NTNU
Kunnskap for en bedre verder

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgave i TDT4305 Big Data-Arkitektur

Faglig kontakt under eksamen: Kjetil Nørvåg/Heri Ramampiaro

Tlf.: 73596755/73591459

Eksamensdato: 16. mai 2017

Eksamenstid (fra-til): 09.00-13.00

Hjelpemiddelkode/Tillatte hjelpemidler: D: Ingen trykte eller håndskrevne hjelpemiddel tillatt.

Bestemt, enkel kalkulator tillatt. Annen informasjon:

Målform/språk: Bokmål

Antall sider (uten forside): 4

Antall sider vedlegg: 0

Kontrollert av:

_____

Dato Sign

Informasjon om trykking av eksamensoppgave Originalen er:

1-sidig X 2-sidig □

sort/hvit X farger □

_____ Merk!

Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

Oppgave 1 – Hadoop – 15 % (alle deler teller likt)

a) Forklar replikering i HDFS.

Replikering i HDFS går ut på at datablokkene blir lagret på mer enn en plass. Som oftest er det tre-veis replikering, som vil si at en datablokk vil bli lagret på tre forskjellige noder. Disse kan være på helt forskjellige lokasjoner. Replikering bidrar til feiltoleranse, ettersom dersom en node krasjer, så ligger fortsatt blokkene som var lagret på den noden på andre noder som kan håndtere forespørsler.

b) Hva er hensikten med NameNode, og hvilken informasjon har denne lagret?

NameNode skal virke som et oppslagsverk for metadata om filene som er lagret i systemet. NameNode kan få inn et filnavn, og returnere blant annet antall replikeringer, blokkidentifikatorene til blokkene som fila består av, samt hvilke datanoder disse blokkene ligger på. Når en kllient for eksempel vil lese en fil, kontakter den NameNode og spør om denne informasjonen. Når NameNode svarer klienten, har klienten den nødvendige informasjonen den trenger for å kontakte aktuelle datanoder direkte. All informasjon som NameNode har kontroll på ligger i hovedminnet for å sørge for mest mulig effektivitet.

c) Hva er input og output på hhv. Map og Reduce-funksjonene?

Input for Map-funksjonen er (key, value)-par, og output er List(key, value)

Input for Reduce-funksjonen er (key, Iterable(value)), og output er List(key, value)

Oppgave 2 – Spark – 20 % (alle deler teller likt)

Anta at man har en fil *countries.tsv* (tsv = tab-separert) som inneholder informasjon om land og karakteristiske termer for landene (i dette tilfellet kan dere anta en tab "\t" per linje, mellom land og term-listen):

```
France wine,art,trains,wine,trains
Canada glaciers,lakes,hockey,maple,grizzly
Norway fjords,fjords,glaciers,trolls
Japan trains,sushi,origami,sushi,fuji
Argentina wine,glaciers,football,glaciers
```

I denne oppgaven skal dere for hver av deloppgavene under vise hvordan de kan løses vha. Spark transformasjoner/aksjoner (Scala, Python eller Java).
Hint: Splitte en tekststreng x basert på "\t" (tab): val y = x.split("\t")

a) Lag en RDD med navn ”data” basert på filen, hvor hver post/objekt er en tekststreng (String)

inneholdende en linje fra filen.

countries_schema = StructType([StructField(“Country”), StringType(), False),

StructField(“CharacteristicTerms”, ArrayType(StringType())),

])

data = spark.read.format(“csv”) \

.option(“delimiter”, “\t”) \

.schema(“countries_schema”)

.load(“countries.csv”)

Fra løsningsforslag:

val data = sc.textFile("countries.tsv").

