LF - Eksamen Big Data 2021H

Karakterskalaen NTNU: Prosentvuderingsmetode

Oppgave 1

a) De tre V-ene som nevnes innenfor Big Data sfæren er volum, velocity (hastighet) og utvalg (variety). Volum går på datastørrelsen alene. Hasighet går på i hvilken fart dataen «treffer» deg og for så vidt hvor raskt man skal kunne agere på informasjonen (f.eks sanntidsanalyse). Utvalg består av data ikke nødvendigvis kommer strukturert, men også gjerne semi-strukturert (v.eks. json) eller helt utstruktuert (videodata f.eks)

Datakildene som brukes i caset er 50-100 ulike sensorer som er montert på flymotoren og som registrer data hvert sekund. Til sammen gir det 1 TeraByte(TB) med data pr. flyvning. Dette er et stort volum, som i sær grad vokser pga. hastigheten (1 måling pr. sekund) selv om antallet datapunkt i seg selv er ganske begrenset. Slik oppgaveteksten lyder, kan det tenkes at dataen er nokså semi-strukturert (sensordata), slik at *variety* ikke er den største utfordringen.

b)

Som definert i CRISP-metodikken (Cross Industry Standars Process for Data Mining) er forretninsgsproblemet den grunnleggende årsaken til at starter opp et analyseprosjekt (del av steg 1 «Forretningsforståelse»/Business understanding). Man må aldri miste forretningsproblemet av syne siden alt arbeid man gjør må være knyttet til å løse nettopp det.

Forretningsproblemet maskinlæringsalgoritmen addresserer her, er å redusere flaskehalsen med tiden det tar å vurdere data fra alarmer, slå opp i historisk database på lignende tilfeller, og ta beslutning om det krever inspeksjon/reparasjon. En bieffekt vil være reduserte kostnader og økt kvalitten på vedlikeholdet av flymotorer som GE gjennomfører. Caset faller inn under begrepet *prediktivt vedlikehold.* I grove trekk følges denne prosessen:

1. Sensordata som input
2. Aggregerer dataen i skyen,
3. Bruker den aggregerte dataen som input i en modell
4. Modellen identifiserer anomaliteter
5. Basert på modelleresultetene, frembringes varslinger til ingeniør-teamet som skal vurdere reparasjon

Unsupervised learning (ustyrt) er å finne sammenhenger og mønstre mellom data, når man vil lære mer om data. Clustering (K-means for eksempel) brukes da gjerne.

Supervised learning (styrt) er når man har historisk data som kan trene en prediksjonsmodell på. Man kan dermed også evaluere hvor god modellen er å prognosere. Når man vet hva man vil predikere, er supervised fin. Regresjon og klassifiseringsproblem kommer inn under denne typen.

Her er det i utgangspunktet fristende å gå for outlier-deteksjon, siden det består av å identifisere sjeldene hendelser, observasjoner, og som gjerne består løses ustyrt (unsupervised). Det er ikke derimot gitt, at det er riktig.

Det kommer egentlig an på om GE sitter på en bank med historiske data allerede der de har «fasit» på om de bør gjøre inspeksjon/reparasjon eller ikke. Basert på at caset beskriver at operatøren oppgave tidligere var å søke i en «knowledge base», kan vi konkludere med det.

I så fall gir det mer mening å bruke en algoritme for klassifisering (boolsk: gjør inspeksjon/gjør ikke inspeksjon) eller klassesannsynlighetsestimering (det er mer enn x % sannsynlig at denne motoren bør inspiseres/repareres). Da blir metoden heller en form for styrt læring (supervised learning)

c)

- Aller først, løser prosjektet faktisk forretningsproblemet som gjorde at man startet prosjektet? Hvis operatørene som gjør vurderingene om vedlikehold ikke opplever algoritmen som nyttig, de ikke stoler på resultatene, eller at de ikke tar resultatene i bruk, hjelper det egentlig ikke hvor god resultatene er i teorien.

- er klassifiseringsalgoritmen testet på en annet datasett enn det er trent på (out-of-sample). Hva er i så fall resulatene på metrics som precision og recall osv, gjerne oppsummert i en confusion matrix. Hvis samme data er brukt både til trening og evaluering er risikoen stor for at modellen er overfitted, og ikke vil levere godt på ny data selv om den kan se bra ut i teorien.

- Innenfor kvalitet på resultatene kan det tenkes at falske positiv (modellen sier «reparer», når det egentlig ikke er nødvendig) er akseptabelt, mens falske negativ er alvorlig (modellen gir ikke beskjed i de kritiske tilfellene der man bør reparere».

