdere en tekst efter dens humør kræver det en stor mængde tekster af samme art som kan undersøges for specielle træk og egenskaber først. For at disse træk og egenskaber kan tilknyttes en bestemt stemning kræver det derudover også at de tekster som klassifikatoren er trænet på, på forhånd er inddelt i de specifikke kategorier.

4.2.2 Begrænsning for case

Dokument klassifikation ville i teorien kunne benyttes til såvel SMS beskeder som sangtekster. Problemet er imidlertid, at der ikke på nuværende tidspunkt ikke er muligt at finde et korpus med hverken humørinddelte sangtekster eller SMS beskeder og det er udenfor omfanget af denne afhandling, at udarbejde sådanne korpora.

I stedet kan der ved at foretage dokument klassifikation af filmanmeldelser udtrækkes de mest informative features, og dem af disse der kan overføres fra film-domænet til SMS-domænet kan eventuelt benyttes i ANEW samlingen

4.2.3 Natural Language Toolkit

Natural language Toolkit er en samling open-source moduler udviklet til python, der benyttes til mange forskellige former for Natural Language Processing, NLP. Til NLTK hører udover moduler til statistisk og lingvistisk behandling af tekster også forskellige korpora der kan benyttes sammen med modulerne.

Ydermere følger der en gratis bog med NLTK, som gennemgår de fundamentale principper indenfor NLP og giver eksempler på hvordan NLTK kan benyttes [**13**].

4.2.3.1 Dokument klassifikation med NLTK

Formålet med denne tutorial er at vise hvad dokument klassifikation kan bruges til og redegøre for hvordan opgaven kan udføres ved hjælp af NLTK's moduler.

I tutorialen benyttes et korpus på 2000 filmanmeldelser, hvoraf 1000 er positive og 1000 er negative. En klassifikator trænes ved at udtrække de egenskaber, som kendetegner om en anmeldelse er positiv eller negativ. Derefter kan klassifikatoren benyttes til at klassificere filmanmeldelser og de mest informative egenskaber kan udledes.

Først importeres modulerne nltk og random. Derefter importeres filmanmeldelserne og der oprettes en liste hvor ordene fra hver anmeldelse er opbevaret sammen med den klasse, negativ eller positiv, de tilhører. Listen hvori anmeldelserne er gemt bliver derefter tilfældigt blandet rundt, dette vil blive forklaret senere

```
>>> import nltk, random
>>> from nltk.corpus import movie_reviews
>>> documents = [(list(movie_reviews.words(fileid)),category)
...     for category in movie_reviews.categories()
...     for fileid in movie_reviews.fileids(category)]
>>> random.shuffle(documents)
```

Nu oprettes der hvad der i Python kaldes en *dictionary*, altså en ordbog, hvor samtlige ord der bliver benyttet i anmeldelserne er gemt sammen med antallet af forekomster. Denne ordbog er sorteret i rækkefølge efter faldende hyppighed, så der oprettes en liste bestående af de 2000 hyppigst forekommende ord.

Derefter oprettes en funktion der undersøger, om de 2000 ord der findes i et givent dokument, dette kaldes en *feature extractor*.

```
all_words = nltk.FreqDist(w.lower() for w in
movie_reviews.words())
>>> word_features = all_words.keys()[:2000]
>>> def document_features(document):
...     document_words = set(document)
...     features = {}
...     for word in word_features:
...             features['contains(%s)'%word]=(word in
document_words)
...     return features
```

Når der nu er defineret en *feature extractor* oprettes en liste kaldet *featuresets* med features, der viser hvilke af de hyppigst forekommende ord der findes i hvert dokument, altså de bliver markeret fundet eller ikke fundet, og hvilken klasse dokumentet hører til, altså positiv eller negativ.

Herefter oprettes et træningssæt til at træne klassifikatoren med, som består af 1900 sidste poster i featuresets. De første 100 poster fra listen lægges i en testliste som klassifikatoren kan testes på, da der ikke må testes på dokumenter som er blevet brugt til at træne med. Nu kan det ses hvorfor dokumenterne skulle blandes tilfældigt tidligere, netop for at give en tilfældig sammensætning af træningssættet og testsættet.

Derefter trænes en Naive Bayes klassifikator på træningssættet og dens præcision måles herefter på testsættet.

```
>>> featuresets = [(document_features(d), c) for (d,c) in
documents]
>>> train_set, test_set = featuresets[100:], featuresets[:100]
>>> classifier = nltk.NaiveBayesClassifier.train(train_set)
>>> print nltk.classify.accuracy(classifier, test_set)
0.85
```

Her opnås 85% procent præcision af klassifikatoren, men det skal nævnes at præcisionen kan varierer selv om forsøges gengives nøjagtig som her. Indholdet af træningssættet og testsættet er jo ikke identisk fra gang til gang. Der er under udførslen af denne tutorial blevet observeret mellem 81% og 86% præcision.

Som det ses herunder kan klassifikatoren nu bruges til at udskrive de features der er fundet mest informative.