b) Lag en ny RDD ”data2” der hvert objekt i ”data” er en ”array/list of strings”, der første tekst streng er land og tekststreng nummer to inneholder termene. Eksempelresultat (Python):

```
[['France', 'wine,art,trains,wine,trains'],
 ['Canada', 'glaciers,lakes,hockey,maple,grizzly'],
 ['Norway', 'fjords,fjords,glaciers,trolls'],
 ['Japan', 'trains,sushi,origami,sushi,fuji'],
 ['Argentina', 'wine,glaciers,football,glaciers']]
```

```
val data2 = data.map(x => x.split("\t"))
```

c) Lag en ny par-RDD (pair RDD) ”data3” der nøkkel (key) er land, og verdi (value) er en “array/list of strings” av karakteristiske termer, eksempelresultat:

```
[('France', ['wine', 'art', 'trains', 'wine', 'trains']),
 ('Canada', ['glaciers', 'lakes', 'hockey', 'maple',
 'grizzly']), ('Norway', ['fjords', 'fjords', 'glaciers',
 'trolls']),
 ('Japan', ['trains', 'sushi', 'origami', 'sushi', 'fuji']),
 ('Argentina', ['wine', 'glaciers', 'football',
 'glaciers'])]
```

```
val data3 = data2.map(x => (x(0), x(1).split(",")))
```

d) Finn antall karakteristiske termer i datasettet (ikke inkludert navn på landene).

data3.flatMap(x => x._2).count()

e) Finn antall *distinkte* karakteristiske termer i datasettet.

data3.flatMap(x => x._2).distinct().count()

f) Basert på ”data3”, lag en RDD som inneholder antall distinkte termer for hvert land.
Eksempelresultat:

```
[('Argentina', 3), ('Norway', 3), ('France', 3), ('Canada', 5),
('Japan', 4)]
```

```
val data4=data3.flatMapValues(x => x).distinct().map(x => (x._1,
1)).reduceByKey((x,y)=> x+y).collect()
```

Oppgave 4 – MinHashing – 10 % (alle deler teller likt)

lx	Element	S 1	S2	S3	S4	lx2
0	p	0	0	0	1	3
1	a	0	1	1	1	2
2	g	1	1	0	0	1
3	i	1	1	1	1	0
4	k	0	1	0	0	4

a) Forklar ”shingling” og hensikten med ”shingling”.
Shingling er brukt for å dele inn et dokument i et sett basert på en gitt verdi for K. Dersom man for eksempel har et dokument og ønsker å bruke 2-shingles for ordene i dokumentet, finner man alle par av ord som kommer etter hverandre, og hver av disse parene blir en shingle. Eksempel med dokument d = “Jeg har eksamen nå”
2-shingles: {Jeg har, har eksamen, eksamen nå}

Hensikten med dette er å gjøre sammenligning enklere og mindre ressurskrevende. Man kan for eksempel definere at et dokument må ha minst en matchende shingle for å være like nok til videre analyse.

lx	Element	S 1	S2	S3	S4	lx2
0	p	0	0	0	1	3
1	a	0	1	1	1	2
2	g	1	1	0	0	1

3	i	1	1	1	1	0
4	k	0	1	0	0	4

b) Figuren ovenfor viser en forekomstmatrise for elementene p/a/g/i/k i settene S1/S2/S3/S4.
Hva er MinHash-signaturene gitt permutasjonene lx og lx2?

	s1	s2	s3	s4
h1	inf	inf	inf	inf
h2	inf	inf	inf	inf

	s1	s2	s3	s4
h1	inf	inf	inf	0
h2	inf	inf	inf	3

	s1	s2	s3	s4
h1	inf	1	1	0
h2	inf	2	2	2

	s1	s2	s3	s4
h1	2	1	1	0
h2	1	1	2	2

	s1	s2	s3	s4
h1	2	1	1	0
h2	0	0	0	0

	s1	s2	s3	s4
h1	2	1	1	0
h2	0	0	0	0

Oppgave 5 – Datastrømmer (streaming data) – 30 % (alle deler teller likt)

Anta at du er ansatt i et firma som analyserer interessetrender for det nye kameraet Sony A9 som Amazon nettopp har startet å selge. Firmaet ditt har bestemt seg for å gjøre dette basert på sosiale media data, inkl. Twitter og Facebook.

a) Gå ut fra at du starter med å bestemme deg for finne antall meldinger som nevner ”Sony A9” ved å bruke såkalte stående spørring (”Standing Query”). Forklar hva som er forskjellen(e) mellom ”standing query” og ”ad-hoc query”. Hvordan kan ”standing query” løse oppgaven?