- Er modellen god på å håndtere støy, manglende data eller feilverdier? Sensorer på flymotor er utsatt for ekstreme belastninger og med den høye målefrekvenser er det ikke til å unngå at man vil få en del søppeldata. Modellen må være i stand til å filtrere ut denne støyen (så den ikke alarmerer hele tiden) samtidig som den ikke filtrerer vekk faktiske anomaliteter.

- Hva er baseline som modellen skal benches mot? GE hadde allerede et system for å gjøre vedlikeholde av flymotorer. Man må kunne måle en reell forbedring sammenlignet med slik det var før.

- Er hele modellen black-box eller kan modelloppførselen forklares? Høy forklaringsgrad er å foretrekke siden det vil være lettere å oppdage om modellen plutselig farer galt av sted.

- Vil det være mulig å deploye modellen skarpt? Hvor ofte skal i så fall modellen kunne kjøre? Vil den være i stand til å strømme data kontinuerlig, eller må data lastes inn batchvis en gang/et par ganger i døgnet?

Som nevnt i CRISP-metodikken, er hele prosessen iterativ, men også spesielt mellom evaluering og forretningsforståelse. Vi må derfor forvente å gå frem og tilbake i flere omganger før vi kan komme med en endelig konklusjon på evalueringen.

Oppgave 2a)

Besvarelse ligger i vedlagt zippet Excel-ark

Oppgave 2 b)

Steg 0: Omdannet kategorisk data (homeowner) til numerisk data (0/1)

Steg 1: Normalisert dataen på homeowner, kredittscore, antall år med kreditthistorie og Revolving balanse. Årsaken til å normalisere er at skalaen på de ulike forklaringsparameterne er så store (år og homeowner/ikke homeowner mye mindre enn revolving balance f.eks)

Steg 2: Beregnet eukledisk distanse fra Alice og til resten av personene i treningsdatasettet for å finne de 5 nærmest naboene

Eukledisk distanse beregnes ved å ta kvadratroten av summen av den kvadrerte distansen for alle features

Steg 3: Basert på beslutningene de 5 nærmeste naboene til Alice har fått (aksept eller avslag), gi Alice det samme. Her bestemmer flertallet

De fem nærmest naboene til Alice er Ilona, Ragnhild, Even, Terje og William. Av de har 4 fått avslag, mens kun Even har fått aksept. På bakgrunn av det får Alice avslag på sin kredittsøknad

Name	Homeowner	Score	History	Balance ($)	Utilization	Decision
Quincey	1	725	20	11320	25	1
William	1	573	9	7200	70	0
Even	1	677	11	20000	55	1
Ragnhild	0	625	15	12800	65	0
Terje	0	527	12	5700	75	0
Yvonne	1	795	22	9000	12	1
Ulf	0	733	7	35200	20	1
Ilona	0	620	5	22800	62	0
Ole	1	591	17	16500	50	0
Per	1	660	24	9200	35	1
Alice	0	640	7	17300	59	?
Average	0,545454545	651,4545	13,54545	15183,64	48	0,5
St.dev	0,522232968	77,78736	6,51711	8581,384	21,56386	0,527046

NORMALISERT

Name	Homeowner	Score	History	Balance ($	Utilization	Decision
Quincey	0,87038828	0,945468	0,9904	-0,45023	-1,0666	0,948683
William	0,87038828	-1,00858	-0,69746	-0,93034	1,020226	-0,94868
Even	0,87038828	0,328401	-0,39058	0,561257	0,324617	0,948683
Ragnhild	-1,044465936	-0,34009	0,223189	-0,27777	0,788356	-0,94868
Terje	-1,044465936	-1,59993	-0,23714	-1,10514	1,252095	-0,94868
Yvonne	0,87038828	1,845357	1,297284	-0,72059	-1,66946	0,948683
Ulf	-1,044465936	1,048312	-1,00435	2,332533	-1,29847	0,948683
Ilona	-1,044465936	-0,40437	-1,31123	0,887545	0,649234	-0,94868
Ole	0,87038828	-0,77718	0,530073	0,153398	0,092748	-0,94868
Per	0,87038828	0,109857	1,604169	-0,69728	-0,60286	0,948683
Alice	-1,044465936	-0,14725	-1,00435	0,246623	0,510113	#VALUE!