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
contains(outstanding) = True   pos : neg    =     13.5 : 1.0
contains(mulan) = True         pos : neg    =      8.4 : 1.0
contains(wonderfully) = True   pos : neg    =      7.7 : 1.0
contains(seagal) = True        neg : pos    =      7.0 : 1.0
contains(damon) = True         pos : neg    =      5.9 : 1.0
```

Det kan altså ses, at en anmeldelse der nævner (Steven) *seagal*, har 7 gange større sandsynlighed for at være negativ end positiv, mens en anmeldelse der nævner (Matt) *damon* kun vil være negative 1 ud af 6(5,9) gange.

Udover navnene på de skuespillere der bliver omtalt i filmanmeldelserne kan det ses, at der også er nogle mere generelle ord, der har betydning for anmeldelsens klassifikation.

4.2.4 Domæneoverførsel af features

Her vil der blive fundet de mest informative features ved dokument klassifikation af filmanmeldeser, og det vil blive vurderet hvilke af disse features der kan overføres til SMS-domænet.

Først benyttes korpuset der er gjort tilgængeligt via Natural Language Toolkit, NLTK, bestående af 2000 film anmeldelser, hvor 1000 er positive og 1000 er negative. Når en klassifikator er blevet trænet på denne samling, kan de mest sigende egenskaber uddrages. Fra disse vurderes så hvilke der kan overføres fra domænet film anmeldelser til noget mere generelt som SMS beskeder. Derved opnås så en række ord, som kan benyttes til at vurdere om en tekst er positivt eller negativt ladet.

Det er nogle af disse ord, der forhåbentlig kan overføres fra film-domænet til SMS-domænet, så de kan hjælpe med at analysere SMS beskeder. At features kan overføres fra mellem domæner er præsenteret og testet med en multi-domæne klassifikator i [**14**]

For at udvinde så mange brugbare features som muligt, er det netop gennemgåede eksempel ændret til at benytte samtlige features fra filmanmeldelserne. Altså hver post i featuresættet indeholder alle ord der bliver benyttet udover den label der hører til det enkelte dokument. Derefter er naiv Bayes klassifikatoren trænet på hele featuresættet. Så er den samme funktion benyttet som tidligere til at udskrive de mest informative ord, denne gang blot med 50 i stedet for 5. De ord de er udtaget, er valgt på baggrund af at de har en lille eller ingen tilknytning til film-domænet per definition, i modsætning til navne på skuespillere for eksempel. De ord der er udtaget er vist her og alle de 50 mest informative features kan ses i *7.1 Appendix I - 50 mest informative features*

astounding	pos : neg	=	12.3 : 1.0
outstanding	pos : neg	=	11.5 : 1.0
ludicrous	neg : pos	=	11.0 : 1.0
insulting	neg : pos	=	11.0 : 1.0
sucks	neg : pos	=	10.6 : 1.0
stupidity	neg : pos	=	9.0 : 1.0
wasting	neg : pos	=	8.3 : 1.0
finest	pos : neg	=	8.1 : 1.0

Dette er altså, de 8 af de mest informative features, som er vurderet til at kunne overføres fra film-domænet til SMS-domænet. De fire nærmeste ord for hvert af featuresne er vidst herunder. De fundet ved at benytte *en-til-mange* LSA med 150 dimensioner på Colorado University's online LSA service [**15**]. Antallet af dimensioner er valgt ved at efterprøve forskellige dimensioner for de forskellige ord og vurdere, hvilket antal gav det bedste resultat.

I *Tabel 1* herunder ses denærmeste naboer for de 8 udvalgte features. Ord i **fed** i tekst er de ord fra tabellen der er blevet valgt som relaterede ord.

astounding	***oustanding***	***ludicrous***	***insulting***
curious 0.52	**interest** 0.62	passion 0.57	**hatred** 0.53
gossip 0.53	talent 0.58	virtue 0.51	**rage** 0.51
contempt 0.51	**devoted** 0.56	scorn 0.50	fear 0.51
betray 0.49	fame 0.56	misery 0.50	**disgusted** 0.49
sucks	***stupidity***	***wasting***	***finest***
stomach 0.48	**absurd** 0.64	alone 0.57	**admired** 0.71
infant 0.47	**indifferent 0.63**	pity 0.57	**beautiful** 0.70
mucus 0.45	curious 0.62	hope 0.56	famous 0.70
nourish 0.42	false 0.61	grateful 0.55	**beauty** 0.67

Tabel 1: Informative features og de nærliggende LSA svar

De relaterede ord samles og deres gennemsnitlige valens- og ophidselsesværdier benyttes for de features der er taget fra filmanmeldelserne som vist i *Tabel* 2.

	Valens	Ophidselse			Valens	Ophidselse
interest	*6,97*	*5,66*		*absurd*	*4,26*	*4,36*
devoted	*7,41*	*5,23*		*indifferent*	*4,61*	*3,18*
outstanding	**7,19**	**5,45**		**stupidity**	**4,44**	**3,77**
hatred	*1,98*	*6,66*		*admired*	*7,74*	*6,11*
rage	*2,41*	*8,17*		*beautiful*	*7,6*	*6,17*
disgusted	*2,45*	*5,42*		*beauty,*	*7,82*	*4,95*
insulting	**2,28**	**6,75**		**finest**	**7,72**	**5,74**

Tabel 2: Genererede ANEW ord fra informative features

Det kan konkluderes, at man med denne metode kan dannes nye ord til ANEW samlingen, hvor man har en god indikation af, at ordene er fremstående når det kommer til, at vurderer stemningen af konteksten.

Derudover er det meget interessant at se, hvordan de relaterede ord der er fundet ved LSA har valens- og ophidselsesværdier der ligger meget tæt på hinanden. Dette fungerer som en form for bevis af begge modeller, da LSA har bestemt ord, der burde være semantisk i nærheden af hinanden, hvilket er blevet bekræftet af ANEW værdierne, som er menneskeligt bestemt. Modsat bliver ANEW samlingen bekræftet, da LSA på baggrund af et stort tekstkorpus har vurderet ordene relaterede, hvilket godkender, at ordene har nærliggende ANEW værdier.

4.3 Latent Dirichlet Allokation

I dette afsnit vil der blive foretaget håndgribelige tests af LDA på forskellige former for brugergenereret data. Først vil anvendelsen af LDA i forhold til casen blive diskuteret efterfulgt af en begrænsning for metodens anvendelse.

4.3.1 Anvendelse på case

Da Latent Dirichlet Allokation resulterer i emner der beskriver den analyserede data, vil LDA kunne benyttes til at reducere dimensionerne af en term-dokument matrice bestående af SMS beskeder for på den måde at koge essensen ned til nogle få emner. Disse emner ville så kunne analyseres på forskellige måde. Ud fra emnerne vil det være muligt at se, hvilke ord der har meget betydning for den underliggende latente betydning. Disse ord kan så eventuelt vægtes højere ved analysen af SMS beskederne med for eksempel ANEW samlingen.

4.3.2 Begrænsning for case

En åbenlyst begrænsning af er for det første, at udregning af de probabilistiske emner kræver meget meget processorkraft og nye SMS beskeder skal derfor kontinuerligt sendes til en server for at blive analyseret. Desuden kræves der en tilstrækkelig samling af SMS beskeder, før LDA kan give nogle sigende emner. Ved programmets etablering

4.3.3 Opdeling af SMS beskeder i emner

Her vil mine egne indsamlede SMS beskeder blive opdelt i emner, der bedst beskriver deres indhold ved hjælp af Latent Dirichlet Allokering. Herunder ses de fem emner der kommer frem når der foretages en LDA analyse af de 577 SMS beskeder der er blevet indsamlet. Koden kan findes after *Appendix* med navnet *dansklda.py*

topic #0: 0.111*bare + 0.094*vi + 0.075*ved + 0.072*smukke + 0.058*helt + 0.058*maade + 0.050*synes + 0.048*glaeder + 0.045*ja + 0.037*mhh
topic #1: 0.101*kl + 0.094*idag + 0.063*blir + 0.061*traene + 0.059*daniel + 0.055*da + 0.047*nej + 0.045*3 + 0.040*fri + 0.037*faerdig
topic #2: 0.150*skal + 0.084*gaar + 0.068*et + 0.066*se + 0.061*lyder + 0.058*tid + 0.053*jo + 0.051*bli + 0.046*hva + 0.044*vildt
topic #3: 0.171*maaske + 0.150*nu + 0.098*have + 0.082*yep + 0.073*lidt + 0.048*noget + 0.048*vaeret + 0.047*okay + 0.029*hmm + 0.020*nok
topic #4: 0.162*skat + 0.157*dig + 0.095*til + 0.079*ikk + 0.064*elsker + 0.064*dejlig + 0.030*hvor + 0.029*kaereste + 0.029*dejligt + 0.027*langt

I det første emner ses det, at det er et emne omhandlende SMS beskeder til min kæreste, afledt af ordet *smukke*. Derudover tyder det på, at emnet beskriver en klassisk "*Jeg glæder mig til at se dig igen*"-besked, set ud fra ordet *glæder*.

Det andet emne omhandler tydeligt beskeder skrevet vedrørende træning. Dette ses for først og fremmest ud fra ordet *træne*, men for mig som analysator, ses det også på at *daniel* er navnet på min faste træningspartner. Det er altså et emne der omhandler aftale af træning, hvilket bakkes op af *idag*, *kl*, *fri*, og *færdig*. Disse ord er blevet benyttet til at beskrive, på hvilket tidspunkt der skulle trænes.

Det tredje og det fjerde emne er ikke til at bedømme, da de ikke umiddelbart indeholder nogle ord der har nogen særligt sigende betydning.

Det femte emne er tydeligt et emne der omhandlende beskeder til min kæreste, da det er beskrevet ved ord som *skat*, *kæreste*, og *elsker*. Dette emne understøttes af *dejlig* og *dejligt*, selv om de ikke direkte peger på et kæreste-emner.

Af analysen kan der afledes en relativt begrænset kontaktflade for brugeren til omgivelserne, da to af emnerne omhandler kæresten, mens det sidste omhandler træning. At der ikke fremkommer flere emner, der afspejler andre underliggende latente strukturer tyder på, at netop kæreste og træning er de dominerende emner i SMS beskederne.

Det kan ud fra analysen af LDA emnerne konkluderes, at Latent Dirichlet Allokering er i stand til bestemme nogle forholdsvis præcise emner, baseret på en relativt lille mængde dokumenter.

4.3.4 Emnefordeling af BAWE

I dette afsnit vil LDA blive anvendt til at opdele korpusset *British Academic Written English*, BAWE[6]. BAWE korpusetblev udviklet i et samarbejde mellem Universiteterne i Warwick, Reading og Oxford Brookes [**7**]. Korpusset indeholder 2761 vurderede tekster skrevet af studerende. Teksterne omhandler fire brede fagområder, *Arts and Humanities*, *Social Sciences*, *Life Sciences* og *Physical Sciences* og i alt er 35 discipliner repræsenteret. Efter *7 Appendix* ses koden hvormed det forbehandlede korpus er blevet analyseret med LDA. Til

[6] På BAWE's hjemmeside kræves det at den følgende tekst præsenteres i afhandlinger, hvor korpusset er anvendt: *The data in this study come from the British Academic Written English (BAWE) corpus, which was developed at the Universities of Warwick, Reading and Oxford Brookes under the directorship of Hilary Nesi and Sheena Gardner (formerly of the Centre for Applied Linguistics [previously called CELTE], Warwick), Paul Thompson (Department of Applied Linguistics, Reading) and Paul Wickens (Westminster Institute of Education, Oxford Brookes), with funding from the ESRC (RES-000-23-0800).*

forbehandling af BAWE og andre korpora benyttes koden der ses i *preproccess.txt* efter 7 *Appendix*

Her ses fem af de 35 emner som er blevet dannet ved LDA. Den samlede mængde emner kan ses i Appendix

