Dependensregler - Lathund

INTRODUKTION

I textprogrammet TeCST är det möjligt för en skribent att skriva, redigera och klistra in text för att få ut läsbarhetsmått och få förslag på hur texten kan skrivas om för att den ska bli mer lättläst. Förenklingsförslagen ska hjälpa skribenten att producera texter som följer de regler som identifierats för att göra en text lättläst. Dessa förenklingsförslag produceras genom att programmet applicerar skript innehållande kodade regler. Skripten består av villkorssatser som appliceras på den parsade och tokeniserade texten som den får in. Alltså när originaltexten har blivit annoterad med information som ordklasser och dependesrelationer. Skripten läser in den processade texten och förändrar den enligt vilkorssatserna. Resultatet av detta returneras som ett förenklingsförslag för användaren,

I den här dokumentationen kommer jag främst att fokusera på de olika skripten, och hur vi har gått till väga för att göra de bättre. Detta eftersom att det är det primära i hur TeCst genomför textförenklingar, samt för att det saknas dokumentation om hur detta gjorts, och implementerats.

BAKGRUND

REGLER

Vida Johansson (2017) har arbetat med skripten tidigare och har skrivit om de olika reglerna som implementerats, vilka är samma regler som Rennes (2015) har använt sig av. Totalt finns det sju regler. Nedan infogar jag en bild från Vidas rapport där hon har definierat de olika reglerna. Reglerna har bara fungerat ibland och har gjort många felaktiga förenklingar, varför det bestämdes att de skulle uppdateras och göras bättre anpassade.

Rule	Definition
Proximization (Prox.)	The rule aims to change the text to make it psychologically closer to the reader. This can be done by directly addressing the reader. This was done by changing the indefinite pronoun *man* (eng: *one*) to *du* (eng: *you*). Also, the correct form of the object corresponding to the pronoun was set, if needed.
Passive-to-active (P2A)	The rule aims to rewrite sentences of passive form to active form. The rule is triggered by a verb with the feature *SFO*, indicating a verb in passive tense.
Quotation inversion (QI)	The rule aims to change the place of a quotation and the person expressing it. The rule is triggered by sentences of quotation-like form. That is, a quotation followed by a comma, a verb, and a pronoun or a noun. A quotation either starts with a dash or has a quotation mark before and after the quote.
Straight word order (SWO)	The rule aims to rearrange the words in a sentence to achieve straight word order. That is, first a subject, then a verb, and then an object.
SPLIT-k	Sentence splits aims to divide long and/or complex sentences into new, simpler sentences. SPLIT-k performs splitting for subordinating and coordinating conjunctions. The rule was triggered by a comma followed by a word with POS-tag *SN* or *KN*.
SPLIT-r	A second split rule, which performs splitting for relative clauses. The rule was triggered by a relative pronoun (POS-tag *HP*) in a nominal phrase.
SPLIT-a	A third split rule, which performs splitting for appositions. The rule was triggered by an apposition (dependency label *AN*) within commas.

TREGEX

Reglerna är implementerade i någonting som kallas *Tregex*, och är en variant av regex. Tregex använder sig av en speciell syntax, vilken kan upplevas som svårförstådd. Därför kommer syntaxen och dess implementationen att förklaras så ingående som möjligt i den här dokumentationen.

GRUNDLÄGGANDE SYNTAX

Den data som skripten bearbetar består av Conll-x formaterade träd. All data som finns att avläsa i Conll-trädet kan tregex-skripten använda sig av. Skripten använder informationen om hur meningen är uppbyggd vid input för att undersöka om den går att skriva om på ett enklare sätt. För att detta ska vara möjligt använder sig skripten av beskrivningar av relationer mellan ord samt dess olika features som utgör de villkorssatser som avgör när olika förändringar ska genomföras.

Villkorssatserna kan liknas vid hur en **if-loop** fungerar i programmeringsspråket Python. Skillnaden ligger i hur syntaxen fungerar. I en if-loop används nyckelordet “if”, om villkoret

som följer nyckelordet uppfylls genomförs konsekvensen som står indenterad på raden under "if". I tregex representeras detta med följande grundläggande syntax:

```
{
        "villkor" and "villkor" or "villkor" and not "villkor"
        ::
        "konsekvens"; "konsekvens";
}
```

Villkoret inleds med "{", och avslutas med "}". Däremellan finns själva villkoret och konsekvenserna. I ovanstående exempel representeras villkoren av texten "villkor", och konsekvenserna av texten "konsekvens". Hur de olika villkoren och konsekvenserna skrivs syntaktiskt kommer jag att återkomma till. Fokus ligger nu enbart på den grundläggande syntaxen. Mellan de olika villkoren används olika operationsord. Dessa ord är "and", "or" och "not", dessa utgör villkorens natur, och används för att kunna kombinera flera olika villkor, samt för att utesluta andra villkor. På raden under villkoren skrivs två ":", och på raden under dem beskrivs konsekvenserna. Konsekvenser kan kombineras genom att de skrivs efter varandra med ett ";" mellan, konsekvensraden avslutas även med ";".

