Entitetsklassificering med hjälp av aktiv maskininlärning

J O H A N W E S S M A N

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2011

Entitetsklassificering med hjälp av aktiv maskininlärning

J O H A N W E S S M A N

Examensarbete i datalogi om 30 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2011
Handledare på CSC var Stefan Arnborg
Examinator var Johan Håstad

TRITA-CSC-E 2011:118
ISRN-KTH/CSC/E--11/118--SE
ISSN-1653-5715

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Referat

Vi undersöker möjligheterna att använda aktiv maskininlärning vid klassificering av entiteter i text. Traditionell maskininlärning bygger på att man har en större mängd annoterad träningsdata tillgänglig. Detta är dock ett problem då
träningsdata oftast inte finns att tillgå, eller är ämnad för en annan typ av kontext. Man måste då manuellt annotera upp en korpus och denna process kan vara tidskrävande och kostsam. Det är här aktiv inlärning kan användas som ett alternativ.

Vid aktiv inlärning använder man sig av en iterativ träningsprocess där systemet börjar med en mindre mängd annoterad träningsdata, vilken sedan byggs på i varje iteration med, av en mänsklig träningsassistent, rättade exempel. Vi undersöker om det aktivt inlärda
systemet kan prestera likvärdiga resultat som det traditionella systemet undersöks. Ett grundläggande traditionellt system implementeras och tränas upp vars resultat jämförs med ett annat system som tränas upp med aktiv inlärning. Det visar sig att aktivt inlärda klassificerare i stort sett kan uppnå samma resultat som en traditionellt upptränad klassificerare.

Frågan om hur många informativa exempel man vill rätta i varje iteration, och hur resultaten påverkas under träningsprocessen om detta varieras uppstår. För att besvara frågan utförs tre olika aktivt inlärda träningsprocesser med olika parametrar.
Rapporten avslutar med att undersöka hur kontextkänsligt ett aktivt inlärt system är. Vid traditionell inlärning uppkommer problemet med att systemet blir känsligt för ny typ av text som skiljer sig från den typ av text som används för träningsdata. Den bäst presterande
klassificeraren får klassificera några olika texttyper, och det visar sig att ett aktivt inlärt system också presterar sämre på andra texttyper än de som används under träningen.

Abstract

Named Entity Recognition with Active Machine Learning

In this thesis the possibilities of using active machine learning for the task of classification of entities in regular text is examined.
The traditional machine learning approach depends on the availability of a larger set of annotated training data. This poses a problem though, since training data is often not available at all, or it was annotated with another context in mind. A new data set must then be annotated and this process is often costly with respect to both time and money. This is where active machine learning can be a viable option.

During active machine learning an iterative training process is used, and with only a small set of annotated data. This small set is then, with each iteration, increased with further examples annotated by a human training assistant.
We investigate if an actively learned system can perform on the same level as a traditional system. Both a traditional system and a system trained with active machine learning are implemented and compared. The results show that an actively trained system can perform on the same level as a traditional system.

The active machine learning process poses another question in how many examples should be annotated and added in each iteration, and what the implications are if the amount is changed. To answer the question three different active learning processes with different parameters are run.
The thesis ends with an examination of how sensitive a system trained by means of active machine learning is to changes in the types of text on which it is used. In the traditional approach a performance drop can be seen when the classifier is used on other types of text than the text used in the training data. The actively trained classifier with the best performance is used on some different types of text, and the result shows that this problem still exists when using active machine learning.

Förord

Detta examensarbete i datalogi är sista delen mot en civilingenjörsexamen i datavetenskap vid Skolan för datavetenskap och kommunikation vid Kungliga Tekniska Högskolan.

Arbetetet utfördes hos Findwise AB, vid Sveavägen i Stockholm, under våren och sommaren år 2010. Findwise AB arbetar med konsultbaserad verksamhet där de utvecklar söklösningar åt andra företag. De var intresserade av att undersöka möjligheterna med namnigenkänning, och om de skulle kunna använda det i sina lösningar åt kunder.

Till en början var vi två som jobbade tillsammans på systemet och sedan gick vi in på varsin gren och utförde våra egna tester och undersökningar. Det var jag, Johan Wessman, och en kamrat från Kungliga Tekniska Högskolan, Christopher Raschke.
Vi utvecklade alltså tillsammans först ett grundläggande system för entitetsklassificering, vilket involverade att ta fram träningsdata, val av maskininlärningsalgoritm och att bygga upp systemet runt omkring. Systemet involverade då två saker, vilka var att kunna träna upp en klassificerare och till sist också att använda en färdig klassificerare på ny, icke annoterad, text. När arbetet sedan blev individuellt utvecklade jag vidare på den del som tränade upp en ny klassificerare, och då med aktiv inlärning tillagt som träningsprocess.

Mer om Christophers arbete står att läsa i hans examensrapport *Klassificering av Named Entities utan handannoterad träningsdata.*

Innehåll

Kapitel 1

Inledning

Vårt behov av att via datorn förstå vad en text handlar om har vuxit sig större och större, men framförallt ökat efter intåget av internet. Också den stora mängd information och data, i form av text, som större företag arbetar med har ökat detta behov. Enligt Kaiser och Miksch [9] finns över 80% av en organisations kunskaper i form av ren text, men den är inte alltid lätt att komma åt, analysera eller använda. Detta har givit upphov till ett nytt kunskapsområde som brukar kallas för informationsutvinning, eller informationsextrahering (IE härefter), efter engelskans ”Information Extraction”.