Forskjellen mellom standing query og ad-hoc query er at ad-hoc spørres kun en gang som en vanlig spørring, mens standing query spørres kontinuerlig og til enhver tid.

Standing query kan løse oppgaven ved å spørre “hvor mange meldinger har nevnt ‘Sony A9’ så langt?”. Svaret på denne querien vil endres hver gang det kommer en ny melding som nevner kameraet.

b) Siden vi kan se sosiale media data som datastrøm har de også de andre karakteristikker og/eller utfordringer enn statiske data. Drøft hva disse er.

Disse karakteristikkene er at:

- I en datastrøm er det kontinuerlig generering av data’
- Det er snakk om veldig store mengder data
- Dataen er ubegrenset. Man vet ikke når eller om datastrømmen slutter.
- Har varierende format og struktur. kan for eksempel bestå av både tekst, bilder og videoer.
- Dataen kommer i høy hastighet.

Den største utfordringen med datastrømmer kommer av den enorme mengden og at den endrer seg svært raskt. Dette krever rask respons i sanntid som er en stor utfordring. En annen er at ettersom datastrømmer er ubegrensede mtp når de slutter, så kan man ikke lagre hele strømmen.

c) Som alternativ til ”standing query” velger du ”bit counting” for løse deler av oppgaven. Forklar hvordan kan oppgaven oversettes til ”bit counting”.

For å bruke bit counting kan man gjøre om strømmen til en bitstrøm, hvor meldinger som ikke inneholder “Sony A9” gir bitverdi 0, mens meldinger som gjør det gir bitverdi 1. Man kan da telle antall 1ere ved bruk av for eksempel DGIM-algoritmen for å finne ut hvor mange meldinger som inneholder “Sony A9”.

d) Analysen din skal se på trendene de siste 2 ukene og du ser for deg antall ”klikk” og ”kjøp” av produkter i Amazon direkte som en del av analysen. Se for deg at du skal finne ut hvor stor andel av ”klikk” og ”kjøp” er gjort på ”Sony A9”. Til dette formålet velger du nå å bruke glidende vindu-prinsippet (“sliding window”). Anta at dette vinduet har en størrelse på 500 ”klikk” og ”kjøp” tilsammen. Forklar hvordan du går fram for å beregne denne andelen.

For det aller første vinduet, dvs de 500 første verdiene i strømmen, er det bare å telle antall klikk og kjøp, og dele dette på total antall i vinduet (500).

Når det kommer en ny verdi inn i vinduet:

Dersom den nye verdien er klikk eller kjøp, må vi endre andelen med (1-j)/N, hvor N er antall i vinduet (500), og j er 1 dersom den eldste meldingen i vinduet inneholdt klikk eller kjøp, og 0 dersom den ikke gjorde det.

e) Når vi først er inne på ”klikk” drøft hvordan ”bloom filter” kan være nyttig til å finne antall klikk. Gjør de antakelsene du finner nødvendig.

Man kan ikke direkte bruke bloom filter for å finne totalt antall klikk, men man kan bruke det for å for eksempel filtrere ut klikk fra samme person. Dersom man ønsker å finne antall klikk fra distinkte brukere på en side, kan man for eksempel hashe på både URLen til siden og brukerIDen til brukeren som klikker, og på den måten sjekke om et klikk fra en bruker ikke har vært registrert før.

f) Bruk “bloom filter”-prinsippet til å fylle ut tabellen nedenfor

Strømelement	**Hash-funksjon - h_1**	**Hash-funksjon - h_2**	**Filtrere Innhold**
			00000000000
56 = 11 1000	100 = 4	110 = 6	00001010000
428 = 1 1010 1100	10010 = 18 => 7	1110 = 14 => 3	00011011000
875 = 11 0110 1011	11001 = 25 => 3	10111 = 23 => 1	01011011000

Hint: bruk *h1(x)=y1 mod 11* og *h2(x)=y2 mod 11* som hash-funksjoner, der verdiene av *y1 og y2* er hentet henholdsvis fra oddetalls-bits fra *x* og partalls-bits fra *x*.