```
>>> lda.printTopics(5)
topic #0: 0.039*egypt + 0.037*burial + 0.017*houses + 0.016*available + 0.014*forest +
0.013*spell + 0.013*leonard + 0.012*period + 0.011*scheme + 0.010*service
topic #1: 0.057*war + 0.038*power + 0.018*interests + 0.016*leader + 0.016*hannibal +
0.015*livy + 0.013*polybius + 0.013*pentheus + 0.013*fight + 0.012*support
topic #2: 0.131*formula + 0.046*values + 0.040*experiment + 0.032*equation +
0.029*results + 0.029*b + 0.028*using + 0.024*m + 0.023*table + 0.016*=
topic #3: 0.068*law + 0.051*nitrate + 0.033*micro + 0.019*state + 0.015*b + 0.014*legal
+ 0.014*objection + 0.013*austin + 0.011*whatever + 0.010*evans
topic #4: 0.064*age + 0.064*children + 0.056*child + 0.028*sli + 0.020*infants +
0.015*1991 + 0.015*child's + 0.014*wilson + 0.014*guilt + 0.012*skills
```

I det første emne, kan det ud fra det de første to ord, *egypt* og burial tyde på at emnet omhandler det gamle Egypten. En søgning på *houses* i teksterne sammen med *egypt og burial*, resulterede i et stykke omhandlende de forskellige metoder hvorpå egypterne i det gamle Egypten begravede deres døde, hvoraf en af dem var at bygge mausoleer der lignede rigtige huse.

For det andet emne leder *war*, *power* , *interests* og *leader* ikke rigtigt til noget specifikt emne andet en noget omhandlende krig eller krigsførelse. Når ordene *hannibal*, *livy* og *polybius* bringes ind i sammenhængen, er det tydeligt at emnet omhandler Hærføreren Hannibal og hans bedrifter under Den Anden Puniske Krig mellem Rom og Karthago. Historier om dette er nemlig blandt andet fortalt af den romerske historiker kaldet Titus "Livy" Livius og den græske historiker Polybius.

Det tredje emner omhandler ud fra resultaterne forskning generelt, da det er beskrevet ved blandt andet, *formula*, *values, experiment, equation* og *result*.

Det fjerde emne er sværere at tyde, da *law*, *nitrate* tyder på en form for biokemi ved søgning i teksterne, mens *state*, *micro*, *law* fører til forskning vedrørende Jordens energibalance samtidig med *micro* og *state* fører til økonomi.

Det fjerde emne er også meget generelt, da det ser ud til at omhandle biokemi, udledt fra *law*, *nitrate*, ,*micro*, *state*, hvor det er *nitrate* der peger mest i retningen af kemi, hvor de andre underbygger dette, mens *micro* indikerer biokemi.

I det femte emne er temaet tydeligt domineret af børn fra *child*, *children og child's*, samt *infant*. Fra SLI, *Specifik Language Impairment*, ledes mod undersøgelser af sprogforstyrrelser hos børn

Det kan konkluderes, at det er relativt nemt, at bestemme essensen af de emner der bliver givet ved LDA analyse. Det er dog besværligt, at finde ud af præcis hvad alle ordene indikere og om de alle har noget værdifuldt at tilføje til sammenhængen.

4.4 Latent Semantisk Analyse

I dette afsnit vil der blive beskrevet anvendelse og begrænsning for brug af LSA på data indsamlet i casen. Derefter vil LSA blive benyttet til at analysere nogle håndgribelige data.

4.4.1 Anvendelse på case

I relation til casen findes der flere forskellige scenarier, hvor Latent Semantisk Analyse kan benyttes til at optimere humørbestemmelsens præcision. De to mest relevante anvendelsesmetoder er:

- *Metode 1: Udvide ANEW samlingen med synonymer til de vurderede ord.*
  Ved at foretage Latent Semantisk Analyse af hvert ord i ANEW samlingen kan ord der har den samme betydning som allerede eksisterende ord i listen tilføjes til listen, med en valens og ophidselses score der er skaleret efter, hvor langt de nyfundne ord er fra det originale ord på listen. Dette vil give mulighed for at få flere hits når SMS beskeder analyseres, da der med den originale samling ikke bliver fundet særlig mange ANEW ord per SMS, grundet SMS beskedernes korte form og det begrænsede ANEW ordforråd.

- *Metode 2: Vurdere SMS beskeder for at undersøge om nogle af de brugte ord i beskeden har synonymer der findes i ANEW samlingen.*
  Ved at foretage LSA af hvert ord i en SMS, kan der findes mange synonymer eller lignende ord for hvert ord i en SMS. På denne måde kan sandsynligheden for, at ordene i en SMS eller dets synonymer findes på i ANEW samlingen forbedres betydeligt.

De to metoder har forskellige konsekvenser for selve vurderingens udførelse i praksis. Det arbejde der skal udføres i forbindelse med *Metode 1* kan alt sammen udføres uafhængigt af applikationen, hvilket vil sige, at det er noget der foretages en gang og derefter står den udvidede liste til rådighed for applikationen uden

videre analyse eller kontakt til en server. Dette gør, at applikationen hurtigt kan vurdere SMS beskederne som de afsendes . Denne fremgangsmåde kræver dog mere processorkraft jo større selve ANEW samlingen bliver, hvilket ikke er ideelt i et mobil miljø.

Metode 2 kræver derimod, at hver afsendt SMS sendes til en server, for at der kan foretages LSA af hvert ord. Når der er udført LSA af SMS beskeden skal de relevante synonymer udvælges ud fra deres lighed med de originale ord og derefter gennemsøges ANEW samlingen for sammentræf. Ved brug af *Metode 2* er det altså nødvendigt at en stor del af beregningerne foregår samtidig med applikationen er i brug, hvilket vil være mere processor- og energikrævende. Den store mængde processering vil nødvendigvis foregå på en server, da det kræver en stor meget processorkraft at foretage LSA tilstrækkelig hurtigt. Dette er dog ikke noget problem, da processorkraft og strømforbrug ikke er nogen begrænsning på serversiden. Grunden til det helst skal gøres hurtigt på serversiden er, at applikationen skal bruge udregningerne til at opdatere sindstilstand og derefter ikke kan vente flere timer når applikationen skal fungerer i realtid. Fordelen ved denne metode er, at SMS beskeden kan blive analyseret både med LSA og ANEW samlingen på serversiden. Dette gør, at der er en minimal belastning af telefonen, der dog skal afsende samtlige beskeder til serveren.

4.4.2 Begrænsning for case

Problemet med at benytte LSA til at forøge præcisionen af de stemningsværdier der beregnes ud fra mobil kontekst er, at selve metoden kræver meget computerkraft til beregning af sammenhængende og egner sig derfor ikke til et batteribegrænset mobilt miljø.

For at omgå denne begrænsning kan tekstbeskeden i stedet sendes til en dedikeret server der analyserer indholdet og resultatet tilbage til applikationen. At skulle uploade SMS beskeden kræver også en smule mere batteri end den originale ANEW sammenligning, men det burde også kunne give en mere præcis bestemmelse.

4.4.3 Synonymer med LSA

I dette afsnit vil Latent Semantisk Analyse blive benyttet til, at finde synonymer til en række ord, for at afprøve præcisionen af LSA. Synonymerne findes for at efterprøve anvendelsen af LSA på casen, altså i tilfælde af at ANEW samlingen skulle udvides med synonymer til de allerede vurderede ord eller hvis SMS beskeder skulle analyseres for, at have flere ord der kunne give match med ANEW samlingen.

For at kunne finde synonymer laves en forespørgsel, hvor det ord man ønsker at findesynonymer til, sammenlignes med termerne i det eksisterende korpus. De termer i korpusset med den største cosinuslighed til søgeordet, burde så have den største semantiske lighed med søgeordet. Altså, vil resultatets præcision afhænge af, hvilket korpus der benyttes og antallet af dimensioner der er bevaret, se eventuelt *2.4.2.4 Reduktion af rang*.

For at undersøge effekten af det valgte korpus vil der blive benyttet tre forskellige korpora til opgaven

- **BaweGut**
  Dette er et korpus jeg selv har sammensat og vil blive forklaret i detaljer senere.
- **Engelsk Wikipedia**

Samtlige sider fra det engelske Wikipedia. Yderligere information om dette korpus kan findes i 0

  Analyse af engelsk Wikipedia.
- **LSA@CU - *General reading up to 1st year college***
  Dette er en online LSA service, stillet til rådighed af University of Colorado Boulder [**15**], hvor der kan foretages forskellige former for LSA forespørgsler. Servicen har flere forskellige korpora , at vælge imellem og her benyttes et bestående af tekster for generel læsning op til første år af gymnasiet.

Ved hjælp af disse tre korpora vil der blive foretaget Latent Semantisk Analyse af en række forskellige ord og resultaterne for de tre vil blive sammenlignet. Først vil konstruktionen af korpusset OxfordGut blive gennemgået og derefter vil der blive foretaget sammenligninger

4.4.3.1 Sammensætning af korpus

Det oprettede korpus BaweGut, er en sammensætning af tekster fra Project Gutenberg og korpusset *British Academic Written English*, BAWE. Fra Gutenberg er benyttet

- En samling af tekster af
  - Charles Dickens
  - Jane Austen
  - Mark Twain
- Samt
  - *A Collection of Stories, Reviews and Essays by Willa Sibert Cather*

Teksterne er blevet behandlet, ved at fjerne stop ord, hvilket er de ord der ikke holder nogen større betydning for sammenhængen, samt tegn. Koden der er anvendt til at gøre dette er *preprocess.txt* efter *7 Appendix*