Olsson [13] definierar IE som en analyseringsprocess av ostrukturerad text för att ta fram information om förspecificerade typer av entiteter, samt de händelser som dessa entiteter är involverade i, och relationen mellan dessa entiteter och händelser. En central del inom IE är namnigenkänning eller entitetsigenkänning, efter engelskans ”Named Entity Recognition” (NER härefter), vilket både används i sig, men som också andra delar inom IE bygger på.

I dag vill man framförallt ta fram system som är noggranna, robusta och tillräckligt snabba för att kunna användas i annan miljö än den som systemet utvecklades i. Ett stort problem ligger dock i att anpassa systemen så att de kan användas för nya uppgifter samt inom nya domäner.
Eftersom datorer i nuläget inte kan användas för att förstå text till fullo, specificerar man typiskt vilka delar och vilken typ av information som är av intresse.
I detta ligger ett problem. Å ena sidan måste man för att lyckas med IE specifikt och icke tvetydigt ange vilken typ av information som man vill ta fram, å andra sidan leder just detta till att man får svårigheter med att anpassa systemet för användning inom nya domäner och för nya uppgifter.
Enligt Olsson [13], är det just detta som har lett till att man istället försöker bygga upp system med hjälp av metoder för att automatiskt lära systemet från färdiga exempel. Det mest använda sättet för att automatiskt lära upp ett system är med maskininlärning, där inlärningsprogrammet kallas inlärare.

Man tränar upp en inlärare med exempel i form av dokument eller en korpus, vilka innan har annoterats manuellt av en expert.

Dock finns det ett stort problem med att manuellt annotera fram exempel. Dessa system kräver stora mängder annoterad data, vilket dels är tidskrävande men också kostsamt att ta fram [10].
Aktiv inlärning reducerar detta annoteringsarbete genom att automatiskt ta fram exempel ur en mängd av icke annoterad data, för att sedan presentera det för en mänsklig expert som manuellt annoterar det.
Vilket exempel som presenteras baseras på något, eller några, mått som beskriver hur pass informativt exemplet är. Till sist uppdateras modellen med den nya informationen och proceduren görs om igen.
Det finns flera positiva aspekter med aktiv inlärning. Det är framförallt att man, i de flesta fall, drastiskt reducerar antalet annoterade exempel som behövs för att träna ett system, men också att man på samma gång som man får ett tränat system bygger upp en korpus av annoterade exempel.
Enligt Vlachos [15] och Shen et al. [14] har aktiv inlärning redan studerats och använts inom områden som textklassificering, ”Part-of-Speech”-taggning samt parsning.

1.1 Mål med rapporten

Målet med rapporten är att undersöka användningen av aktiv inlärning vid NER i vanlig text. Mer specifikt syftar rapporten dels till att belysa hur pass bra aktiv inlärning står sig i jämförelse med vanlig maskininlärning, men också vilken skillnad som fås när man varierar antalet användargjorda annoteringar i varje iteration.

Till en början förklaras vad NER egentligen innebär och hur det fungerar i praktiken, samt vad det kan användas till inom IE. Det ges också exempel på två huvudangreppssätt för att utföra NER, nämligen handgjorda regelbaserade system och de maskininlärningsbaserade systemen.
Efter det förklaras vad maskininlärning innebär och några av de olika typer av maskininlärningsalgoritmer som kan användas för entitetsigenkänning förklaras.
Svårigheterna med träningsdata gås därefter igenom och det leder in på varför aktiv inlärning är intressant. Det grundläggande konceptet med aktiv inlärning och några olika varianter av aktiv inlärning och hur de fungerar i praktiken förklaras.

Med en förståelse för vad maskininlärning och aktiv inlärning är, kommer sedan rapporten att utreda hur man kan ta fram ett system som med hjälp av dessa metoder hittar namngivna entiteter i en text.
Ett system implementeras sedan med en vald maskininlärningsalgoritm samt en metod för aktiv inlärning, vilket testerna utförs på som ett ”Proof of Concept”.

1.2 Problemformulering och avgränsning

I rapporten undersöks om man med ett aktivt inlärt system kan uppnå lika bra resultat som med ett vanligt system. Utöver det undersöks också hur mycket hjälp som måste ges till systemet i varje iteration för att man ska uppnå bra resultat. Rapporten undersöker dessutom hur typen av träningsdata påverkar systemets prestanda på olika typer av texter.
Rapporten utreder inte hur olika typer av maskininlärningsalgoritmer påverkar resultatet.

Som nämndes tidigare måste man inom aktiv inlärning ta fram de maximalt informativa exemplen för annotering i varje iteration. I denna rapport görs det heller ingen jämförelse mellan olika typer av metoder för att ta fram de maximalt informativa exemplen.
Till sist kan också nämnas att det endast är amerikansk engelska som språk som systemet och rapporten behandlar, men med mindre modifikationer skulle systemet också fungera på ett annat språk.

1.3 Rapportens struktur

Resten av rapporten är organiserad som följer. Kapitel 2 går igenom bakgrunden inom ämnet. Det ska ge en förståelse för vad NER är, samt förklara hur maskininlärning kan användas för att genomföra detta.
Dessutom förklaras hur aktiv inlärning fungerar, och kapitlet avslutas med exempel på några existerande system.
I Kapitel 3 beskrivs hur systemet är uppbyggt och vilka komponenter som ingår, samt vilka algoritmer och metoder som används.
Med systemets arkitektur klar, förklaras de olika testerna som genomfördes, och till sist redovisas vilka resultat som de gav.
De två sista kapitlen, Kapitel 4 och Kapitel 5, går sedan igenom slutsatserna baserade på de erhållna resultaten och till sist vad som vore intressant som framtida arbete.