INGÅENDE SYNTAX - VILLKOR

Nu när den grundläggande syntaxen har avhandlats är det dags att djupdyka i hur villkoren utformas. Det skripten tar in är alltså Conll-träd, och det som reglerna kan förändra är ordföljd, meningslängd osv. Det är regler som baserar sig på relationer mellan orden i meningarna, och vilka features de olika orden har. Därför skrivs villkoren genom att beskriva specifika situationer där ett Connll-träd ser ut på ett sätt som gör att det faller inom ramarna för det som uppfyller villkoren för en regelförenkling. Ett bra exempel på detta är Quotation inversion. Denna regel undersöker ifall en mening följer formen "Det var inte bra, sa Karin", eller "-jaha, sa Karin". Om meningen ser ut på detta sätt ska skriptet göra om den till "Karin sa att det inte var bra, och "Karin sa jaha". Skriptets uppgift är alltså att lyckas med uppgiften att känna igen en mening som är uppbyggd på det sätt att regeln QI ska appliceras. I reglerna görs detta genom villkor som beskriver det utseende som ett conll-träd har om meningen har den sökta meningsuppbyggnaden.

I tregex kan man beskriva dessa förhållanden genom att beskriva de olika noderna i trädet. Varje ord i trädet är placerat på en nod. Genom att beskriva hur noderna förhåller sig till varandra, och vilka features de har kan man lyckas identifiera rätt meningar som ska förenklas. Till exempel kan man göra detta genom att beskriva vilken ordklasstagg en nod ska ha, samt att den ska ha en förälder som ligger till höger i meningen. Detta representeras på följande vis i tregex:

a postag “AB” and ←.(x)

“a” är namnet på noden, noden har ordklasstaggen “AB” och har en förälder, x, som är belägen till höger i meningen. I exemplet beskrivs relationer mellan noder som har speciella features, och det är i stort sätt så här tregex fungerar rakt igenom. När mer komplexa villkor sätts adderas fler relationer mellan noder och fler features till noderna. Namnet på noden har ingen betydelse, men sätts med fördel efter ett mönster som gör det lätt att känna igen vilken typ av nod det handlar om. Exempelvis har noder med ordklasstaggen “VB” ofta namnet v.

När flera villkor blandas är det viktigt att hålla koll på parenteser. Om man vill göra ett specifikt villkor kan det bli många olika features som söks hos många noder i trädet, vilket kan göra att det blir krångligt. Det är exempelvis möjligt att lägga in att en nod ska ha en specifik ordklasstagg, ha en förälder till höger, och dessutom ska föräldern ha en specifik ordklasstagg, och inte ha ett verb till höger om sig. Detta skulle se ut på följande vis i tregex.

a postag “AB” and <- -. (v postag “VB” and not $+ (v2 postag “VB”)

Syntaxen för alla förhållanden och villkor syns i tabellen nedan.

Condition	Definition
.<- -	Has left child
- ->.	Has right child
<- -.	Has right head
.- ->	Has left head
.<-	Has adjacent left child
->.	Has adjacent right child
<-.	Has adjacent right head
.->	Has adjacent left head
>	Has child
> >	Has predecessor
<	Has head
< <	Has successor
$- -	Has left neighbor
$++	Has right neighbor
$-	Has adjacent left neighbor
$+	Has adjacent right neighbor

INGÅENDE SYNTAX - KONSEKVENSER

Om, och bara om, alla villkor är uppfyllda i villkorssatsen kommer konsekvensen att appliceras. Som tidigare demonstrerat går det att kombinera flera olika konsekvenser, men först vill jag introducera de olika handlingarna som kan utfärdas i konsekvensen. De olika

handlingarna visas i tabellen nedan som är tagen från Vidas rapport (Johansson, 2017).

Action	Syntax	Definition
copy	copy (group | node) x before (group | node) y	Copies x and appends the copy of x before y.
move	move (group | node) x after (group | node) y	Moves x and places it after y.
delete	delete (group | node) x	Removes node x from the tree.
set	set postag x "NN"	Changes the POS-tag of x to *NN*.
set_head	set_head x (heads | headed_by) y	Sets node x as head of y or vice versa
try_set_head.	try_set_head x (heads | headed_by) y	Sets the head of a node, but does not fail if the tree becomes cyclic or disconnected, unlike set_head.
group	group x y	Creates a virtual arc between x to y, which is then considered in copy, move and delete actions.