4.4.3.2 Sammenligninger med LSA

I dette afsnit vil der blive foretaget sammenligner af termer med tre de tre forskellige korpora der blev beskrevet tidligere. Der vil blive sammenlignet nogle ord med tydelig stemningsmæssige ladninger og der vil så blive diskuteret, hvilke af de tre der klarer sig bedst. I de opstillede tabeller, vil resultaterne for engelsk Wikipedia, BaweGut og LSA@CU blive vist fra venstre mod højre. Det skal nævnes, at LSA som regel altid giver en cosinuslighed på 1 for det ord som der er blevet søgt på, da de er ens og dette vises ikke i tabellerne. Ordene der laves forespørgsler er valgt på baggrund af et mål om , at analysere nogle ord der ligger spredt i valens- og ophidselsesrummet. Alle forespørgsler af fortaget med 150 dimensioner, da dette efter test af de tre korpora har vist, at give de bedste resultater. Koden for LSA forespørgslerne kan findes efter7*Appendix* som *wikiquery.py* og *bawegutquery.py*

Først startes med *happy,* hvilket betyder lykkelig eller glad. Dette ord er valgt da det er et fundamentalt ord og der kan ikke tages fejl af dets følelsesmæssige ladning.

Wikipedia LSA	***BaweGut LSA***	***LSA@CU***
afraid(0.80028)	happiness(0.567584)	sad(0.87)
wonderful(0.79598)	happiest(0.474054)	loved(0.78)
everything(0.76442)	heart(0.451928)	liked(0.74)
my(0.76345)	sorrow(0.404142)	wonderful(0.74)

Med engelsk Wikipedia som korpus, er der kun to af de fire resultater som kommer tæt på en betydningsmæssig lighed, nemlig *afraid* og *wonderful*. *Afraid* befinder sig nemlig i den modsatte ende af det følelsesmæssige spekter, og det er en af hagerne ved LSA, at det netop også giver høje ligheder for antonymer. Altså ord der har den modsatte betydning af et givent ord, omvendt af synonymer. Derudover findes *wonderful*, som i hvert fald kan give associationer til *happy*, men dog uden at være et decideret synonym.

Med BaweGut fås til gengæld nogle fine resultater, et af resultaterne er en bøjning af selve søgeordet, nemlig *happiest*, mens et andet er *happiness* , altså lykke eller glæde, hvilket må siges at ligge tæt på af *happy*. På BaweGut's liste findes også *Sorrow*, hvilket er et antonym til *happy* og grundet LSA opbygning, kan disse ikke undgås. Ordet *heart*, har ikke nogen direkte relation til *happy*, men dukker sandsynligvis op, da ordet *heart* og *happy* ofte findes i lignende kontekster.

LSA@CU korpusset giver nogle udemærkede resultater, som blandt andet er antonymet, *sad*, og de to ord*, loved* og *liked*, altså at være elsket og at være vellidt, hvilket er to ord der kan associeres med at være glad. Som med Wikipedia LSA dukker ordet *wonderful* også op her.

Det næste ord til at blive sammenlignet er *bad*, altså slem, ond eller dårlig, som ikke holder nogen speciel ophidselse, men til gengæld en lav valens uden at være et antonym til *happy*.

Wikipedia LSA	***BaweGut LSA***	***LSA@CU***
anymore(0.78755)	better(0.47550)	trouble(0.78)
gotten(0.78208)	spanked(0.45333)	just (0.75)
stupid(0.77838)	unmarketable(0.44833)	got (0.74)
damn(0.77139)	dont(0.44201)	too(0.74)

Her klarer Wikiepedia sig igen ikke særlig godt, da det eneste ord der har en association til *bad* er *stupid*, selv om det ikke er et synonym.

BaweGut klarer sig bedre, allerede med det første ord *better*, som er en bøjning af *good*, der jo er et antonym til *bad.* Udover dette har både *spanked* og *unmarketable* associationer til *bad*. I det man får smæk når man har været slem og noget ikke kan sælges når det er dårligt, hvilket begge er betydninger af *bad*.

Korpusset for LSA@CU giver kun resultatet *trouble*, hvilket associere til *bad* ved at man får problemer når man er slem, eller man er problemer for andre, når man er slem eller ond.

Det næste ord der sammenlignes er *tired*, som ikke holder lige så meget værdi medhensyn til valens som *happy*, men det er valgt da det derimod siger meget om *ophidselsen.*

Wikipedia LSA	***BaweGut LSA***	***LSA@CU***
never(0.91321)	chestnut(0.45430)	sleep(0.77)
trouble(0.91174)	hungry(0.53284)	morning(0.75)
pretend(0.90633)	puppy(0.47536)	asleep(0.74)
knows(0.89864)	gild(0.45050)	sleepy(0.70)

For ordet *tired* ses nogle meget dårlige resultater for både Wikipedia og BaweGut, da ingen af resultaterne kommer tæt på at være hverken antonymer eller synonymer til *tired*. Her viser LSA@CU modsat nogle rigtig gode resultater, med *sleep*, morning, *asleep* og *sleepy*, hvoraf *sleepy* er et rigtig godt synonym, mens *sleep* og *asleep* holder rigtig god association til det at være træt. Det sidste

morning associere også godt *tired*, da man ofte er træt eller søvning om morgenen.

Det sidste ord er *angry*, sur eller vred, da det har den modsatte ophidselse af *tired*, men samtidig en definerbar og lav valens.

Wikipedia LSA	***BaweGut LSA***	***LSA@CU***
shocked(0.92012)	manner(0.47518)	anger(0.85)
disgusted(0.91361)	resentment(0.44416)	unhappy(0.79)
embarrassed(0.91089)	contempt(0.43899)	afraid(0.76)
anger(0.91039)	hasty(0.43594)	upset(0.74)

Her klarer Wikipedia korpusset sig rigtig godt. Først gives tre øverste ord, *shocked*, *disgusted*, *embarresed*, som er knyttet til vred, da disse kan fremkalde vrede eller det at være blive sur. Dette leder til det næste ord *anger,* eller vrede, som tydeligt ligger tæt på *angry*.

For BaweGut gik det mindre godt, *resentment,* altså harme og bitterhed associere relativt godt til vrede, hvilket *contempt*, foragt, også gør. Derimod passer *hasty* og *manner* overhoved ikke på *anger*.

Til sidst klarer LSA@CU korpusset sig rigtig godt, med *anger* som det første ord, efterfulgt af *unhappy*, hvilket også er et ord med lav valens, det har dog ikke den samme høje ophidselse. Det har *afraid* og *upset* til gengæld, selv om de ikke er synonymer til *anger* kan det at være bange, samt det at være ked af det eller oprevet, godt associeres med det at være vred.

Som konklusion må det sluttes, at LSA@CU foretog den bedste Latent Semantiske Analyse, efterfulgt af mit eget sammensatte korpus BaweGut og til sidst med Wikipedia på sidsteplads.

Med hensyn til kvaliteten af resultaterne kan det ses, at LSA kan give nogle rigtig gode resultater, men samtidig kan resultaterne risikere ikke at have nogen tilknytning til forespørgselsordet overhovedet.

Af den slutning kan det konkluderes, at den 2. metode der er beskrevet under*4.4.1 Anvendelse på case*, på ingen måde er realistisk som. Dette skyldes, at metoden forudsætter, at der findes synonymer og der foretages ANEW sammenligning automatisk og uovervågning, hvilket ikke er muligt. Det er ikke muligt at foretage uovervåget udvælgelse af synonymer, da resultaterne ikke nødvendigvis er synonymer, men de kan både være antonymer eller uden relation overhoved.

Den 1. metode kan stadig lade sig gøre, såfremt, at ANEW samlingen udvides ved at foretage overvåget Latent Semantisk Analyse og manuel udvælgelse af synonymer. Derved kan man blot stille spørgsmålet, om LSA har lettet denne proces betydeligt.

Udvikles der et korpus, eller en justering af metoden der sikrer en større nøjagtighed og en metode hvorpå antonymer kan fravælges, ville den begge metoder kunne øge antallet af matches med ANEW samling betydeligt.

4.5 Analyse af engelsk Wikipedia

I dette afsnit beskrives, hvordan det engelske Wikipedia kan analyseres ved hjælp af henholdsvis Latent Dirichlet Allokation og Latent Semantisk Analyse. Først beskrives den nødvendige forudgående processering.

4.5.1 Præliminære trin

For at det kan lade sig gøre, at analysere det engelske Wikipedia hentes først et dump af Wikipedia's sider og dette dump opdeles i tre filer som beskrevet på Gensim's hjemmeside [**6**] under "*Experiments on the English Wikipedia*". De nødvendige filer er

- *wiki_en_wordids.txt*
  En kortlægning mellem id-numre og ordene fra det engelske Wikipedia. Denne kortlægning udgøre en *ordbog* for det korpus som består af siderne fra Wikipedia.
- *wiki_en_bow.mm*
  En corpus iterator, bestående af en Market Matrix-fil. Denne indeholder en optælling af, hvert af ordene fra Wikipedia repræsenteret ved deres id fra *wiki_en_wordids.txt*. Denne er ækvivalent med term-dokument matricen.
- *wiki_en_tfidf.mm*
  En term frekvens-invers dokument frekvens tranformation af *wiki_en_bow.mm*.

De tre filer fylder tilsammen omkring 20GByte og på min bærbare computer tog konverteringen cirka 13 timer[7]. Tiden vil afhænge af computerens egenskaber.

[7] Acer 5942g 1.6GHz Intel Core i7, 8GB RAM

4.5.2 Analyse med LDA

For at efterprøve *Gensim*'s ydeevne bliver der foretaget Latent Dirichlet Allokation på samtlige af det engelske Wikipedia's sider. Ved at gøre dette kan der opnås det ønskede antal emner, der bedst sammenfatter indholdet af det engelske Wikipedia, ud fra de ord der bedst beskriver emnerne.

Herunder ses koden der benyttes, til at udvinde en LDA-model med 200 emner fra det engelske Wikipedia.