Kapitel 2

Teoretisk Bakgrund

Områden som NER, maskininlärning och aktiv inlärning är breda områden som kräver en djupare genomgång. I detta kapitel förklaras det mer ingående om de olika områdena och de begrepp som är viktiga att förstå.

2.1 Named Entity Recognition

Som nämndes tidigare vill man inom IE ta fram specifik information ur en text, det vill säga ta fram vilka entiteter och händelser som figurerar i texten och hur dessa entiteter samspelar, eller hur de är involverade i en viss händelse.
Men för att kunna avgöra detta måste man först ta reda på vilka ord som är entiteter och av vilken typ de olika entiteterna är. Det är detta som brukar kallas för namnigenkänning eller ”Named Entity Recognition” på engelska (NER).

NER är alltså en av de grundläggande delarna inom IE och enligt Jurafsky och Martin [8] är det oftast det första steget man arbetar med för att utveckla IE-system.
Generiska NER-system fokuserar främst på att upptäcka entiteter som personer, företag och organisationer, men det är helt upp till den som designar systemet att definiera vilka typer av entiteter man vill kunna upptäcka.
Jurafsky och Martin [8] påpekar dock att systemen oftast utökas för att behandla ordtyper som inte är namngivna entiteter per definition, men som är praktiskt viktiga. Det inkluderar till exempel datum, tid, namngivna händelser, men också mått, penningbelopp och procenttal.

Ett typiskt exempel på en mening med entiteter utmärkta:

<ENAMEX TYPE=”PERSON”>Bill Gates</ENAMEX> stepped down as chief executive officer of <ENAMEX TYPE=”ORGANIZATION”>Microsoft</ENAMEX> in <TIMEX TYPE=”DATE”>January 2000</TIMEX>.

Det har dock växt fram en mer allmän definition av IE och NER, vilket också gäller de entiteter som man vill kunna upptäcka. Dessa har utvecklats under det sena 80-talet och under 90-talet vid en serie konferenser vid namn MUC (Message Understanding Conference).
De entiteter som definierats är indelade i tre grupper, vilka är *ENAMEX*, *TIMEX* och *NUMEX*. Inom varje grupp definieras ett flertal entitetstyper, närmare bestämt finns inom *ENAMEX* personer, organisationer och platser, inom *TIMEX* datum och till sist inom *NUMEX* procenttal och penningbelopp.

Vad kan då NER användas till mer konkret? Enligt Borthwick [3] används det exempelvis till:

- Mer träffsäkra sökmotorer. För att lättare kunna gå igenom resultat om det är en person man söker, och det finns svar som samtidigt refererar till en plats med samma namn.
- Lättare organisering av dokument. Om alla entiteter som är kända i ett dokument finns tillgängliga, kan man tekniskt sett filtrera på en viss individs namn för att hitta alla dokument som nämner den individen.
- Innan man granskar texten skulle en lista med entiteter som nämns i texten kunna visas.
- För att automatiskt indexera böcker, då det oftast är de namngivna entiteterna själva som står i bokens index.
- Ett första steg till andra mer avancerade uppgifter inom IE.

Systemen för NER har förändrats mycket sedan forskningen inom området påbörjades. Till en början utvecklades endast handgjorda system, men senare övergick utvecklingen till automatiserade lösningar med maskininlärning.
De två följande sektionerna behandlar kort vad de två områdena innebär.

2.1.1 Handgjorda system

De handgjorda systemen är, som namnet föreslår, framtagna för hand av mänskliga designers, och bygger mycket på deras intuition, skriver Borthwick [3]. Han tar som ett exempel ett system kallat ”Proteus”, skrivet i Lisp som presenterades under MUC-6. Det systemet var uppbyggt av ett stort antal reduktionsregler som alla i sig var kontextspecifika och därmed fungerade sämre för nya typer av texter.

Borthwick [3] ger också exempel på några regler ur det systemet, och visar vad de gör rätt och vad de gör fel.

Såhär ser en av dem ut:

* from Date to Date => Date

- **Korrekt:** from August 3 to August 9, 1997

- **Inkorrekt:** We moved the conference from April to June to allow more time for preparation (här ska April och June taggas separat som datum men denna regel kommer felaktigt att tagga dem ihop som ett datum).

Detta är ett typiskt exempel på en handskriven regel och på hur det kan bli fel. Borthwick [3] nämner till och med att det i de flesta fall finns ett flertal undantag till reglerna, som manuellt också måste anges för att systemet ska göra rätt. Vanligtvis är det en omöjlighet att inom rimliga tidsramar kunna identifiera alla undantag och skriva speciella regler för dem.
Problemen som gjort att fokus har skiftat ifrån de handgjorda systemen till de automatiserade är främst att det är dyrt att bygga upp handgjorda system. Det krävs utbildade lingvister för att skriva reglerna och upptäcka de undantag som kan förekomma, och dessutom tar det lång tid att skriva reglerna.
Det uppstår dessutom problem med att byta domän som systemet skall användas på, då många av reglerna kan behöva skrivas om för en annan typ av text. Liknande problem uppstår om systemet skall användas för ett nytt språk, då regler och lexikon måste skrivas om.

2.1.2 Maskininlärningsbaserade system

För att undvika de svårigheter som uppstår med handgjorda system måste man på något sätt bygga ett system som tränar upp sig självt, och på så sätt minska kostnaden.
Det är här som maskininlärning kommer in. Det finns flera olika algoritmer med vilka man kan bygga upp ett maskininlärt system, men de har alla några aspekter gemensamt. Alla metoderna kräver att en stor mängd annoterade exempel som systemet tränas upp med finns tillgängligt. Det innebär att alla entiteter som förekommer i texten som skall annoteras taggas ut. Under träningen går sedan systemet igenom ett exempel i taget och tränas upp på hur kontexten runt varje träningsinstans i texten ser ut.

