Implementering av ordprediktor

DD1418 Språkteknologi och introduktion till maskininlärning

Projekt av Nathalie Haghshenas och Anton Johansson

Introduktion

Denna projektuppgift gick ut på att implementera en ordprediktor i python. Ordprediktorer går bland annat att hitta i mobiltelefoner med syfte att hjälpa användare spara tid och knapptryck med förslagen de ger. Användaren kan vid inmatning enkelt välja något av de föreslagna orden och slipper därmed skriva hela ordet. Dessutom kan ordprediktorer hjälpa användare korrigera stavfel genom att ge förslag på rättstavade ord och kan även lära sig nya ord som användaren nyttjar.

Bakgrund

En ordprediktor kan implementeras genom en statistisk språkmodell. Statistiska språkmodeller är grundläggande inom språkteknologi och maskininlärning och används för att förutsäga sannolikheten för ordsekvenser i ett språk. Dessa modeller baseras på sannolikhetsfördelningar av ord i ett givet sammanhang och kan appliceras i exempelvis ordprediktorer, maskinöversättning och taligenkänning. Den typ av statistisk språkmodell som används är en n-gram-modell, där 'n' representerar antalet ord i sekvensen som beaktas för att förutsäga nästa ord. I vårt fall är N=3, det vill säga att en trigram-modell används. n-gram antar att sannolikheten för ett ord endast beror på de föregående (N-1) orden, vilket är ett Markov-antagande. Denna förenkling gör det möjligt att hantera språkets komplexitet genom att begränsa kontexten som tas i beaktande. Det går därigenom att formulera språket matematiskt som en betingad sannolikhet enligt följande:

$P(w_i \mid w_{i-n}, \ldots, w_{i-1})$ där $w_{i-n}, \ldots, w_{i-1}$ är de föregående orden till w_i

Till exempel får då trigram-sannolikheten för att ordet “dogs” förekommer efter orden “I like” värdet $P(\text{"}dogs\text{"} \mid \text{"}I\ like\text{"})$.

Själva sannolikhetsfördelningarna beräknas med hjälp av Maximum-Likelihood Estimering, MLE. MLE är en statistisk metod som beräknar förekomsterna av ord i ett corpus (genom funktionen C), och utifrån dessa beräknas den relativa frekvensen av de olika n-grammens förekomst. Matematiskt blir detta:

$$P(w_i \mid w_{i-n}, \ldots, w_{i-1}) = \frac{C(w_1, \ldots, w_i)}{C(w_1, \ldots, w_{i-1})}$$

För att beräkna trigram-sannolikheten för “I like dogs” beräknas alltså

$$P(\text{"}I\ like\ dogs\text{"}) = \frac{C(\text{"}I\ like\ dogs\text{"})}{C(\text{"}I\ like\text{"})}$$

För prediktion av ordet som skrivs samt för att hantera felstavning används även Levenshtein-avstånd. Levenshtein-avstånd är ett sätt att mäta textavståndet mellan 2 strängar och definieras som det minsta antalet textredigereringar (insättning, borttagning, ersättning) som krävs för att förändra en sträng till en annan. Genom att jämföra en felstavad sträng med modellens uppsättning korrekta ord kan Levenshtein-avståndet användas för att identifiera och föreslå den mest sannolika korrigeringen, tillsammans med ordens n-gram-sannolikheter. I vår implementation användes Python-bibliotekets implementation av beräkning av Levenshtein-avstånd.

Hypotes

Modellen förväntas kunna generera sammanhängande meningar och föreslå ord som, i och med den utökade kontexten, är grammatiskt korrekta på ett effektivare sätt jämfört med modeller som bygger på enbart bigram. Vidare förväntas modellen bli mer träffsäker och därmed öka kvaliteten hos prediktorn med större storlekar på corpus. Samtidigt förväntas corpus, i egenskap av vår datas verklighetstrogna natur leda till att förslagen prediktorn genererar är väl anpassade för prediktorer i telefoner och andra små enheter, men dess autenticitet tros också leda till att vissa icke-korrekta slangord och uttryck blir del av språkmodellen.

Implementation

Implementationen bestod av 3 program, “TrigramTrainer.py”, “Predictor.py”, samt “TrigramTester.py”.

“TrigramTrainer” är utökad från “BigramTrainer” från kursens andra laboration. Den fyller samma funktion, men nu även med trigram-sannolikheter. De beräknas på samma sätt som bigram-sannolikheterna och skrivs till en fil som utgör vår språkmodell.