```
>>> import gensim, logging
>>> logging.basicConfig(format='%(asctime)s :
%(levelname)s : %(message)s', level=logging.INFO)
>>> id2word =
gensim.corpora.wikicorpus.WikiCorpus.loadDictionary('C:/w
iki_en_wordids.txt')
>>> mm = gensim.corpora.MmCorpus('C:/wiki_en_bow.mm')
>>> print mm
MmCorpus(3216231 documents, 100000 features, 498431459
non-zero entries)
>>> lda = gensim.models.ldamodel.LdaModel(corpus=mm,
id2word=id2word, numTopics=200, update_every=1,
chunks=10000, passes=1)
```

Først importeres Gensim modulet, derefter importeres og aktiveres logging, så der udskrives informationer om de hændelser der foretages i Python skallen[8]. Derefter importeres *wiki_en_wordids.txt* som en ordbog og *wiki_en_bow.mm* importeres som corpus iterator.

Til sidst oprettes LDA-modellen ved hjælp af den oprettede corpus iterator og ordbog. Dette gøres i et forløb, ved at analysere stykker på 10.000 dokumenter af gangen og opdaterer modellen efter hvert stykke.

Når modellen er oprettet kan de 200 emner udskrives. Herunder ses de fem første emner, samt det sidste. Disse er altså nogle af de emner, som er oprettet af LDA-modellen og som beskriver indholdet af Wikipedia ved hjælp af de ord der bedst beskriver emnerne.

[8] *shell*

topic #0: 0.033*storm + 0.026*tropical + 0.020*hurricane + 0.017*damage + 0.016*earthquake + 0.016*weather + 0.009*tornado + 0.009*depression + 0.009*winds + 0.008*cyclone
topic #1: 0.038*olympics + 0.033*summer + 0.031*championships + 0.027*cricket + 0.026*olympic + 0.020*men + 0.020*games + 0.016*athletics + 0.015*rank + 0.014*women
topic #2: 0.053*singapore + 0.052*philippines + 0.035*malaysia + 0.034*indonesia + 0.032*asian + 0.025*philippine + 0.024*asia + 0.020*filipino + 0.015*manila + 0.015*indonesian
topic #3: 0.093*ron + 0.060*harry + 0.046*hunter + 0.035*fan + 0.032*fisher + 0.032*wars + 0.030*luke + 0.028*falcon + 0.026*potter + 0.026*watson
topic #4: 0.048*species + 0.032*genus + 0.014*genera + 0.010*ucucha + 0.010*evolution + 0.008*description + 0.008*described + 0.008*extinct + 0.007*fossil + 0.007*retrieved
...
topic #199: 0.034*research + 0.030*institute + 0.030*professor + 0.029*science + 0.026*college + 0.022*alumni + 0.017*faculty + 0.017*dr + 0.016*degree + 0.014*studies

Ud fra de ord der beskriver det første emne, #0, fra Wikipedia, kan det ses, at det må have noget at gøre med naturkatastrofer, da de ord indeholder oversat er: *storm, orkan, skade, jordskælv, vejr, tornado, vinde* ogde to ord *tropisk* og *lavtryk* der tilsammen udgøre tropisk lavtryk. Tropisk lavtryk er den første indikator på en cyklons opståen [**16**].

De efterfølgende to emner, #1 og #2, omhandler De Olympiske Lege og nogle øgrupper i Sydøstasien.

Emne #3 er sværere, at tyde da de ord de indeholder falder i forskellige grupper

Harry Potter	***Star Wars***	***Sherlock Holmes***	***Diverse***
Harry	Luke	Watson	Fan
Potter	Falcon		Fischer
Ron			

De ord der falder i tydelige kategorier omhandler alle film, eller eventyr universer. Hertil kunne fan være en fælles attribut, da der uundgåeligt er fans til alle de tre universer. Betydningen af *fischer* som enten kan være et efternavn eller en fisker mangler dog at blive afklaret.

Ud fra de beskrivende ord omhandler det femte emne, #4, tydeligt biologi, mens det sidste emne, #199, omhandler uddannelse.

Ud fra analysen af de givne emner fra LDA analysen, ses det at det er meget nemt at uddrive meningen med de emner der gives af LDA. Ganske vidst indeholder

emnerne ord, hvis kontribution ikke umiddelbart kan bestemmes, men den generelle konsensus af ordene i emnet er tydelig.

4.5.3 Analyse med LSA

Ved Latent Semantisk Analyse af det engelske Wikipedia, benyttes de samme filer som er beskrevet i det foregående afsnit. Dog bliver der, som det ses på koden herunder, benyttet den korpus iterator, hvorpå der er blevet benyttet tf-idf transformation.