Borthwick [3] nämner att det är naturligt att ifrågasätta denna metod. Byts inte en kostsam och lång process ut mot en annan process som också tar lång tid? Dock är skillnaden större än vad en första anblick föreslår, och han skriver att en jämförelse kan göras mellan de fåtal dagar det tar att annotera en träningskorpus för maskininlärning, mot de månader som det tar att utveckla ett handgjort system. Dessutom kräver inte annoteringen att det är lingvister som gör det.

2.2 Maskininlärning

Inom maskininlärning försöker man svara på frågan hur man skall konstruera program som automatiskt förbättras med hjälp av erfarenhet.
Det är ett brett ämne som utnyttjar kunskap och resultat inom många olika områden. Enligt Mitchell [12] inkluderar det områden som statistik, artificiell intelligens, filosofi och informationsteori, för att nämna några.

Mitchell [12] definierar maskininlärning som:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Representationen av den erfarenhet som programmet skall lära sig av brukar alltid ligga nära uppgiften i sig. Som ett exempel kan tas ett schackspelarprogram, där erfarenheten kan beskrivas av ett antal partier med aspekter såsom antalet pjäser på bordet, vilka positioner pjäserna har just nu och vilket det senaste motståndardraget var.
Erfarenheter brukar beskrivas av en vektor med olika attribut. Varje attribut motsvarar en del information om omständigheterna för just den erfarenheten, som i schackexemplet ovan.
Som ett exempel representeras en erfarenhet $e \in E$, ett program ska lära sig från, av en följd attribut,

$$(f_1,f_2,...,f_k)$$

där f_1 till f_k är värdena för attributen 1,...,k som beskriver erfarenheten e. Ett av dessa attribut i erfarenhetsvektorn, till exempel det sista, förutspås och de värden som detta attribut kan anta motsvarar de olika klasserna. Instansvektorn är då

$$(f_1,f_2,...,f_{k\text{-}1})$$

och det gäller att förutspå det sista attributets (f_k) värde, givet de resterande *1,...,k-1* attributen. Värt att notera är att erfarenhetsvektorn innehåller attributet f_k, men att en instans inte gör det.
Erfarenheter som systemet ska lära sig av brukar kallas för träningsexempel, och då en erfarenhets klass skall förutspås brukar det kallas för en instans. Då uppgiften består i att förutspå en av en diskret mängd diskreta kategorier, utan relativ ordning, brukar inlärningsprocessen kallas för klassificering.

Traditionellt finns det två huvudgrenar inom maskininlärning: övervakad inlärning och icke övervakad inlärning [13]. I övervakad inlärning brukar erfarenheterna vara par bestående av exempelvektorn och det korrekta värdet, eller klassen, som associeras med exemplet. I icke övervakad inlärning däremot, så ges inte träningsexemplen till inläraren tillsammans med det korrekta värdet för det exemplet, utan

inläraren tittar istället på träningsdatan för att hitta mönster och strukturer som den sedan kan basera beslut på.
Dock är det endast övervakad inlärning som beskrivs i denna rapport, och härefter förklaras några av de maskininlärningsalgoritmer som kan användas för just NER.

2.2.1 Beslutsträd

Beslutsträd inom maskininlärning kan beskrivas som en riktad acyklisk graf där noderna representerar ett specifikt attribut, grenarna mellan noder och deras barn representerar värden för den nodens attribut och till sist löven i trädet vilka representerar de diskreta värdena för en målklass.
Träningen av beslutsträd går inte till på samma sätt som träningen hos andra maskininlärningsalgoritmer, utan ett beslutsträd byggs upp genom att analysera träningsexemplen. När trädet skapas används det attribut som, på egen hand, bäst förutspår de korrekta klasserna för den tillgängliga träningsdatan som rot i trädet. Trädet byggs sedan på med en gren, från roten, för varje möjligt värde som rotens attribut kan anta. De tillgängliga träningsexemplena delas därmed upp över alla grenar beroende på det värde som attributet antar i just det exemplet.
Proceduren utförs för varje gren, dock bara på den delmängd träningsexempel som tillhör grenen i fråga. Ett nytt attribut som bäst förutspår korrekt klass för de resterande exemplena väljs på nytt, och sätts som nästa nod i det delträdet. Detta fortsätter tills alla exempel vid en viss nod tillhör samma målklass.
Trädet kan ses som ett stort antal *Om-Så*-frågor, där varje fråga (nod), leder till ett mer specifikt svar.

Vid klassificering av en instans med hjälp av beslutsträd vandrar man alltså från trädets rot mot löven. Vid varje nod jämförs de värden som instansens attribut har med de som är tillgängliga, ända tills ett löv nås vilket visar vilken klass instansen tillhör.

2.2.2 Artificiella Neuronnät

Artificiella neuronnät, eller ANN (engelskans Artificial Neural Networks), är en grupp linjära och ickelinjära statistiska verktyg som kan användas för antingen klassificering eller regression genom att modellera relationen mellan indata och utdata [13].
Metoden försöker efterlikna hur neuronerna i den mänskliga hjärnan kommunicerar med varandra, i ett stort nätverk, Haykin [6].
Ett ANN kan beskrivas som en graf där noderna (neuronerna) är sammankopplade av ett stort antal kanter. Ett ANN i sin helhet modellerar en funktion som omvandlar nätverkets indata, (de attribut som beskriver ett exempel till det aktuella problemet), till utdata. Denna funktion består i sin tur av en kombination delfunktioner som alla omvandlar indata vid en specifik nod i nätverket till utdata för samma nod. Hur alla delfunktioner kombineras ihop till nätverkets slutliga funktion

bestäms av vikter, och det finns en vikt för varje kant [13].