“Predictor.py” är programmet i vilken själva prediktorn finns, och innehåller dessutom en GUI för användaren att nyttja. Detta i syfte att efterlikna de ordprediktorer som finns i telefoner och därmed göra användningen av detta program intuitiv. GUI:n visar en textlåda som användaren anger input i. När användaren gör detta visas 3 knappar under textlådan. Inuti dem står ordförslagen som prediktorn har genererat. Dessa förslag uppdateras vid varje keyboard-input från användaren. Vid tryck på någon av knapparna lägger programmet till ordet som stod i knappen till textfältet.

GUI:n hanteras i en klass döpt “GuiWindow”. Själva prediktionen hanteras i en klass vid namn “Predictor”. Vid användning läser programmet först in språkmodellen i dictionaries som används för att effektivt kunna nå data om ord och deras respektive sannolikheter. Sedan, vid användarinteraktion med GUI:n, anropas diverse metoder som använder språkmodellens data samt användarens tidigare input för att generera förslag på ord som användaren försöker uttrycka. Dessa anrop görs av funktioner i GUI-klassen som i sin tur sedan uppdaterar knapparna för att innehålla det som prediktorn returnerar.

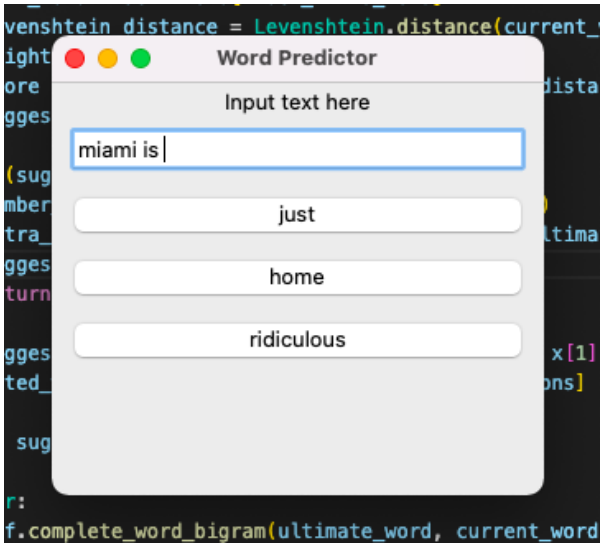


Bild 1. Prediktorn i körning, med GUI.

Intressanta aspekter hos prediktorn är bland annat att *Back-off-teknik* används. Det innebär att prediktorn, när den stöter på n-gram som inte skådats i träningscorpus och det därmed inte finns statistik på, kommer försöka hitta statistik för ordföljden med hjälp av en mindre grads n-gram. För vår prediktor innebär det att om en ordföljd på de två senaste orden inte har några förekomster i träningscorpus (och således att ingen statistik för den följden och möjliga efterföljande ord går att hitta) kommer prediktorn att försöka hitta statistik för en kortare ordföljd, alltså enbart det senaste ordet. Prediktorn går från att använda trigram-sannolikheter till bigram-sannolikheter när trigram inte finns, samt motsvarande för bigram till unigram. Detta gör att prediktorn kommer kunna ge rimligare förslag även för osedda eller ovanliga ordföljder från användaren.

En annan intressant aspekt av prediktorn är de två uppsättningarna metoder och hur de skiljer sig. När programmet ser att det föregående ordet är avslutat - senaste input är ett mellanslag - kallas en av tre metoder från den ena uppsättningen på (beror på antal föregående ord). Dessa metoder använder uni-, bi- eller tri-gram-sannolikheter för att generera 3 förslag på nästkommande ord innan användaren börjar skriva på det. Om användaren istället skriver på ett ord kallas en metod av tre från den andra uppsättningen metoder. Dessa kombinerar både n-gram-sannolikhet samt Levenshtein-avstånd (beräknat genom “python-Levenshtein” importerat från Pythons bibliotek)

från den sträng användaren skrivit till möjliga ordförslag för att förutsäga ordet användaren skriver på. Detta fyller två syften, dels bygger förslagen nu på det användaren börjat skriva, (exempelvis kommer “I like d” troligtvis generera “dogs” som ett alternativ, medan endast “I like“ kanske genererar 3 andra förslag på saker folk gillar eller möjligtvis orden “to”, “it”, “when” etc.) och dels används detta i syfte att korrigera stavfel. Om användaren menat att skriva “I like dogs” men råkar skriva “I like dofs” kommer metoden kombinera Levenshtein-avståndet (vilket är 2 mellan “dogs” och “dofs”) samt trigram-sannolikheten för “I like dogs” och generera “dogs” som förslag på ord, vilket förhoppningsvis låter användaren korrigera sitt stavfel.