```
>>> import logging, gensim
logging.basicConfig(format='%(asctime)s : %(levelname)s :
%(message)s', level=logging.INFO)
>>> id2word =
gensim.corpora.wikicorpus.WikiCorpus.loadDictionary('C:/wi
ki_en_wordids.txt')
mm = gensim.corpora.MmCorpus('C:/wiki_en_tfidf.mm')
>>> print mm
MmCorpus(3216231 documents, 100000 features, 498431459
non-zero entries)
>>> lsi = gensim.models.lsimodel.LsiModel(corpus=mm,
id2word=id2word, numTopics=200,chunks=10000)
```

Herunder ses nogle af de emner der blev oprettet af LSA. De først emner der er på listen er Wikipedia meta-emner der omhandler administrations- og oprydningsskabeloner, samt forskellige dataindsamlings robotter der automatisk indsamler data om byer, musik, sport og så videre. Derfor vises nogle emner længere nede fra listen.

```
topic #10(75.397): 0.501*"station" + 0.295*"railway" + 0.232*"stations" + -0.219*"party" +
0.168*"radio" + -0.162*"election" + 0.137*"bytes" + 0.130*"z" + 0.128*"fm" + 0.111*"contribs"
topic #11(73.815): -0.566*"bytes" + -0.524*"z" + -0.450*"contribs" + 0.138*"station" +
0.131*"link" + -0.109*"logo" + -0.097*"gif" + 0.095*"railway" + -0.094*"cydebot" + -0.072*"x"
topic #12(71.293): -0.434*"olympics" + -0.353*"olympic" + -0.258*"summer" + -
0.218*"championships" + -0.175*"medal" + -0.174*"athletes" + -0.156*"games" +
0.137*"football" + -0.137*"medalists" + 0.133*"party"
topic #13(68.130): 0.330*"station" + 0.320*"party" + 0.243*"election" + 0.163*"stations" +
0.160*"railway" + 0.158*"radio" + 0.136*"olympics" + -0.125*"church" + 0.124*"liberal" +
0.117*"elections"
topic #14(66.020): -0.258*"district" + -0.220*"village" + -0.192*"films" + -0.184*"film" + -
0.179*"municipality" + 0.145*"game" + -0.139*"town" + -0.135*"river" + -0.128*"church" + -
0.112*"population"
INFO:lsimodel:topic #15(63.716): -0.600*"bar" + -0.383*"text" + -0.372*"fontsize" + -0.342*"xs"
+ -0.266*"till" + -0.251*"shift" + -0.067*"delete" + -0.067*"id" + -0.060*"value" + 0.055*"keep"
```

Det første udskrevne emne, #10, falder i tre forskellige hovedgrupper ud fra de positive cosinussammenligner

Radio	***Togstation***	***Diverse***
Radio	Railway	Bytes
FM	Station	Contribs
Station	Stations	
Stations		

Som det kan ses er der to grupper, hvor der kan udvindes en fælles betydning for ordene, nemlig **Radio** og ***togstation***. Ordene i ***Diverse*** har ikke umiddelbart nogen association til hinanden eller de andre grupper. Meningen med det samlede emne #10, er ikke til at udlede. En Googlesøgning [**17**] på ordene i de to første grupper, finder en Wikipedia-side der omhandler Bath FM, en radiostation med lokaler i en gammel togstation. Det er ikke til at sige om dette er et tilfælde eller ej, men det er den eneste fælles forbindelse mellem de to hovedgrupper. Ordene under ***Diverse*** kan være "støj" fra Wikipedia, fra skabeloner eller lignende.

Emne #11 indeholder næsten ingen særligt meningsfulde ord og de ord der findes tyder på, at emne #11 bygger på en robot-skabelon af en art, da ordet *cydebot* også indgår.

Det tredje emne, #12, er langt nemmere at tyde, da samtlige ord omhandler sport og især De Olympiske Lege. Et emne der også kunne ses under *4.5.2 Analyse med LDA.*

Emne #13 indeholder kun positive cosinusligheder for *railway*, *elections*, *church*. Disse ord har ingen umiddelbar tilknytning til hinanden og derfor kan ingenting udvindes for dette emne.

Ordene fra emne #14 falder godt sammen i grupper

By	***Film***	***Diverse***
Town	Film	Church
Village	Films	River
District		
municipality		

Der er altså to hovedgrupper, hvor ordene har en fælles konceptuel betydning, ***By*** og ***Film***, mens ordene i ***Diverse*** sagtens kan falde ind under begge de to andre grupper. En *kirke* og en *flod* kan sagtens findes i sammenhæng med både byer og film.

Det femte emne, #15, virker desværre også til at være noget der er afledt af en skabelon, da ordene omhandler justering af en hjemmesides udseende eller lignende.

Ud fra de analyserede LSA emner ses det, at det vurdering af de givne emner er besværligt. Der kan være tvivl om, hvorvidt de konklusioner der drages på baggrund af ordene i hvert af emnerne, overhoved er korrekt.

Under de forberedende beregninger til indholdet af afsnittet *4.4.3.2 Sammenligninger med LSA* blev der gjort nogle opdagelser der har stor betydning for analysen af det engelske Wikipedia og dette vi blive forklaret i det efterfølgende underafsnit.

4.5.3.1 Udforskning af koncepter

I den forskende proces forud for afsnittet *4.4.3.2 Sammenligninger med LSA* blev der gjort en opdagelse under en forespørgsel på ordet *marvel* afledt af en original søgning på *marvelous*. Resultatet af søgningen på *marvel* ses herunder i tabellen i det øverste venstre hjørne

marvel(1.00000)	***religion(1.00000)***
superheroes(0.99487)	religions(0.93318)
supervillains(0.99117)	adherents(0.88740)
comics(0.99087)	beliefs(0.88287)
supervillain(0.98941)	belief(0.85813)
busiek(0.98908)	teachings(0.85668)
mutates(0.98873)	sects(0.85104)
daredevil(0.98868)	theologies(0.85092)
namor(0.98816)	universalism(0.84960)
pencilled(0.98669)	religiosity(0.84856)
mmorpg(1.00000)	***context(1.00000)***
gamers(0.95380)	sense(0.86354)
mmorpgs(0.94704)	meaningful(0.85305)
mmo(0.94670)	ambiguity(0.84736)
gaming(0.94087)	describe(0.83351)
gamespy(0.93953)	phrase(0.82366)
massively(0.93770)	meanings(0.81117)
multiplayer(0.93318)	contexts(0.80895)
rpgs(0.9271)	meaningless(0.80183)
gamer(0.92575)	unambiguously(0.80174)

Ved at lave en forespørgsel på ordet *marvel*, resulterede LSA analysen i en masse ord der beskriver hvad *Marvel™* universet indeholder [**18**]. *Marvel™* er nemlig et selskab, der laver tegneserier, med mere, omhandlende superhelte og superskurke, hvilket tydeligt fremgår af svarende fra LSA. Imellem svarene er også navne på karakterer, nemlig Namor og Dare Devil, mens en tegneren (Kurt) Busiek også er med.

Ved også at søge på ord som r*eligion*, *mmorpg*[9] og context, kan det ses, at Wikipedia generelt giver meget beskrivende svar på både specifikke og overordnede begreber.

Ud fra søgeordet og cosinuslighederne for de resulterende LSA termer, kan det altså ses at man generelt ved, at foretage forespørgsler på overordnede begreber, kan få håndgribelige og meget forklarende ord omhandlende disse.

På baggrund af de slutningerne der blev lavet i *4.4.3.2 Sammenligninger med LSA* og *4.5.3Analyse med LSA*, kan tilføjelsen af denne opdagelse tyde på, at korpora som det engelske Wikipedia, generelt fungerer bedst som en slags konceptuelle opslagsværker når det kommer til Latens Semantisk Analyse. Korpusset har nemlig ikke vist sig synderligt egnet til normal semantisk analyse eller emneinddeling.

Dette skyldes nok måden, hvorpå teksten i korpusset er opbygget, da det er skrevet med den intention at forklare et specifikt emne, hvorimod korpora indeholdende skønlitteratur som regel vil have en mindre målrettet form med meget få faglige termer.

4.5.4 Konklusion på analyse

Det ses tydeligt ud fra analysen af engelsk Wikipedia med henholdsvis Latent Dirichlet Allokation og Latent Semantisk Analyse, at emnerne fra LDA er betydeligt nemmere at forstå end dem fra LSA. Det skal også nævnes, at hensigten med LSA ikke er indeksering af tekst i emner, men nærmere forskellige sammenligning ved brug af ord, sætninger og dokumenter.

Det har desuden vist sig, at der ved brug af LSA kan findes meget præcise beskrivelser af forskellige begreber, men det kræver, at der foretages manuelle søgninger på disse og de dannes ikke selv, som ved LDA.

[9] *Massively multiplayer online role-playing game*

Kapitel 5

5 Konklusion

Der er i løbet af denne afhandling blevet forklaret om fire forskellige analysemodeller, der hver især har deres anvendelse indenfor semantisk analyse.

Der er blevet udviklet en applikation der kan indsamle forskellige former for mobil kontekstdata, med det formål at analysere disse data. De indsamlede SMS beskeder er indledningsvist blevet analyseret med ANEW samlingen og den beregnede stemning er blevet holdt op imod brugerens egne indførte stemning. Desuden er brugernes humør fordelt på døgnet og på forskellige lokationer blevet undersøgt for at finde tegn på specifikke adfærdsmønstre

Ud fra dette kan det konkluderes, at ANEW samlingen alene ikke er tilstrækkelig til at bestemme brugerens humør, men det virker realistisk, at mere præcise humørbestemmelser kan findes ved videreudvikling af metoden. I forsøget på at bestemme adfærdsmønstre er der ikke indsamlet tilstrækkelig data, til at kunne hævde hvorvidt de indsamlede data er tilfældige forekommende eller bestemt af en tendens hos brugerne.

På baggrund af anvendelse af de forskellige semantiske analysemodeller, kan det konkluderes, at de hver især udmærker sig indenfor deres felter. De probabilistiske modeller, naiv Bayes dokument klassifikation og Latent Dirichlet Allokation, egner sig især godt til at opdele tekstdata indenfor emner eller kategorier, baseret på sandsynlighedsfordelinger beregnet ud fra forekomsten af ord fordelt på dokumenter i et korpus. Modellen med den geometriske anskuelse, Latent Semantisk Analyse, viser rigtig gode resultater og modellens anvendelse viste sig egnet både til at finde synonymer, som blev godkendt ved deres ANEW score, samt til at forklare koncepter ved forespørgsler til det engelske Wikipedia. Præcisionen af LSA viste sig dog ikke at være god nok til uovervåget udvælgelse af synonymer, hvilket samtidig skyldes, at der er et problem idet antonymer også fremkommer med høj lighed.

ANEW samlingen gav ikke synderligt gode resultater, da den brugt til stemningsanalyse, ikke vurderede SMS beskederne synderligt præcist og ej heller kunne kende forskel på det positivt og negativt ladede filmanmeldelser.

Bibiografi

6 Bibliography

[1] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, First Edition ed., J. Steele, Ed. Sebastopol, CA: O'Reilly Media, Inc, 2009.

[2] M. Hart. (1971) Project Gutenberg. [Online]. http://www.gutenberg.org/

[3] (2011, Apr.) Natural Language Toolkit. [Online]. http://www.nltk.org/Home

[4] T. L. Griffiths, M. Steyvers, and J. B. Tenenbaum, "Topics in Semantic Representation," *Psychological Review*, vol. 114, no. 2, pp. 211-244, 2007.

[5] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landrauer, and R. Harshman, "Indexing by Latent Semantic Analysis," *Journal of the american society for information science*, p. 391407, 1990.

[6] R. Řehůřek. (2011, Mar.) Gensim – Python Framework for Vector Space Mo delling. [Online]. http://nlp.fi.muni.cz/projekty/gensim/index.html

[7] Y. E. Kim, et al., "Music Emotion Recognition: A State of The Art Review," in *International Society for Music Information Retrieval Conference*, Utrecht, 2010, pp. 255-266. [Online]. http: //www2.warwick.ac.uk/fac/soc/al/research/collect/bawe/

[8] last.fm. [Online]. last.fm

[9] Twitter. [Online]. http://twitter.com/

[10 A. Mislove, S. Lehmann, Y.-Y. Ahn, J.-P. Onnela, and J. N. Rosenquist. (2011, Apr.) Pulse of the Nation - U.S. Mood Throughout the Day inferred from Twitter. [Online]. http://www.ccs.neu.edu/home/amislove/twittermood/

[11 (2011, Apr.) YouTube - Pulse of the Nation. [Online].

] http://www.youtube.com/watch?v=ujcrJZRSGkg

[12 P. S. Dodds and C. M. Danforth, "Measuring the Happiness of Large-Scale Written Expression: Songs, Blogs, and Presidents," *Journal of Happiness Studies*, vol. 11, no. 4, pp. 441-456, Jul. 2007.

[13 (2011, Apr.) Natural Language Toolkit. [Online]. http://www.nltk.org/Home
]

[14 M. Dredze and K. Crammer, "Online Methods for Multi-Domain Learning
]
and Adaptation," University of Pennsylvania.

[15 U. o. Colorado. (2011, Apr.) LSA at CU Boulder. [Online]. http://lsa.colorado.edu/
]

[16 U. o. Illinois. (2010, Apr.) WW2010 University of Illinois. [Online].
]
http://ww2010.atmos.uiuc.edu/%28Gh%29/guides/mtr/hurr/stages/td.rxml

[17 Google. [Online]. www.google.com
]

[18 Marvel. [Online]. http://marvel.com/
]

Appendix

7 Appendix

7.1 Appendix I - 50 mest informative features

50 Mest informative features fra dokument klassifikator trænet på 1900 filmanmeldelser