När ett ANN ska tränas byggs först ett nätverk designat för uppgiften i fråga upp, och sedan bestäms vikterna för alla kanterna i nätverket baserat på tillgängliga träningsexempel. Nätverkets utformning av utdata skiljer sig ofta åt beroende på vilken uppgift nätverket har. I fallet med regression, beräknas nätverkets utdata av en linjärkombination av utdatalagrets noder. Vid klassificering däremot måste en tröskelfunktion användas för att få nätverkets slutliga utdata. På så sätt fås ett mindre område värden fram från en större domän av utdatavärden.

ANN har använts till en mängd praktiska problem som till exempel igenkänning av skrivna bokstäver, igenkänning av talade ord och ansiktsigenkänning [12].

2.2.3 Multinomial Logistisk Regression, eller Maximum Entropy

Vid vanlig regression är målet som bekant, givet ett antal observationer där varje observation associeras med ett antal attribut, att kunna ge ett reellt värde som utdata. I språkteknologi är målet att kunna klassificera en observation. Det förutsätter att utdatavärdena är en mindre mängd, till exempel sant eller falskt. Detta brukar kallas för logistisk regression och bygger vidare på vanlig regression, men ger istället ett distinkt värde från en mindre mängd utdata, samt också en sannolikhetsfördelning över de möjliga utdatavärdena.
Som i detta fall, då ett större antal klasser ska kunna förutspås används istället en ytterligare påbyggnad av logistisk regression, som brukar kallas multinomial logisitisk regression, eller Maximum Entropy (ME härefter), Jurafsky och Martin [8]. Som nämndes innan ger alltså ME en sannolikhetsfördelning över klasserna för varje instans. Detta utmärker en probabilistisk klassificerare. Sannolikheten används sedan för att basera ett beslut på, och kan till exempel vara att den klass som instansen har högst sannolikhet att tillhöra väljs. Ett annat sätt är att titta på en mening åt gången och sedan använda en avkodningsalgoritm som till exempel Viterbis algoritm, för att hitta och välja den sekvens av klassificeringar som har störst sannolikhet att uppkomma, [3, 1].

Man arbetar som nämndes tidigare med ett antal attribut som beskriver en erfarenhet. Vid ME har varje kombination av attribut f_i, och klass c, en vikt w_{ic} vilka sedan kombineras ihop för att ta fram en sannolikhet för den klassen, och den klass som är mest trolig. Inom ME ges sannolikheten för en klass c givet en observation x med attribut f_i av

$$p(c|x) = \frac{1}{z} \exp\left(\sum_i w_{ic} f_i\right) \tag{2.1}$$

där z är en konstant för att alla sannolikheterna korrekt ska summeras till ett. Detta är dock en förenklad version av den riktiga ekvationen, vilken kommer att ges senare.

Summan innanför exponenten känns igen från vanlig linjär regression.

Logistisk Regression

Logistisk regression bygger vidare på linjär regression och används, som sades ovan, i det fallet då ett distinkt värde, av en mindre mängd diskreta värden, istället för reella värden skall förutspås, och detta brukar kallas för klassifikation [8]. Som ett exempel kan tas det enklaste fallet med binär klassifikation, det vill säga då det ska avgöras om en observation x är sann *(y = sann)*, eller falsk *(y = falsk)*. Målet är alltså i detta fall att utdata y antar värdena 1 eller 0 för sant och falskt respektive, och dessutom att få en klassificerare som tar en ny observation x och ger värdena 0 (falsk) eller 1 (sant) som utdata. Dessutom skall modellen också kunna ge sannolikheterna över om observationen tillhör klass 0 eller klass 1, betingat av träningsdata och nya instansattribut. Används den linjära modellen

$$P(y = 1|x) = \sum_{i=0}^{N} w_i f_i \tag{2.2}$$

uppstår dock problem, men om man för en stund bortser från det, hur skulle det se ut om man antar att det skulle fungera felfritt? En sådan modell skulle kunna tränas genom att för varje observation ange målklassen, det vill säga *y=1* om observationen tillhör klassen eller *y=0* om observationen inte tillhör klassen. Varje observation x skulle ha en attributvektor f, och viktvektorn w skulle tränas så att det förutspådda felet skulle minimeras. Efter träning skulle man sedan givet en observation ta inre produkten mellan observationens attributvektor med viktvektorn, $(w \cdot f)$, för att på så sätt få sannolikheten för en klass.
Som nämndes innan uppstår dock ett problem med denna modell, och det är att det inte finns en gräns för vilka värden som utdata kan anta. Det finns inget som tvingar utdata att vara en giltig sannolikhet mellan 0 och 1, utan det kan vara värden mellan $-\infty$ och ∞.
Enligt Jurafsky och Martin [8] löses detta genom att, istället för att förutspå en sannolikhet, förutspå logaritmen för oddsen mellan utfallen. Odds definieras som förhållandet mellan sannolikheten att observationen tillhör klassen och sannolikheten att observationen inte tillhör klassen.
Om detta sättes in i den linjära modellen fås då, med oddsen för att y är sann, följande:

$$\log\left(\frac{p(y = 1|x)}{1 - p(y = 1|x)}\right) = w \cdot f \tag{2.3}$$

Vidare förenkling ger sedan de slutliga ekvationerna:

$$p(y = 1|x) = \frac{e^{w \cdot f}}{1 + e^{w \cdot f}} \tag{2.4}$$

och

$$p(y = 0|x) = \frac{1}{1 + e^{w \cdot f}} \tag{2.5}$$

När nu dessa ekvationer härletts, hur tränas vikterna i w upp givet en mängd träningsobservationer? Det som sökes är de vikter w som maximerar sannolikheterna för de observerade y-värdena i träningsdatan, givet de tillhörande observationerna x. För den i:te observationen $x^{(\mathrm{i})}$, väljs vikterna i praktiken enligt:

$$\hat{w} = \arg\max_{w} P(y^{(\mathrm{i})}|x^{(\mathrm{i})}) \tag{2.6}$$

Detta är dock endast för en observation, och målet är att hitta de optimala vikterna för hela träningsmängden, det vill säga över produkten av sannolikheterna:

$$\hat{w} = \arg\max_{w} \prod_{i} P(y^{(\mathrm{i})}|x^{(\mathrm{i})}) \tag{2.7}$$

Man brukar dock arbeta med det högra ledet logaritmerat, för att på så sätt kunna använda sig av logaritmreglerna för att bryta isär produkten till en summa av logaritmerade termer. På så sätt fås en enskild term för varje utdatavärde och dessa ser innanför logaritmen, i det binära fallet, ut som de härledda ekvationerna (2.4) och (2.5).
Substitueras dessa termer logaritmerade in i (2.7) och förenklas fås till slut en ekvation att träna på ut. I detta fall får vi:

$$\hat{w} = \arg\max_{w} \sum_{i} y^{(\mathrm{i})} \log \frac{1}{1+e^{-w\cdot f^{(\mathrm{i})}}} + (1-y^{(\mathrm{i})}) \log \frac{e^{-w\cdot f^{(\mathrm{i})}}}{1+e^{-w\cdot f^{(\mathrm{i})}}} \tag{2.8}$$

För att träna metoden används oftast en annan form av algoritm som till exempel *Gradient Ascent*, *Conjugate Gradient* eller nån form av iterativ skalningsalgoritm. Dessa bygger på att man iterativt förändrar vikterna och avslutar denna process när ett visst kriterium uppnåtts, som antas visa att det inte lönar sig att fortsätta.

Multinomial Logistisk Regression

Vanlig logistisk regression fungerar alltså vid klassificering av en observation i två klasser. Dock består de flesta klassificeringsproblem i praktiken av en större mängd klasser. Logistisk regression kan för dessa fall också användas men måste då definieras om, och brukar då kallas för multinomial logistisk regression [8]. Det antas då att utdata y tillhör en av C klasser c_1,c_1,...,c_C. Som nämndes tidigare används ekvation (2.1) för att estimera sannolikheten att y tillhör klassen c, det vill säga

$$p(c|x) = \frac{1}{z} \exp\left(\sum_{i} w_{ic} f_i\right) \tag{2.9}$$

I detta fallet med multinomial logistisk regression fås alltså en uppsättning vikter för varje klass, det vill säga vikterna beror på klassen c. För diskussionens skull

sätts också antalet attribut som varje observation innehåller till N stycken.
De enskilda uträkningarna för en klass normaliseras. Normaliseringsfaktorn z är

$$z = \sum_{c \in C} p(c|x) = \sum_{c' \in C} \exp\left(\sum_{i=0}^{N} w_{c'i} f_i\right) \tag{2.10}$$

och detta ger alltså insatt i (2.11) den slutliga ekvationen

$$p(c|x) = \frac{\exp\left(\sum_{i=0}^{N} w_{ci} f_i\right)}{\sum_{c' \in C} \exp\left(\sum_{i=0}^{N} w_{c'i} f_i\right)} \tag{2.11}$$

För att sedan träna och klassificera med modellen går man till väga på samma sätt som med boolesk logistisk regression vilket beskrevs tidigare. Skillnaden består i att en sannolikhet nu fås ut för varje klass, och därmed en sannolikhetsfördelning klasserna. Skall en definitiv klass väljas för en instans kan till exempel den klass vars sannolikhet var störst för den givna instansen väljas, men oftast brukar man titta på en längre sekvens av instanser. I det fallet väljs den sekvens av klassificeringar som ger störst sannolikhet för hela instanssekvensen.

2.2.4 Attribut i praktiken

Tidigare i detta kapitel har begrepp som erfarenhet och attribut förklarats, men vad består ett attribut av i praktiken, och mer specifikt vad utgör ett attribut när man arbetar med NER?
Varje attribut ska på något sätt beskriva en del av kontexten för ett exempel eller en instans. Det kan alltså innehålla information om hur ordet i sig är uppbyggt. Finns det några siffror i det aktuella ordet? Finns det några versaler i ordet? Men ett attribut kan också bero av orden runt omkring. Vilka ordklasser har orden framför och efter det aktuella ordet?

De flesta attribut ger någon form av binär utdata som svar. Som ett exempel kan man ta: Om ordet i fråga har versal: attributet blir 1 och sant, annars 0 och falskt. Nedan följer en genomgång av de mest använda formerna av attribut.

Binära attribut

Den vanligaste formen av attribut som används inom NER, och som nämns av Borthwick [3], Bender et al. [1], Borthwick et al. [2], är binära attribut.
Dessa består vanligtvis av [3, 1]:

- Är det ett tvåsiffrigt tal?
- Är det ett fyrsiffrigt tal?
- Är ordet en blandning av bokstäver och siffror?

