Beslutsträdsinduktion för probabilistisk taggning

Johan Enell
johan@enell.nu

Fredrik Larsson
dat98fla@ludat.lth.se

Sammanfattning

I detta projekt har vi granskat och implementerat en taggnings metod som är en utveckling av probabilistisk taggning. Skillnaden från den probabilistiska taggnigen är att transitions-sannolikheten uppskattas med hjälp av ett beslutsträd. Fördelen med detta är att man kan få en hög precision från en liten mängd exempel data. Ett problem som vi upptäckte var att skapandet av trädet tog väldigt lång tid. Av den anledningen utvecklade vi även ett annat sätt att skapa trädet på som avsevärt förbättrade den tidsåtgången. Enligt artikeln som vi fick algoritmen från var precisionen högre än det konventionella metoderna. Vi har inte kunnat testa om detta stämmer och vi har inte heller kunnat testa om våran metod fungerar eftersom det inte funnits tid att implementera de delarna av programmet.

1 Inledning

Att bestämma satsdelar på ord är inte helt trivialt eftersom dess betydelse kan vara ambivalent beroende på vilket sammanhang det befinner sig i. För att klara av detta brukar man titta på det sammanhang som ordet befinner sig i för att lista ut ordets rätta satsdel.

Det finns många olika metoder för att lösa detta problem på olika sätt. Vissa använder något form av regelsystem andra använder probabilistiska metoder. Även neurala-nätverk används för att lösa problemet. Om man tittar närmare på de probabilistiska metoderna så ser man att alla använder någon form av markovmodell. Eftersom det blir en stor mängd parametrar när man ska bestämma satsdelar (speciellt när man använder trigrams) blir det svårt för dessa metoder att det svårt för dessa metoder att uppskatta små sannolikheter.

I den algoritm som vi har jobbar med används ett beslutsträd för att uppskatta transitonssannolikheten.

2 Probabilistisk taggning

TreeTagger algoritmen grundar sig på den välkända och vanligt förekommande ngram taggeralgoritmen. Båda två modellerar sannolikheten av en taggad sekvens av ord rekursivt med formeln:

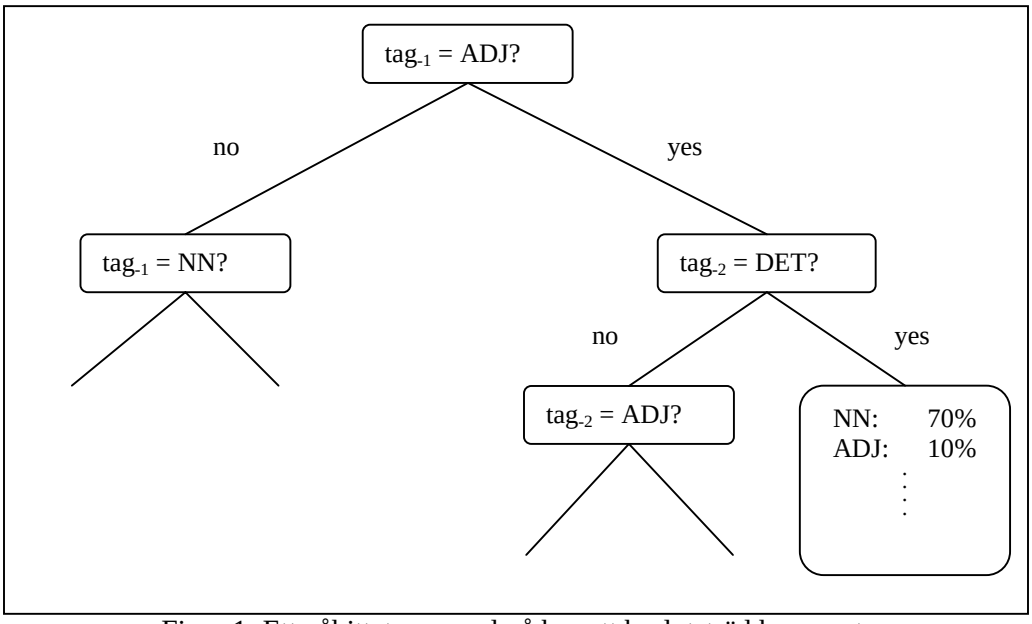
$$\begin{aligned} &p(w_1 w_2 \ldots w_n, t_1 t_2 \ldots t_n) := \\ &p(t_n | t_{n-2}, t_{n-1}) p(w_n | t_n) \\ &p(w_1 w_2 \ldots w_{n-1}, t_1 t_2 \ldots t_{n-1}) \end{aligned} \tag{1}$$