```
contains(avoids) = True          pos : neg    =     13.0 : 1.0
contains(astounding) = True      pos : neg    =     12.3 : 1.0
contains(slip) = True            pos : neg    =     11.7 : 1.0
contains(outstanding) = True     pos : neg    =     11.5 : 1.0
contains(3000) = True            neg : pos    =     11.0 : 1.0
contains(fascination) = True     pos : neg    =     11.0 : 1.0
contains(ludicrous) = True       neg : pos    =     11.0 : 1.0
contains(insulting) = True       neg : pos    =     11.0 : 1.0
contains(sucks) = True           neg : pos    =     10.6 : 1.0
contains(thematic) = True        pos : neg    =     10.3 : 1.0
contains(hudson) = True          neg : pos    =     10.3 : 1.0
contains(hatred) = True          pos : neg    =     10.3 : 1.0
contains(seamless) = True        pos : neg    =     10.3 : 1.0
contains(conveys) = True         pos : neg    =      9.7 : 1.0
contains(addresses) = True       pos : neg    =      9.7 : 1.0
contains(incoherent) = True      neg : pos    =      9.7 : 1.0
contains(annual) = True          pos : neg    =      9.7 : 1.0
contains(dread) = True           pos : neg    =      9.7 : 1.0
contains(accessible) = True      pos : neg    =      9.7 : 1.0
contains(excruciatingly) = True neg : pos    =      9.0 : 1.0
contains(reminder) = True        pos : neg    =      9.0 : 1.0
contains(frances) = True         pos : neg    =      9.0 : 1.0
contains(stupidity) = True       neg : pos    =      9.0 : 1.0
contains(sans) = True            neg : pos    =      9.0 : 1.0
contains(illogical) = True       neg : pos    =      9.0 : 1.0
contains(winslet) = True         pos : neg    =      9.0 : 1.0
contains(fairness) = True        neg : pos    =      9.0 : 1.0
contains(deft) = True            pos : neg    =      9.0 : 1.0
contains(gump) = True            pos : neg    =      9.0 : 1.0
contains(mulan) = True           pos : neg    =      9.0 : 1.0
contains(weaknesses) = True      pos : neg    =      8.3 : 1.0
contains(unimaginative) = True   neg : pos    =      8.3 : 1.0
contains(wasting) = True         neg : pos    =      8.3 : 1.0
contains(studies) = True         pos : neg    =      8.3 : 1.0
contains(refreshingly) = True    pos : neg    =      8.3 : 1.0
contains(predator) = True        neg : pos    =      8.3 : 1.0
contains(detract) = True         pos : neg    =      8.3 : 1.0
contains(feeble) = True          neg : pos    =      8.3 : 1.0
contains(scum) = True            pos : neg    =      8.3 : 1.0
contains(seagal) = True          neg : pos    =      8.2 : 1.0
contains(finest) = True          pos : neg    =      8.1 : 1.0
```

```
contains(wisecracking) = True   neg : pos    =      7.7 : 1.0
contains(balancing) = True      pos : neg    =      7.7 : 1.0
contains(narrates) = True       pos : neg    =      7.7 : 1.0
contains(rounded) = True        pos : neg    =      7.7 : 1.0
contains(lore) = True           pos : neg    =      7.7 : 1.0
contains(symbols) = True        pos : neg    =      7.7 : 1.0
contains(mediocrity) = True     neg : pos    =      7.7 : 1.0
contains(silverstone) = True    neg : pos    =      7.7 : 1.0
contains(denial) = True         pos : neg    =      7.7 : 1.0
```

```
import os

dirs = ['C:\\movie_reviews\\neg','C:\\movie_reviews\\pos']
anew_file = open('C:\\ANEW-Engelsk.txt','r')
anew = anew_file.readlines()
score = 0
found = 0
poslist = []
neglist = []
negcount = 0
poscount = 0
for dir in dirs:
    score = 0
    found = 0
    print dir
    for file in os.listdir(dir):
        dirfile = os.path.join(dir,file)
        reader = open(dirfile)
        textlist = reader.readlines()
        score = 0
        found = 0
        count = 0
        for tl in textlist:
            split = tl.lower().split()
            for sl in split:
                for al in anew:
                    asplit = al.lower().split(",")
                    if(sl==asplit[0]):
                        # print 'found: '+sl+', score: ',asplit[1]
                        score = score + float(asplit[1])
                        found = found +1
        if(dir=='C:\\movie_reviews\\neg'):
            neglist.append(score/found)
            negcount = negcount +1
        else:
            poslist.append(score/found)
            poscount = poscount +1
outputfile = open('C:\\moodcount.txt','w')
outputfile.write("Negative folder\n")
count = 0
score = 0
for i in neglist:
    score = score + i
    outputfile.write('Score: '+str(i)+'\n')
    if(float(i)<5):
        count = count +1
average = score/negcount
outputfile.write('Correct count: %s \n' % count)
outputfile.write('Found in files: %s \n' % negcount)
outputfile.write('Average: %s \n' % average)
outputfile.write('Positive folder\n')
count = 0
score = 0
for i in poslist:
    score = score + i
    outputfile.write('Score: '+str(i)+'\n')
    if(float(i)>4.5):
```

```
        count = count +1
average = score/poscount
outputfile.write('Correct count: %s \n' % count)
outputfile.write('Found in files: %s \n' % poscount)
outputfile.write('Average: %s \n' % average)
outputfile.close()
```

```
dirs = ["C:/Users/Ironman/AppData/Roaming/nltk_data/corpora/movie_reviews/neg",
"C:/Users/Ironman/AppData/Roaming/nltk_data/corpora/movie_reviews/pos"]
anew_file = open('C:\\ANEW-Engelsk.txt','r')
anew = anew_file.readlines()
anewword = []
anewvalence = []
for i in anew:
    s = i.split(",")
    anewword.append(s[0])
    anewvalence.append(s[1])

from decimal import *
import os
def parse(dir):
    text = ""
    for line in open(dir):
        text = text + line
        split = text.lower().split()
    return split

class MyCorpus(object):
    def __iter__(self):
        for dir in dirs:
                for filename in os.listdir(dir):
                    dirfile = os.path.join(dir,filename)
                    yield dictionary.doc2bow(parse(dirfile))

import gensim
from gensim import corpora, similarities, models
text = []
corpus = []
for dir in dirs:

    for filename in os.listdir(dir):
        # print filename
        dirfile = os.path.join(dir,filename)
        text = parse(dirfile)
        # print text
        corpus.append(text)


dictionary = corpora.Dictionary(corpus)

print dictionary
corpus_memory_friendly = MyCorpus()
print corpus_memory_friendly
tfidf = models.TfidfModel(corpus_memory_friendly)
corpus_tfidf = tfidf[corpus_memory_friendly]

text = []
for doc in corpus_tfidf:
    text.append(doc)

count = 1
scorelist=[]
test = 0
for doc in text:
```

```
    score = 0
    found = 0
    for word in doc:
        j = 0

        w = dictionary[word[0]]
        s = word[1]
        if(anewword.count(w)>0):
            index = anewword.index(w)
            anewscore = anewvalence[index]
            score = score + float(s)*float(anewscore)
            found = found + 1
            test = test + s
            print "found: " + w+", score ",float(anewscore)," multiplier: ",s," result: ",
            score
    print test
    break
    scorelist.append(score)
    if(count%100==0):
        print count
    count = count +1
file = open('C:/tfidfanewscore.txt','w')
for l in scorelist:
    file.write(str(l)+"\n")


file.close()

def getLDA():
    return lda
def getmm():
    return corpus_memory_friendly
def getdict():
    return dictionary
def gettext():
    return corpus
def gettfidfcorpus():
    return corpus_tfidf
def getScorelist():
    return scorelist
```

```
dirs = ["C:/textmessages1.txt"]
stoplist = set('alle lige naar dens havde ogsaa boer kun har kunne goere dem sin faar kan
ikke alle de ikke ham eller lide gjorde det naar enten fordi siger ofte nogle sandsynligt
er dyrt vores hvad sagde om siden endnu ikke faaet hverken nogensinde tvaers hun os der dog
lod hendes af paa ca ville af kunne eller blandt egne ind mindst din han fra hende hvem der
er deres ogsaa var oensker det men ellers med end maa mig disse sige os vil samtidig kan
var min og derefter naesten er jeg det en som paa har i nogen hvis der ikke snarere kunne
tis hvordan andre som du kan efter de fleste hvorfor en af jeg saa vaere der at er det sin
unge under indtil det hver den over igen mod foer nedenfor mellem de to gjorde goere ned
under par yderligere der han  her ovre selv selv jeg sig lader mere maa mig en gang burde
vores os ud over samme hun burde saa deres egen de gennem meget var godt hvis plejer ville
ikke burde vil er du jeres selv ( ) r  . : - ? -- " ; < > .. ... / \\ *'.split())
import os
def parse(dir):
    text = ""
    for line in open(dir):
        text = text + line
        split = text.lower().split()
    return split

class MyCorpus(object):
    def __iter__(self):
        for line in open(dirs[0]):
                    yield dictionary.doc2bow(line.lower().split())

import gensim
from gensim import corpora, similarities, models
text = []

dictionary = corpora.Dictionary(line.lower().split() for line in corpustext)
stop_ids = [dictionary.token2id[stopword] for stopword in stoplist if stopword in dictionary
.token2id]
once_ids = [tokenid for tokenid, docfreq in dictionary.dfs.iteritems() if docfreq == 1]
dictionary.filterTokens(once_ids+stop_ids)
dictionary.compactify()
dictionary.filterExtremes()

corpus_memory_friendly = MyCorpus()

lda = gensim.models.ldamodel.LdaModel(corpus=corpus_memory_friendly, id2word=dictionary,
numTopics=5, update_every=1, chunks=1, passes=1)

def getLDA():
    return lda
def getmm():
    return corpus_memory_friendly
def getdict():
    return dictionary
def gettext():
    return corpus
```