När det gäller konsekvenserna är det viktigt att veta att de olika orden INTE är knutna till de olika noderna. När man i villkoret beskriver de features och relationer som tillhör en specifik nod namnger man noden, men om man flyttar ordet som tillhör noden följer inte noden med. Noderna är alltså fasta index. Om man glömmer bort detta kan man snabbt bli förvirrad, och inte förstå varför man får de konsekvenser som man får.

Utöver de funktioner som finns listade i tabellen ovan finns även några till funktioner som kan användas i konsekvenserna. Dessa är:

Add	add form “hej” after node a	Adderar ordet hej
Split	split before group v and after node k.	Raderar den del som beskrivs inom split
Conj	conj v pres	Böjer verbet v till presens

Add-funktionen adderar ett ord till trädet. I exemplet ovan står det att ordet adderas efter en specifik nod, dock insåg vi att detta inte är fallet i praktiken. Ord som adderas med “add” hamnar ALLTID, enligt våra observationer, sist i meningen och behöver därför flyttas runt. Conj fungerar precis som det står att det gör. Verbet böjs efter de former som angivits. Denna funktion fungerar bara på verb än så länge. Vi har börjat kolla på en implementation för att kunna böja substantiv i systemet, detta behövs för att förbättra reglerna, främst P2A men detta kommer sannolikt inte hinnas med. Split funktionen delar upp en mening och raderar de ord som finns inom de noder som sätts.

TESTMILJÖ

För att kunna uppdatera och utvärdera reglerna byggdes en testmiljö. Testmiljön körs från terminalen, och där skickar man även in vilken mening man vill förenkla. När meningen har parsats och tokeniserats appliceras alla regler på meningen. När meningen returneras kan alltså flera skript ha förenklat meningen. Om den förenkling som returneras godkänns av övervakaren uppdateras reglerna och meningen läggs till i en guldstandard. På så vis skapas en guldstandard samtidigt som reglerna uppdateras. Eftersom att en guldstandard byggs upp under tiden som reglerna uppdateras, körs de nya reglerna alltid på de meningar som redan lagts in i guldstandarden. Det gör att vi kan upptäcka när förändringar i reglerna gör att meningar förenklas på fel sätt.

Om en förändring i en regel orsakar att en gammal mening blir felaktigt förenklad ska programmet ge ett felmeddelande för att göra övervakaren uppmärksam på att den nya regeln orsakar problem, och för att det ska vara enklare att gör om regeln igen så att den fungerar felfritt. Felmeddelandet består av den korrekt förenklade guldstandarden och den felaktigt förenklade meningen efter varandra så att de lätt ska kunna jämföras och felsökas. I testmiljön appliceras alla regler på en mening och i guldstandarden kontrolleras meningarna mot alla regler, detta för att kontrollera så att meningarna inte krockar med varandra. I nuläget appliceras endast en regel på en mening i TeCST-tjänsten, men i testmiljön ska man kunna utvärdera regler så att flera regler ska kunna appliceras samtidigt på en mening.

FÖRÄNDRINGAR

De förändringar som har gjorts i regelskripten är ganska stora, och när det gäller några av skripten är de nästan helt utbytta mot nya regler. Skripten har uppdaterats i och med att vi har testat olika meningar, observerat vilka fel som uppstår, följt av en förändring i skriptet. Detta har gjort att många av skripten har blivit mer specifika än de ursprungligen var.

TIPS OCH TRICKS

Det som vi har tyckt fungerat bra är att jobba med ett skript i taget, samt att utvärdera regeln genom att skriva in olika varianter av samma typ av mening för att säkerställa att regeln verkligen fungerar som den var tänkt att göra.

FRAMTIDA FÖRSLAG

Det verkar som att det är svårt att genom handskrivna regler inludera alla olika varianter och möjliga versioner av meningar som reglerna ska appliceras på. Därför föreslår vi att man genom övervakad maskininlärning lär ett system att skriva sina egna förenklingsregler. Genom att göra det skulle man kunna undvika problem som uppstår när människor författar

regler. Människor kan inte komma ihåg alla regler, och är väldigt långsam på att söka igenom den stor mängden information som finns i skripten, vilket gör att reglerna mest troligt inte blir så effektiva och välanpassade som de skulle kunna vara om maskininlärning kopplades in och kompletterade den mänskliga kompetensen.

Ett annat förslag på förbättring inom textförenklingsområdet är att lämna iden om att förenkla text mening för mening. Många av de professionella textförenklarna som vi varit i kontakt me under projektets gång har påpekat att människor inte förenklar texter mening för mening och att det därför inte är möjligt att skriva regler som skulle kunna förenkla meningar på ett bra sätt om det är baserat på mening för menings förenkling. Detta är någonting som skulle kunna bidra till att föra området framåt. I kombination med maskininlärningen kanske detta skulle kunna skapa en mer människolik textförenkling.

Vi föreslår att man implementerar funktionen att kunna böja substantiv.

Group-funktionen verkar inte fungera som den har beskrivits i Johansson, 2017.