Denna metod skiljer sig på hur transitonssannolikheten, $p(t_n | t_{n-2}, t_{n-1})$, uppskattas. För det mesta använder ngram-taggers en formel som baserar sig på ”maximum likelihood estimation (MLE)” principen:

$$p(t_n | t_{n-2}, t_{n-1}) = \frac{F(t_{n-2}, t_{n-1}, t_n)}{F(t_{n-2}, t_{n-1})} \tag{2}$$

$F(t_{n-2}, t_{n-1}, t_n)$ är antalet förekomster av trigrammet t_{n-2}, t_{n-1}, t_n i korpuset och $F(t_{n-2}, t_{n-1})$ är antalet förekomster av bigrammet t_{n-2}, t_{n-1}.

Ett stort problem med denna uppskattning är att många frekvenser är så små så den sannolikhet man får fram inte är tillräckligt trovärdig. Det blir speciellt svårt i de fall då frekvensen är noll och man kan inte avgöra om motsvarande trigram är syntaktiskt korrekt eller om det bara är väldigt ovanligt. En annan viktig punkt är att en robust tagger ska kunna klara indata som inte är helt grammatiskt korrekt. Annars kan det inkorrekta



Figur 1: Ett påhittat exempel på hur ett beslutsträd kan se ut.

leda till att hela uttryck blir tilldelade noll i sannolikhet, oberoende av sekvensen av taggar. Detta skall undvikas.

Därför modifieras formeln ovan så att nollsannolikheterna ersätts med ett litet tal och seden normaliserar man om sannolikheterna så att summan blir 1. Valet av det tal som skall användas är väsentligt för kvalitén på resultatet.

3 TreeTagger

TreeTagger använder sig, till skillnad från ngram tagger, av ett binärt beslutsträd för att uppskatta transitions-sannolikheter. Figur 1 visar ett exempel på hur ett sådant träd ser ut. Sannolikheten för ett givet trigram fås genom att följa dess väg genom trädet tills man kommer till ett löv. Om vi t.ex. tittar på sannolikheten för att ett substantiv kommer efter en determinant som i sin tur kommer efter ett adjektiv p(NN| DET, ADJ) måste vi först svara på frågan i roten; Är taggen på position -1 ett adjektiv? I detta fall är svaret ja, så då går vi vidare till nästa nod via ja-vägen. Där svarar vi på frågan om taggen på plats -2 är en determinant. Även detta är sant så då kommer vi ner i ett löv. Där kan vi se att sannolikheten för detta trigram är 70%.

3.1 Skapa ett beslutsträd

Beslutsträdet som byggs från en träningsmängd av trigram använder en modifierad version av ID3-algoritmen. I varje rekursionsteg skapas ett test som delar mängden av trigram två delmängder. Delmängderna väljs så att skillnaden i sannolikhetsdistributionen för den tredje taggen blir så stor som möjligt. Testet undersöker en av de två framförliggande taggarna och kontrollerar om det är identiskt med en tag t i en mängd av taggar. Testet ser ut på följande sätt:

$$\begin{aligned} & tag_{-i} = t \\ & i \in \{1,2\} \\ & t \in T \end{aligned} \tag{3}$$

där T är en mängd av taggar.

Vid varje rekursionsteg jämförs alla möjliga test och det test som ger mest information tilldelas till nuvarande nod. Sedan utvidgas träder rekursivt på de två delmängderna som skapats utav testet. De träd som skapas läggs till som ja- och nej- träd på den nuvarande noden.

Det värde som används för att jämföra testerna, q, är den mängd information som fås av den tredje taggen när testet är utfört. Att maximera informationen är ekvivalent med att minimera genomsnittet av den information, I_q, som krävs för att den

tredje taggen går att identifiera efter att resultatet av test q är känt.

$$I_q = -p(C_+|C)\sum_{t\in T} p(t|C_+)\log_2 p(t|C_+) - p(C_-|C)\sum_{t\in T} p(t|C_-)\log_2 p(t|C_-) \quad (4)$$