“TrigramTester.py” är utökad från BigramTester, även den från laboration 2. Den används för att beräkna ett testcorpus kors-entropi gentemot modellen. Den har utökats för att även kunna hantera trigram. Mer om utvärdering nedan.

Data

Datan hämtades från “Mining, analyzing, and modeling text written on mobile devices” (Vertanen, Kristensson, 2021), en intressant studie med syfte att undersöka hur datainhämtning för språkforskning kring skrivande på små enheter kan genomföras. Forskarna webscrapade miljontals inlägg från nätet och använde signaturer för att kunna urskilja de enheter som inläggen skrivits på. Resultatet blev, förutom insikter i hur denna typ av data kan hämtas, flertalet dataset som tillgängliggjordes för mer träffsäker språkmodellering gällande hur folk skriver på olika typer av enheter. Dessutom undersöktes kvaliteten på modeller som tränats på just dessa data jämfört med traditionella modeller som tränats på mer formella skrifter, och visade på högre träffsäkerhet (lägre perplexitet). Ur dessa dataset valdes Iphone- och Android-seten för användning i prediktorn.

Evaluering

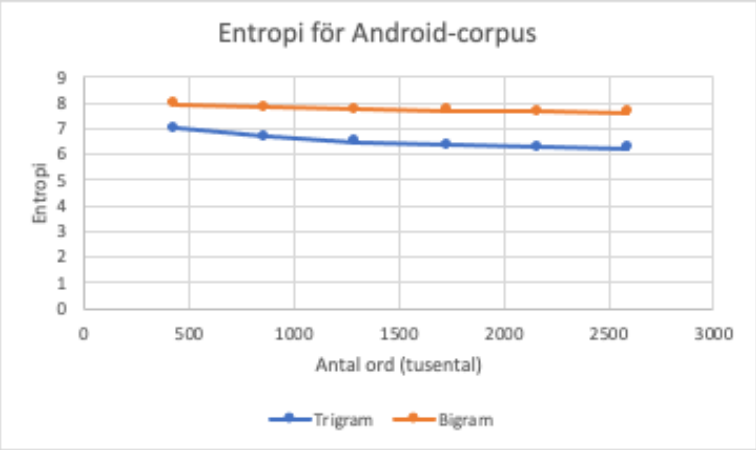
Ordprediktorn evaluerades genom utvärdering av kors-entropin för ett testkorpus givet språkmodellen. Likt i kursens andra laboration använde vi ett testprogram för detta, “TrigramTester.py”.

Evalueringen gjordes på två av dataseten, Android-corpuset samt Iphone-corpuset. Datan innehöll färdiga tränings- och test-set, vilket gjorde evalueringen betydligt enklare. Träningscorpuset delades till en början på 6, som sedan “TrigramTrainer.py” och “BigramTrainer.py” från Labb 2 genererade varsin modell ifrån. Sedan testades dessa modeller i TrigramTester.py gentemot den givna testfilen för respektive corpus och resultaten antecknades. TrigramTester.py utvärderar i första hand baserat på trigram-sannolikheter, om de ej är

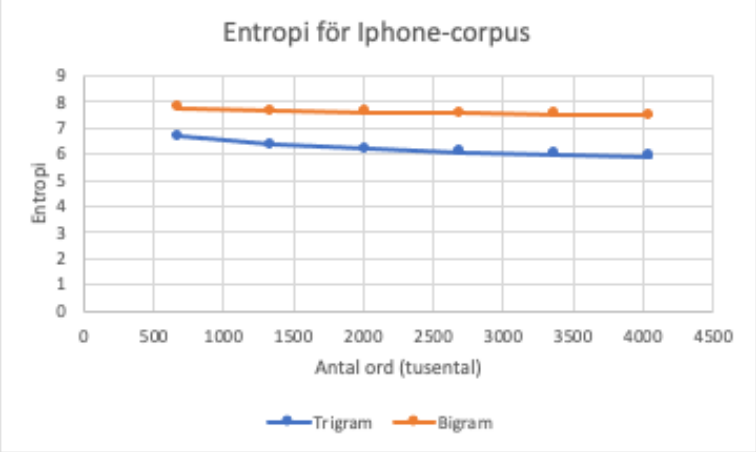
tillgängliga utvärderas bigram-sannolikheter och sedan unigram-sannolikheter, vilket gör att båda modellerna kunde köras med samma fil och samma parametrar & lambdavärden. Träningscorpuset utökades sedan med ⅛ för varje iteration av utvärderingen (6st totalt) varpå processen upprepades.