Oppgave 6 – Anbefalingssystem og sosiale grafer (recommender systems and so cial graphs) – 20 % (alle deler teller likt)

Firmaet du er nyansatt i er spesialist på anbefalingssystemer. Din oppgave er å utvikle gode anbefalingsalgoritmer og metoder.

a) Med fokus på fordeler og ulemper sammenlikn ”collaborative filtering” mot ”content-based recommendation”.

Collaborative filtering bruker rating av et produkt gitt av brukere som har erfaring med produktet. Denne ratingen kan være både eksplisitt (rating 1-10) eller implisitt (feks ved å se på hvor lenge en bruker har sett på en youtube-video før han klikket seg bort).

Content-based recommendation baserer seg på å se på innholdet i produktet. Dersom produktet

man vil lage anbefalingssystem for er filmer, kan man da for eksempel se på regissør, skuespillere, språk, lengde osv.

Collaborative filtering har en tendens til å gi bedre resultater, altså mer riktige anbefalinger enn content-based recommendation. Derfor burde man bruke collaborative filtering der man har mulighet (særlig hvis brukerne kan rate produktene man vil anbefale). Det er likevel noen problemer med collaborative filtering. Såkalt cold start problem oppstår når man har enten et helt nytt produkt eller en helt ny bruker. Dersom det er snakk om et nytt produkt, betyr det at ingen har ratet det produktet enda, og man får derfor problemer med å utføre likhetsestimering. Dette ville ikke vært et problem dersom man brukte content-based recommendation, da man uansett ville hatt tilgang på informasjonen man trenger. Når det gjelder en ny bruker, så er problemet at man ikke vet noe om smaken til brukeren enda. En løsning på dette kan være å ha en generell og generisk anbefaling som viser produkter som de alle fleste liker, fram til man har samlet mer data fra brukeren.

SE BILDE UNDER

Collaborative Filtering	**Content-based Recommender**
Fordeler: - Fungerer på alle type produkt - Lett og intuitivt å sette opp basert bruker/produkt-matrisen	**Fordeler**: - Trenger ikke data på andre brukere - Kan gi anbefaling til brukere med unike smak - Kan anbefale nye og ikke populære produkt (ingen cold start på nye produkter) - Bakgrunnen for anbefaling kan forklares til bruker
Ulemper: - Cold-start-problemet: trenger nok brukere i systemet til å finne "match" - Sparsitet: veldig få produkt har ratings. Problemer med å finne nok bruker-ratings på produkt - First raters: Kan ikke anbefale produkt som ikke har ratings fra før - Popularitets bias: Kan ikke anbefale produkt til brukere med unike smak. Mest tendens til å anbefale populære produkt.	**Ulemper**: - Avhengig av god feature engineering, i.e., utfordrende med å hente ut features på alt - Anbefaling til nye brukere er avhengig av å kunne ha en god brukerprofil - Overspesialisering, i.e., anbefalingene har tendens til å være kun innen brukerens profil (lav grad av serendipity)

b) Anta at du vil bruke bruker-relasjoner, som feks. "friends" og "followers" fra sosiale media, til å bygge opp brukerprofiler. Slike relasjoner kan tegnes i en graf. Bruk dette som utgangspunkt til å svare på følgende spørsmål:

i. Forklar forskjellene på "*overlapping*" og "*non-overlapping communities"* i grafer. Hvordan kan vi enkelt finne ut om to grafer overlapper.