KVADRATISK DISTANSE

Name	Homeowner	Score	History	Balance ($	Utilization
Quincey	3,666666667	1,194043	3,979024	0,48561	2,486022
William	3,666666667	0,741876	0,094178	1,38525	0,260215
Even	3,666666667	0,226248	0,376712	0,098995	0,034409
Ragnhild	0	0,037185	1,506849	0,274986	0,077419
Terje	0	2,110274	0,588613	1,827264	0,550538
Yvonne	3,666666667	3,970502	5,297517	0,935495	4,750538
Ulf	0	1,429381	0	4,351023	3,270968
Ilona	0	0,066106	0,094178	0,410781	0,019355
Ole	3,666666667	0,396802	2,354452	0,008691	0,174194
Per	3,666666667	0,066106	6,804366	0,890954	1,23871
Alice	0	0	0	0	0

KV.ROT AV SUMMEN AV KVADRERT DISTANSE

Quincey	3,436766634
William	2,479553609
Even	2,098340027
Ragnhild	1,377112655
Terje	2,253150889
Yvonne	4,315173023
Ulf	3,00854978
Ilona	0,768388222
Ole	2,569203271
Per	3,559045264

SORTERT FRA NÆRMEST TIL LENGST UNNA

Ilona
Ragnhild
Even
Terje
William
Ole
Ulf
Quincey
Per

Oppgave 3 PYTHON:

a) Fremgangsmåte vist i vedlagt .py-fil.

b) Fremgangsmåte vist i vedlagt .py-fil.

Klassifikasjonsrapporten skrives ut når .py-fil kjøres, og varierer litt fra kjøring til kjøring. Basert på resultatene i klassfikasjonsrapporten gir det ikke signifikant forbedring å gå over tredybde på 3 (eller for så vidt 2). Derfor begrenses max_depth til 3 og min impurity til 0.01

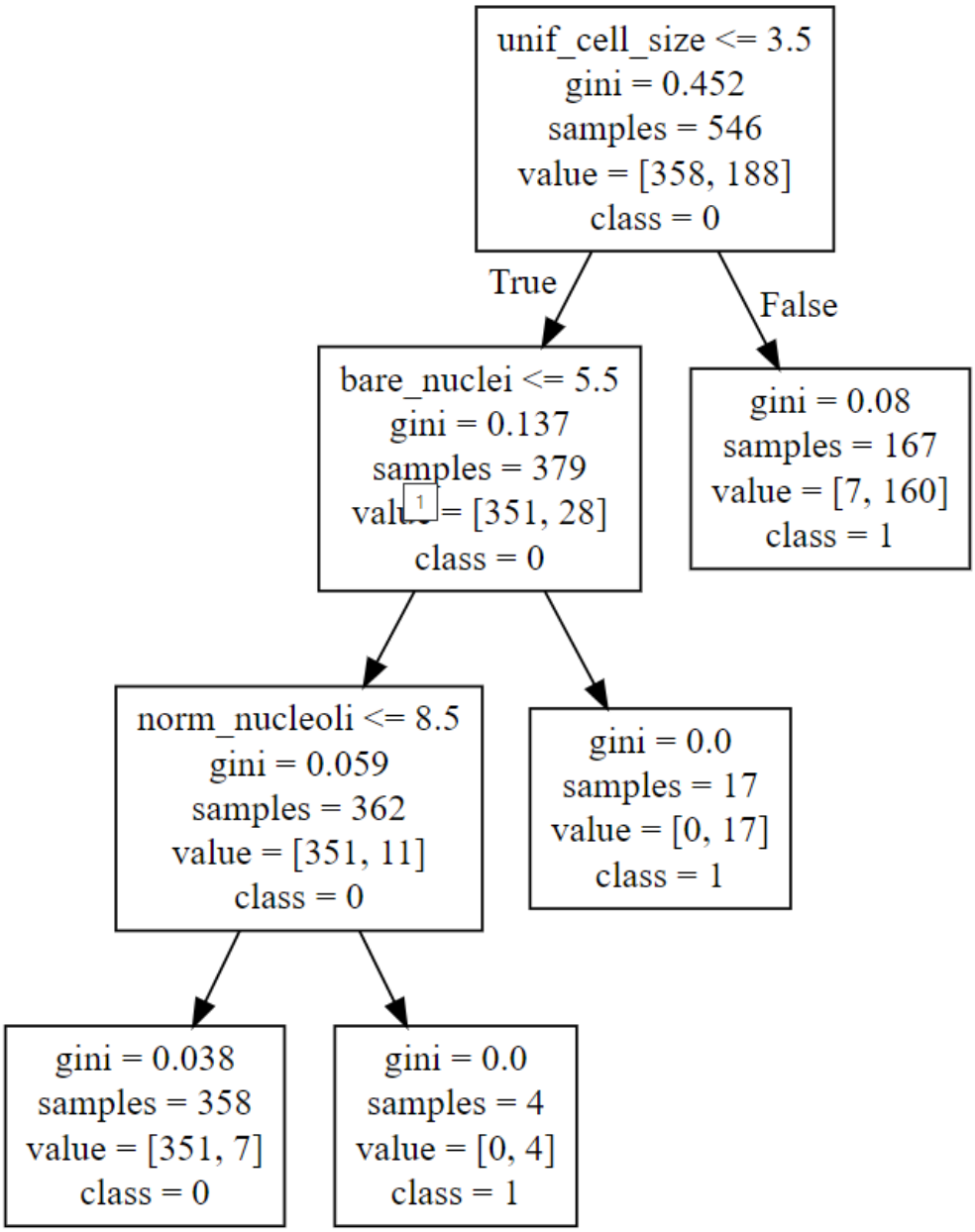
Ellers er det en god tommelfingeregel å trene på 80% og teste på 20%, derav valget på test_size=0.2

```
For kolonne classes : Counter({0: 458, 1: 241})
              precision    recall  f1-score   support

           0       0.96      0.95      0.96        82
           1       0.93      0.95      0.94        55

    accuracy                           0.95       137
   macro avg       0.95      0.95      0.95       137
weighted avg       0.95      0.95      0.95       137
```