- Består ordet av endast versaler?
- Är första bokstaven i ordet en versal?
- Har ordet ett specifikt pre- eller suffix?

Lexikala attribut

Denna typ av attribut är också vanliga, och brukar titta på specifika ord som i språket ofta förekommer runt omkring de olika typerna av entiteter [3]. Till exempel förekommer ordet ”till”, i svenskan, ofta framför personer eller platser. Det kan därför vara bra att använda sig av ett attribut som ger en etta som utdata om ordet före är ordet ”till”, då det tyder på att ordet i fråga borde vara en person eller plats. Denna typ av attribut liknar de regler som används i handgjorda system, och måste därför bytas ut om systemet skall användas på nya domäner.

Sektionsattribut

Som Borthwick et al. [2] och Borthwick [3] beskriver det kan också attribut som beror på i vilken del av texten som ordet står användas. Då behövs därför ett attribut per sektionstyp i de texter som systemet ska användas för. Eftersom varje ord i en sektion av texten aktiverar det motsvarande attributet ger inte denna typ av attribut hög precision, men är ändå viktiga för att skilja klasser åt [2]. Dessutom kräver denna typ av attribut att just texter som är uppdelade i sektioner som tidningsartiklar och liknande används.

Listattribut

Denna typ av attribut kräver att listor med olika typer av entiteter finns tillgängliga. Det kan vara listor med personnamn, företagsnamn, universitetsnamn, länder, städer, förkortningar och datum [3]. Det behövs alltså ett attribut per lista som aktiveras då ett ord finns med i den listan.

2.3 Aktiv inlärning

Inom aktiv inlärning tränas inte inlärningsalgoritmen på all den tillgängliga träningsdatan, som i vanlig inlärning, utan det är inlärningsalgoritmen själv som har kontroll över vilken del av data som den ska lära sig av. Denna kontroll innebär att inläraren själv väljer ut de exempel ur data som den för tillfället är mest osäker på, och dessa presenteras sedan för ett orakel, vilket oftast är en människa, som berättar för inläraren vad som är felaktigt.

Metoden behöver två mängder av data till att börja med, en lite mindre mängd data som redan är uppmärkt och en större mängd av icke uppmärkt data. Metoden producerar en upptränad klassificerare, och en mängd nyuppmärkt data.

En typisk algoritm för aktiv inlärning [13]:

1. Initiera processen med att träna inlärningsalgoritm B på annoterade träningsdatamängden D_I, och erhåll klassare C.
2. Använd C på icke annoterade datamängden D_U och erhåll den nya datamängden D_U'.
3. Identifiera de n mest informativa instanserna, (till exempel de mest osäkra), från D_U', och kalla dessa för I.
4. Fråga oraklet om klassningar av alla instanser i I.
5. Flytta instanserna i I, med klassningar, från D_U' till D_I.
6. Träna om B på D_I för att erhålla en ny klassare C'.
7. Upprepa steg 2 till 6, tills D_U är tom eller ett stoppkriterie har uppnåtts.
8. Utdata blir sedan en klassare tränad på D_I.

I de flesta fall är metoden alltså en iterativ process som utförs tills att ett visst stoppkriterium uppnåtts. Två exempel på aktiva inlärningsmetoder förklaras i nästa sektion.
Några exempel på områden inom vilka aktiv inlärning har använts med positivt resultat är [13]:

- Named Entity Recognition, NER.
- Textkategorisering.
- "Part-of-Speech"-taggning.
- Parsning av text.

2.3.1 Olika typer av aktiv inlärning

Det finns som nämndes tidigare olika typer av metoder för aktiv inlärning och det som skiljer dem åt är i hur de mest informativa exemplen väljs ut. Två av de vanligast förekommande varianterna är antingen att de mest informativa exemplen väljs med hjälp av någon form av osäkerhetsmått, eller att man använder sig av en kommitté av klassare, och tar ut de exempel som de svarar mest olika på.

Aktiv inlärning med hjälp av osäkerhetsmått

Vid aktiv inlärning med hjälp av osäkerhetsmått tränas en klassare upp från den tillgängliga mängden annoterad träningsdata och denna används sedan för att undersöka den icke annoterade träningsdatan. Därefter väljs de träningsinstanser som klassaren är mest osäker på ut från mängden och dessa presenteras sedan till en mänsklig annoterare [13].
Den allmänna algoritmen presenterad i sektion 2.3 används, men det antas att de

mest informativa instanserna är de som klassaren är mest osäker på.

Vilket typ av osäkerhetsmått som används kan variera och det finns inga universellt använda. Shen et al. [14] nämner dock tre olika kriterier som kan användas när man bestämmer sitt mått, och dessa är informativitet, representativitet och variation. Informativitet ges oftast av hur långt den givna instansens sannolikhetsfördelning, över de givna klasserna, är från den likformiga sannolikhetsfördelningen. Shen et al. [14] använder sig av Support Vector Machines (SVM härefter) som inlärningsalgoritm. Vid denna metod byggs ett hyperplan som skall separera de olika klasserna upp. Man söker efter ett antal supportvektorer, vilka ligger på två parallella hyperplan som bäst separerar träningsmängden, och centrerat mellan dessa två parallella hyperplan hittar man det hyperplan som separerar klasserna. I det fallet tas de instanser vars attributvektorer ligger närmre hyperplanet än supportvektorerna ut, och dessa instanser är de mest informativa/osäkra.
Hur pass representativ en instans är ges av hur många träningsinstanser i träningsdata som liknar den valda instansen. Ju fler instanser som har liknande attributvektorer och målklass, desto mer representativ är en instans för träningsmängden. Till sist använder de sig av variation, och i detta fall över de instanser som väljs ut. De instanser som har så stor variation som möjligt med hänsyn till varandra väljs ut [14].