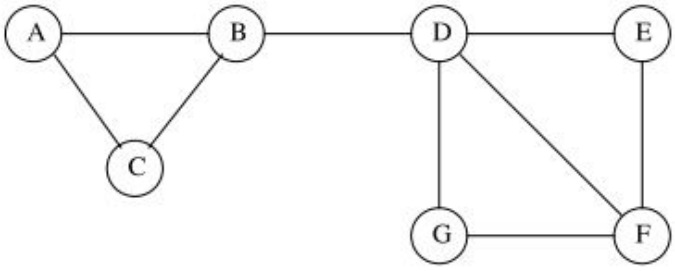
Forskjellen mellom overlappende og ikke-overlappende communities i grafer, er at i overlappende communities, så kan en node i grafen tilhøre flere communities (clusters), mens i ikke-overlappende communities, så kan en node i grafen maks tilhøre en community.

Vi kan enkelt finne ut om to grafer overlapper ved å bruke en "adjacency matrix" som er fylt med alle punktene i grafen, og se på kant-tettheten. Jo større tetthet av kanter, jo større er sannsynligheten for at grafene overlapper.

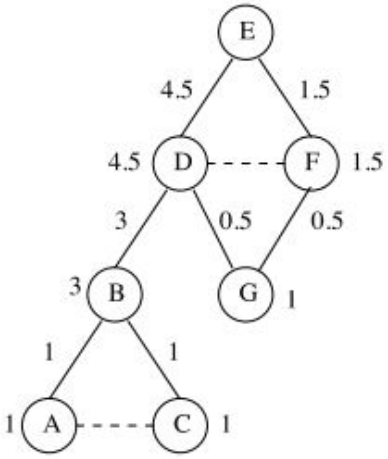
ii. Hva er hovedhensiktene med "community detection"? Forklar.

Hovedhensiktene med community detection er å finne noder som har noe til felles med hverandre. Dette kan for eksempel være personer på facebook som har mange felles venner, eller det kan være en gruppe med proteiner som interagerer med andre proteiner på en viss måte. Når man har funnet slike grupperinger, kan man finne ut hva de har til felles, og bruke dette til for eksempel videre analyse eller markedsføring.

c) Girvan-Newmans metode for beregning av "betweenness" er en vanlig metode innen graf-mining-teori. Vis hvordan du går fra Figur 1 til Figur 2 nedenfor ved å bruke Girvan-Newmans metode til å beregne "*betweenness*" i grafen i Figur 1.

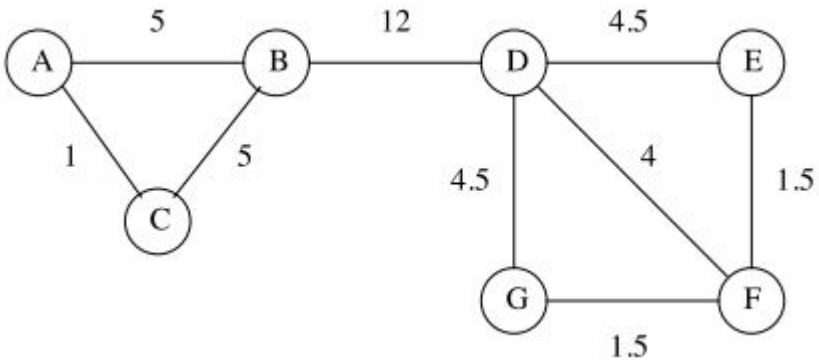


Figur 1: Start-graf



Figur 2: Steg 3 i første iterasjon.

d) Hvordan kan resultatfiguren nedenfor brukes til detektere ”communities”?



Denne figuren viser edge betweenness mellom nodene i grafen. For å detektere communities/ finne cluster, kan man i første steg fjerne den kanten med høyest edge betweenness, altså kanten mellom B og D. Alle noder fra og med b og til venstre for b må gå gjennom den kanten for å komme til D og alle noder til høyre for d. Det er derfor den kanten har høyest edge betweenness.