Eksempel på treet i .dot-format:

```
digraph Tree {
node [shape=box] ;
0 [label="unif_cell_size <= 3.5\ngini = 0.452\nsamples = 546\nvalue =
[358, 188]\nclass = 0"] ;
1 [label="bare_nuclei <= 5.5\ngini = 0.137\nsamples = 379\nvalue =
[351, 28]\nclass = 0"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
2 [label="norm_nucleoli <= 8.5\ngini = 0.059\nsamples = 362\nvalue =
[351, 11]\nclass = 0"] ;
1 -> 2 ;
3 [label="gini = 0.038\nsamples = 358\nvalue = [351, 7]\nclass = 0"] ;
2 -> 3 ;
4 [label="gini = 0.0\nsamples = 4\nvalue = [0, 4]\nclass = 1"] ;
2 -> 4 ;
5 [label="gini = 0.0\nsamples = 17\nvalue = [0, 17]\nclass = 1"] ;
1 -> 5 ;
6 [label="gini = 0.08\nsamples = 167\nvalue = [7, 160]\nclass = 1"] ;
0 -> 6 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
}}
```

Grafisk representasjon av dette vises på neste side.



Oppgaven etterspør ikke noen forklaring av treet, men i utg.pkt er detganske selvforklarende siden man kan «apply»-e reglene i treet på nye data og se om klassifiseringen blir 0 eller 1.

c) Normalisering av dataene er ikke nødvendig fordi alle features har samme måleenhet.

Klassifiserings-beslutningstrær er dessuten immune mot ulike skalaer på inputvariablene, men det er ikke noe vi har gått gjennom i pensum.

Oppgave 4a (med elementer som også er relevant for 4c):

a) Utskalering, eller horisontal skalering om man vil, er metoden der man kobler mange, gjerne hundretalls standardmaskiner (gjerne kalt commodity hardware) sammen i clustre som deler på oppgaven med å lagre og prosessere data. Man går altså vekk i fra den tradisjonelle tilnærmingen

med å øke prosesskraft og lagringskapasitet (modernisere hardwaren) på én enkelt, eller et fåtall maskiner, for å løse oppgaver som setter større krav til ytelse. Dette paradigmeskiftet har skjedd de senere årene fordi veksten i datamengde har vært mange ganger større enn utviklingen i hardware. Utskalering henger i Big Data-sammenheng tett sammen med bruk av NoSQL-databaser («not only sql»), fordi bruk av NoSQL-databaser blir en naturlig konsekvens når man fragmenterer over så mange datamaskiner. I de neste avsnittene der det drøftes rundt ulempene med horisontal skalering, blir derfor en ulempe med NoSQL også til en ulempe med horisontal skalering, selv om det er nyttig å være bevisst på at det er forskjell her.

Fordelen med å utskalere er at man (i hvert fall i teorien) kan utvide i evigheten med nye og nye maskiner (koble inn flere og flere maskiner i clustre) etter hvert som behovet øker, mens man med oppskalering til slutt vil nå et tak uansett hvor mye penger man investerer.