Resultaten för båda dataseten är sammanfattade nedan.

Resultat



Figur 1: Korsentropi för trigram-språkmodell och bigram-språkmodell med Android-corpus



Figur 2: Korsentropi för trigram-språkmodell och bigram-språkmodell med Iphone-corpus

Resultaten visar att kors-entropin för språkmodellen som nyttjar trigram är lägre än för språkmodellen med enbart bigram och unigram, vilket är i linje med hypotesen och inte särskilt förvånande. Graferna visar även att entropin blir lägre ju större träningscorpus är, vilket beror på att språkmodellerna får ökad förmåga att förutsäga ord träffsäkert ju mer träning de får. Att

skillnaden mellan de olika modellerna tycks bli större när träningscorpus ökar i storlek är även det i linje med förväntan och förklaras genom att större träningscorpus ger mer extensiva språkmodeller vilket höjer prestandan i högre grad för modeller som fångar mer kontext, alltså högre n-gram.

Diskussion

Utvärderingsmetoden som använts är empirisk, verifier- och replikeringsbar samt ger bra resultat som enkelt representeras grafiskt. Med det sagt har den brister, främst i dess validitet. Bland annat är evalueringen egentligen gjord på språkmodellen, inte ordprediktorn. Prediktorn beror på den bakomliggande modellen, så till en stor grad kommer en bra språkmodell leda till en bra ordprediktor, men flera faktorer styr vad som utgör en bra prediktor, exempelvis antal sparade knapptryck och tiden som sparas. Dessutom tillkommer stavningsrättning samt förmåga att lära sig nya ord. Även om prediktorn implementerar just stavningsrättningar och förslag så utvärderas dessa inte, och inte heller antal sparade knapptryck eller tiden användaren sparar genom prediktorn. Optimalt hade varit att ha ännu en evalueringsmetod, exempelvis genom beräkning av antalet sparade knapptryck ordprediktorn ger upphov till, samt möjligtvis en metod för att utvärdera hur ofta prediktorn korrigerar vanligt förekommande stavfel.

Sett till hypotesen har resultaten varit i linje med förväntan. Språkmodellen med trigram har, genom att ta hänsyn till en större kontext, varit bättre på att förutsäga ord och ge förslag än vad en modell med bigram hade varit. Prediktorn har därigenom kunnat föreslå flera ord i följd som i de flesta fall kan formas till en grammatiskt korrekt mening. Det stora och genuina corpus som använts har också varit bidragande till detta, men samtidigt har det haft andra effekter, som att en del slang och vanliga felstavningar blivit del av modellens vokabulär (exempelvis förekommer “t” som ett ord 108 gånger).

Slutsats

Ordprediktorn har implementerats och följt förväntningarna gällande dess effektivitet. Från ett corpus har en statistisk språkmodell utvecklats och implementerats som grund för prediktorn. Vidare har Levenshtein-avstånd tillämpats för effektivare prediktion och stavningsrättning. Prediktorns prestanda har sedan evaluerats genom evaluering av språkmodellen med avseende på kors-entropi, vilket bekräftade vår hypotes. För framtida arbete bör utvärderingen kompletteras med en mer tillämpad utvärderingsmetod för att mäta prediktorns användbarhet samt en funktion som låter prediktorn lära sig ord ur användarens vokabulär och inkorporerar det i modellen.

Appendix

Referenser:

Vertanen K, Kristensson PO. Mining, analyzing, and modeling text written on mobile devices. *Natural Language Engineering*. 2021;27(1):1-33. doi:10.1017/S1351324919000548
https://www.cambridge.org/core/journals/natural-language-engineering/article/abs/mining-analyzing-and-modeling-text-written-on-mobile-devices/A60B599D7E92B5DB9CBDE243A80626C3

Datakälla:

Mobile Text Dataset and language models (Vertanen & Kristensson)
https://digitalcommons.mtu.edu/mobiletext/