```
dirs = ["C:/bawe.txt"]

stoplist = set('all just when its also had should to only has might do them his get cannot
every they not him nor like did this either where because says often some likely are dear
our what said for since yet does got neither ever across she be we who however let hers by
on about would of could or among own into least twas your he from her whom there been their
too was wants that but else with than must me these say us will while can were my and then
almost is am it an as at have in any if no rather able tis how other which you may after
most why a off i so the being theres theres there\'s thats that\'s it\'s its those under
itd itd up until thats each it\'ll itll isn\'t above again agains\'t arent before below
between both cant couldnt didnt doesnt doing dont down during few further hadnt hasnt
havent having he\'d he\'ll he\'s here heres herself himself hows id i\'ll i\'m i\'ve itself
lets more mustnt myself once ought ours   ourselves out over same shan\'t she\'d she\'ll
she\'s shouldn\'t such theirs themselves theyd theyll theyre theyve through very wasnt wed
well were weve werent whats whens wheres whos whys wont wouldnt youd youll youre youve
yours yourself yourselve ( ) r , . : - ? -- " ; < > .. ... / \\ * one two'.split())
import os
def parse(dir):
    text = ""
    for line in open(dir):
        text = text + line
        split = text.lower().split()
    return split

class MyCorpus(object):
    def __iter__(self):
        for line in open(dirs[0]):

                    yield dictionary.doc2bow(line.lower().split())

import gensim
from gensim import corpora, similarities, models
text = []
corpus_file = open(dirs[0],'r')
corpustext = corpus_file.readlines()
dictionary = corpora.Dictionary(line.lower().split() for line in corpustext)
stop_ids = [dictionary.token2id[stopword] for stopword in stoplist if stopword in dictionary
.token2id]

once_ids = [tokenid for tokenid, docfreq in dictionary.dfs.iteritems() if docfreq == 1]

dictionary.filterTokens(once_ids+stop_ids)
dictionary.compactify()
corpus_memory_friendly = MyCorpus()
lda = gensim.models.ldamodel.LdaModel(corpus=corpus_memory_friendly, id2word=dictionary,
numTopics=35, update_every=1, chunks=5, passes=1)

def getLDA():
    return lda
def getmm():
    return corpus_memory_friendly
def getdict():
    return dictionary
def gettext():
    return corpus
```

```
public static void testprocess(String input,String output){

        FileInputStream fis = null;
        BufferedReader br = null;
        Boolean found = false;
        String file = input;
        String fileout = output;
        String[] stopwords = "all just when its also had should to only has might do them
        his get cannot every they not him nor like did this either where because says often
        some likely are dear our what said for since yet does got neither ever across she
        be we who however let hers by on about would of could or among own into least twas
        your he from her whom there been their too was wants that but else with than must
        me these say us will while can were my and then almost is am it an as at have in
        any if no rather able tis how other which you may after most why a off i so the
        being theres there\'s that\'s it\'s those under it\'d up until thats each it\'ll
        isn\'t above again against aren\'t before below between both can\'t couldn\'t
        didn\'t doesn\'t doing don\'t down during few further hadn\'t hasn\'t haven\'t
        having he\'d he\'ll he\'s here here\'s herself himself how\'s i\'d i\'ll i\'m i\'ve
        itself let\'s more mustn\'t myself once ought ours   ourselves out over same
        shan\'t she\'d she\'ll she\'s shouldn\'t such theirs themselves they\'d they\'ll
        they\'re they\'ve through very wasn\'t we\'d we\'ll we\'re we\'ve weren\'t what\'s
        when\'s where\'s who\'s why\'s won\'t wouldn\'t you\'d you\'ll you\'re you\'ve
        yours yourself yourselve".split(" ");
        String[] stopsigns = "' ? \" \n € $ £ = < > * ^ % & / - . & } { _ . ` [ ] , ; : ) (
        ! / \\* <fnote> </fnote> <quote> </quote> <abstract> </abstract> <heading> <
        heading> < abstract> <figure > <table > < quote> < fnote> <picture > < list>
        </heading> <figure/> <table/> <list> </list> < enote> < url > < url > <enote>
        </enote>".split(" ");
        String outputLine = "";
        Boolean added = false;
        Date date = new Date();
        text.clear();
        lowT="";
        @SuppressWarnings("deprecation")
        String time = date.toGMTString();
        System.out.println("Start time: "+time);
        outputLog = "";
        try {
          fis = new FileInputStream(file);
          br = new BufferedReader(new InputStreamReader(fis));
          while (br.ready()) {
            String temp = br.readLine();
            text.add(temp);
          }

        } catch (FileNotFoundException e) {
          e.printStackTrace();
        } catch (IOException e) {
          e.printStackTrace();
        }
        int counter = 0;
        for(String l : text){
            counter++;
            outputLine = "";
            lowT = l.toLowerCase();

            String temp = lowT;
```

```
            for(String s : stopsigns){
                lowT = temp;
                if(s.equals("'")){
                    temp = lowT.replace(s, "");
                }
                else{
                    temp = lowT.replace(s, " ");
                }
            }
            String[] splitstring = temp.split(" ");
            for(String t : splitstring){
                for(String s : stopwords){
                    if(t.equals(s)){
                        found = true;
                    }
                }
            if(!found){
                outputLine = outputLine+t+" ";
            }
            found = false;

            }
        outputLog = outputLog + outputLine+"\n";
        if(counter > 2700){
            System.out.println("line processed: "+counter);
        }
        else if(counter % 100==0){
            System.out.println("line processed: "+counter);
        }
        }
        System.out.println("writing file");
          try{
                // Create file
                FileWriter fstream = new FileWriter(fileout);
                    out = new BufferedWriter(fstream);
                out.write(outputLog);
                out.close();

                //Close the output stream
                }catch (Exception e){//Catch exception if any
                  System.err.println("Error: " + e.getMessage());
                }

        Date date1 = new Date();
        @SuppressWarnings("deprecation")
        String time1 = date1.toGMTString();
        System.out.println("finish time: "+time1);

    }
```

```
from decimal import *
import gensim
from gensim import corpora, models, similarities
dictionary = corpora.dictionary.Dictionary.loadFromText('C:\\wiki_en_wordids.txt')

lsi = models.LsiModel.load("C:/lsi150.lsi")

#index =
similarities.MatrixSimilarity.load('C:\\Corpora\\oxfordgut\\50\\oxfordgut50index.index')
# termcorpus=gensim.matutils.Dense2Corpus(lsi.projection.u.T)
# index = gensim.similarities.MatrixSimilarity(termcorpus)
# index.save("C:/wikiindex150")
index = similarities.MatrixSimilarity.load("C:/wikiindex150")

print 'saved index'
queryWord = "wikipedia"
bowqueryVec = dictionary.doc2bow(queryWord.lower().split())
lsiqueryVec = lsi[bowqueryVec]
print 'created queryVec: '+str(lsiqueryVec)
def printsims(query):
    # get cosine similarity of the query to each one of the 12 terms
    sims = index[query]
    # print the result, converting ids (integers) to words (strings)
    fmt = ["%s(%f)" % (dictionary[idother], sim) for idother, sim in enumerate(sims)]
    return fmt

text = printsims(lsiqueryVec)
query_list = []
for lines in text:
    indexstart = lines.find('(')
    name = lines[0:indexstart]
    number = lines[indexstart+1:-2]
    query_list.append((name,Decimal(number)))

sorted_query =sorted(query_list,key=lambda score: score[1],reverse=True)
file = open('C:\\wiki'+queryWord+'query150.txt','w')
docstring = ""
for l in sorted_query:
    docstring = docstring + l[0]+"("+str(l[1])+")\n"
file.write(docstring)
file.close()
print 'done'
```

```
import gensim
from decimal import *
from gensim import corpora, models, similarities
dictionary = corpora.Dictionary.load('C:\\Corpora\\oxfordgut\\oxfordgutdictionary.dict')

lsi = models.LsiModel.load('C:\\Corpora\\oxfordgut\\150\\model.lsi')
index = similarities.MatrixSimilarity.load(
'C:\\Corpora\\oxfordgut\\150\\oxfordgut150index.index')

# termcorpus=gensim.matutils.Dense2Corpus(lsi.projection.u.T)
# index = gensim.similarities.MatrixSimilarity(termcorpus)
# index.save('C:\\Corpora\\oxfordgut\\150\\oxfordgut150index.index')

def printsims(query):
    # get cosine similarity of the query to each one of the 12 terms
    sims = index[query]
    # print the result, converting ids (integers) to words (strings)
    fmt = ["%s(%f)" % (dictionary[idother], sim) for idother, sim in enumerate(sims)]
    return fmt

text = "this is bawegut"
split = text.lower().split(" ")
print split
for t in split:
    queryWord = t
    bowqueryVec = dictionary.doc2bow(queryWord.lower().split())
    lsiqueryVec = lsi[bowqueryVec]
    print 'created queryVec: '+str(lsiqueryVec)

    text = printsims(lsiqueryVec)
    query_list = []
    for lines in text:
        indexstart = lines.find('(')
        name = lines[0:indexstart]
        number = lines[indexstart+1:-2]
        query_list.append((name,Decimal(number)))

    sorted_query =sorted(query_list,key=lambda score: score[1],reverse=True)
    file = open('C:\\Corpora\\oxfordgut\\150\\NLTKoxfordgut'+t+'150.txt','w')
    docstring = ""
    for l in sorted_query:
        docstring = docstring + l[0]+"("+str(l[1])+")\n"
    file.write(docstring)
    file.close()

print 'done'
```