Fordelen er også at redudans gjerne sikres implisitt ved at man gjerne lagrer kopierer av data i blokker som dupliserer på flere maskiner, men ulempen med dette er såklart at lagringskostnaden går opp.

Ulempen med utskalering er blant annet at arkitekturen blir mer kompleks, og man må utvikle løsninger for å få en god arbeidfordeling mellom alle maskinene. Hadoop er et eksempel på en teknologi som sikrer dette.

Selv om tilgjengeligheten øker (A-en i CAP-teoremet), kan man ikke nødvendigvis garantere datakonsistens til enhver tid (C-en i CAP-teoremet) (eller eventual consistency for den del, for noen systemer). Et system som krever at dataen til enhver tid er riktig vil derfor utskalering og noSQL-databaser ikke nødvendigvis være et godt valg.

En annen ulempe er at utskalering er at det ikke finnes en felles standard for NoSQL-databaser, eller spørringer mot disse. Dels skyldes det at teknologien er nyere, og dels at den er så forskjellig at det er vanskelig å lage standard. Bruk av horisontal utskalering og noSQL setter derfor større krav til en IT-organisasjon enn tradisjonelle systemer, og det er et mindre *community* å lene seg på.

Et «real-world»- eksempel der den tradisjonelle metoden med oppskalering og RDBMS (relasjonelle databasestyringssystemer) gjerne brukes er med tidskritiske bankoverføringer. Her må man til en hver tid vite hva som står på konto. Andre eksempler fra virkeligheten kan være der hverken datavolumet eller datahastighet spesielt høy, og der dataen kommer på et strukturert format. Om man allerede har en god struktur her, er det ikke noe grunn til å risikere både penger og kompleksitet.

Et eksempel fra virkeligheten der horisontal skalering er å foretrekke er opplasting og visninger av multimedia-innhold, f.eks videoer til Youtube. Her er det umulig å ikke fordele arbeidet på mange maskiner, og det gjør heller ikke noe om dataen er konstistent til enhver tid (det utgjør ingen krise om antall visninger er litt feil en liten stund).

Oppgave 4b:

Fordelen med å lagre datablokker på flere enn 3 noder er

1. Man øker redudansen, også back-up-muligheter med kræsj

2. Man øker sjansen for at en node har kapasitet til å gjennomføre operasjoner på dataen den har lagret, altså at man sikrer *datalokalitet (data locality)*. Det er dette som gjør Hadoop-clustre så effektiv, fordi noden kan aksessere dataen lokalt i stedet for å sende over nettverket. Det er for øvrig

masternoden (NameNoden) som holder oversikten og dirigerer hvilke slaver (DataNoder) som skal gjennomføre en oppgave. Flere datablokker gir mer å spille på for NameNoden.

Ulempen med å lagre datablokker på flere enn 3 noder er:

1. Økning av lagringskostnad. Vi trenger flere noder eller mer diskplass på hver node. Vi stanger derfor raskere i taket igjen, og på utskalere på nytt.

2. Mer kompleks software som ofte er dyrere å utvikle enn «tradisjonell software».

Oppgave 4c:

Eventuell konsistens betyr at det (kan) ta noe tid før databasen er konsistent. Dette står i kontrast til til RDBMS med sterk konsistens som håndterer atomiske transaksjoner og som låser tilgang til databasen fra andre prosesser så lenge transaksjonen ikke er ferdig («comitted» - og dermed kan love å alltid være oppdatert).

Fordelen med eventuell konsistens over sterk konsistens er:

- Å gjøre innsettinger i databasen er mye mer effektivt siden du ikke må vente på bekreftelse
- Man står mye friere til å lagre ustrukturert data fordi man ikke er bundet til å følge et *skjema*, men kan i stedet lagre ned i en dokumentdatabase (json), nøkkel-verdi-lager, wide-column-store, eller graf-database.
- Det skalerer bedre på store clustre (eller er strengt talt en forutsetning på store clustre, fordi man i praksis ikke kan sjekke alle noder om at alt er synkronisert til enhver tid).

Ulempen med eventuell konsistens over sterk konsistens er:

- Data vil bli usynkron mellom noder av og til. Man må derfor ha et system for å løse opp i slike konflikter. Dette øker kompleksiteten.
- Som nevnt i a), en rekke ulike teknoligier (MongoDB, Cassandre, Hadoop etc) har ulik grad av eventuell konsistens. Man må derfor ha mer kunnskap for å drifte systemer basert på eventuell konsistens, og samme grad av standarisering finnes ikke.

Eksempel på en NoSQL-db med eventuell konsistens er Cassandra