Där C är de trigram som finns i exempelmängde för den aktuella noden. C_+ är de trigram där test q lyckas och C_+ är de trigram då testet misslyckas. $p(C_+|C)$ är sannolikheten att testet q lyckas, motsvarande för $p(C_-|C)$. Slutligen är $p(t|C_+)$ sannolikheten för den tredje taggen om test q lyckas och $p(t|C_-)$ om det misslyckas. Dessa sannolikheter uppskattas med MLE:

$$p(C_+|C) = \frac{f(C_+)}{f(C)} \quad (5)$$

$$p(C_-|C) = \frac{f(C_-)}{f(C)} \quad (6)$$

$$p(t|C_+) = \frac{f(t, C_+)}{f(C_+)} \quad (7)$$

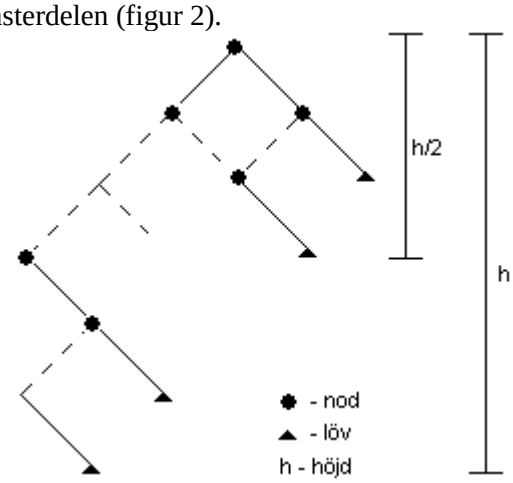
$$p(t|C_-) = \frac{f(t, C_-)}{f(C_-)} \quad (8)$$

Där $f(C)$ är antalet trigram i träningsmängden för den aktuella noden. $f(C_+)$ är antalet trigram där testet lyckas, motsvarande för $f(C_-)$. $f(t, C_+)$ är antalet trigram som klarade testet och där tredje taggen är t. Den rekursiva expansionen av trädet stannar om nästa test genererar en delmängd av trigram som är mindre än ett förutbestämt tröskelvärde, n ($f(C_+) < n$ eller $f(C_-) < n$). Sedan uppskattas sannolikheten $p(t|C)$ för den tredje taggen från de trigram som kommit till denna nod och värdet sparas i lövet.

$$p(t|C) = \frac{f(t, C)}{f(C)} \quad (9)$$

4 FastTree

När man tänker efter hur trädets form kommer att kunna tänkas vara inser man att det i princip alltid kommer att se likadant ut, och att man inte kan göra så mycket åt det. Med vänsterbarn som nej (ja), högerbarn som ja (nej) och att två ja ger löv resulterar alltid i ett träd som kommer luta starkt åt vänster (höger), med ett större antal noder. Vid ett fullt utvecklat träd med alla trigram-kombinationer kommer det nedersta lövet av högerdelen av trädet alltid ligga på halva djupet av det nedersta lövet i vänsterdelen (figur 2).



Figur 2: En enkel skiss av hur trädet kommer att se ut.

Tvärt emot vad namnet antyder är inte FastTree snabbare än något annat på att användas. Namnet försöker istället säga att den skapar träd betydligt snabbare än ID3.

Tanken med FastTree är enkel och metoden likaså. Eftersom trädet alltid, i princip, kommer ha samma form saknar balansering vikt vid skapandet av trädet och minskar därmed beräkningarna.

FastTree bygger istället helt på förekomster av unigram och bigram. Det vanligaste unigrammet väljs som rot i trädet, det näst vanligaste som vänsterbarn och så vidare. Högerbarnet är den vanligaste taggen med föräldern som föregående tagg, d.v.s. det vanligaste bigrammet med föräldern som första tagg i bigrammet. Dess vänsterbarn är den näst vanligaste taggen med den förra föräldern som föregående tagg. Ett andra högersteg resulterar i ett löv då vi har tre taggar som utgör ett trigram; vilket är vad vi söker.

På detta sätt bygger FastTree ett snabbt, korrekt, men förmodligen inte ett optimalt träd.

5 ID3 vs. FastTree

Problemet med ID3-baserade träd är uppenbar. För stora mängder träningsdata kan skapandet av trädet ta olämpligt lång tid. Algoritmen vi har implementerat har en logaritmisk tidskonsumtion med avseende på storleken av korpusen. Det hjälper dock föga då varje steg tar allt för lång tid och att ökande av antal olika tagg-typer ger en exponentiell tidsökning.

Nackdelen med FastTree är att det förmodligen skapar ett sämre träd. När algoritmen väljer nod vid yttersta nivån, dvs. Utan några ja-svar, tar trädet inte hänsyn till hur underträden kommer att se ut. Det är då möjligt att trädet som genereras av FastTree-algoritmen är genomsnittligt sämre än ID3-trädet vid sökning. Men eftersom det fortfarande är korrekt och att det är den totala tidsåtgången som spelar roll kan FastTree vara ett alternativ då man inte har ett färdigt träd att tillgå.