Aktiv inlärning med hjälp av kommitté

Denna metod bygger, precis som metoden med osäkerhetsmått, på sampling av instanser ur en mängd av icke annoterad träningsdata [13]. Kommittémetoden använder sig dock av flera klassare som bildar den så kallade kommittéen.
Klassarna tittar alltså alla på samma instanser och det som avgör vilka de mest informativa instanserna är, är hur pass oeniga klassarna är med hänsyn till vilken klass de tror att instansen i fråga tillhör. Själva idén med en kommitté av klassare bygger dock på antagandet att man har en hög varians mellan klassarnas upptränade hypoteser [13]. Om alla klassarna var exakt likadant upptränade skulle ingen oenighet råda mellan dem, och alltså ingen beslutsgrund finnas.
Denna typ av aktiv inlärning ser inte exakt ut som den allmänna. Den grundläggande algoritmen ser istället ut som:

1. Initiera processen med att träna inlärningsalgoritm B på annoterade träningsdatamängden D_L, och erhåll kommittéen av klassare C.
2. Använd varje klassare i C på icke annoterade datamängden D_U och erhåll den nya datamängden D_U'.
3. Välj ut de n mest informativa instanserna från D_U', och erhåll D_U".
4. Fråga oraklet om klassningar av alla instanser I från D_U".
5. Flytta I, med klassningar, från D_U" till D_L.
6. Träna om B på D_L för att erhålla en ny kommitté av klassare C.

7. Upprepa steg 2 till 6, tills D_U är tom eller ett stoppkriterie har uppnåtts.
8. Utdata blir sedan en klassare tränad på D_L.

2.3.2 Kort om stoppkriterium

Det finns många olika angreppssätt till hur inlärningsprocessen skall avslutas med aktiv inlärning, men ingen metod som är universellt erkänd som bäst. De vanligaste angreppssätten inkluderar metoder såsom att sätta ett mål på träffsäkerheten (mer om träffsäkerhet i avsnitt 2.4) och stanna processen när det målet har nåtts, att köra processen ett bestämt antal iterationer, att fortsätta tills mängden med icke annoterad träningsdata tagit slut, eller att fortsätta processen tills prestanda planar ut [15].
En mer avancerad variant är att titta på hur träffsäkerheten förändras mellan iterationer. Om träffsäkerheten avstannar vid en ungefärlig nivå, eller om den rent av minskar stannar man inlärningsprocessen.

En annan metod är att undersöka hur klassifikationerna av, den för tillfället upptränade, klassaren på den icke annoterade träningsdatan förändras mellan iterationer. Om två efterföljande iterationer av klassare förutspår samma klass på alla, eller näst intill alla, instanser i träningsdata så avbryts inlärningsprocessen. Det bygger på antagandet att de mest informativa instanserna är de som förändras mest mellan iterationer.

Vlachos [15] skriver om en annan metod som tittar på hur säker klassaren är och använder det som ett stoppkriterium. I varje iteration låter man alltså den tillfälligt upptränade klassaren gå igenom och förutspå målklasser på en separat testmängd med icke annoterade instanser. Idén är att man stoppar träningsprocessen då klassarens noggrannhet, över testmängden, håller sig på en stadig nivå eller försämras under ett givet antal av iterationer.

2.4 Prestationsmått inom språkteknologi

Hur utvärderar man sedan sitt system när det är upptränat och klart? Det finns ett antal mått som normalt brukar användas inom NER men också i allmänhet inom språkteknologi.
När systemet utvärderas bör inte samma data som användes vid träningen användas. De två bör vara separerade för att undvika överinlärning [13]. Överinlärning kan uppkomma då samma data används för träning som för prestationstest. Det kommer sig av att systemet normalt lär sig specialfall och egenheter i träningsdata som inte är typiska för annan data, och resultatet blir då att ett sämre resultat fås då systemet används på icke tidigare sedd data.

I denna rapport används tre mått på prestation, vilka är *Precision*, *Recall* och

F-värde. Ett fjärde mått som ibland används är träffsäkerheten, eller *Accuracy*.
Alla dessa mått har gemensamt att de defineras av fyra termer, nämligen *true positives* (TP), *true negatives* (TN), *false positives* (FP) och *false negatives* (FN). Hur många som förekommer av varje term efter att klassaren har klassat testmängden anger de olika måtten.
Till TP hör alla instanser som klassaren korrekt klassade som tillhörandes en viss klass. På samma sätt hör till TN alla instanser som korrekt klassades att inte tillhöra en given klass. FP består av det antal instanser som felaktigt klassades att tillhöra en given klass, och till sist ges FN av alla de instanser som felaktigt klassades att inte tillhöra en given klass.

Med hjälp av dessa kan de olika prestationsmåtten beräknas. Måtten antar värden mellan 0 och 1, där de högre värdena är de eftersökta. Dessutom brukar dessa värden skrivas som en procentsats.

Träffsäkerheten (efter engelskans *Accuracy*) ges av antalet korrekt klassificerade instanser, det vill säga delat på alla klassificeringar, enligt:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.12}$$

Precisionen (efter engelskans *Precision*) definieras som förhållandet mellan antalet korrekt klassificerade instanser och det totala antalet instanser som klassificerats till en klass:

$$Precision = \frac{TP}{TP + FP} \tag{2.13}$$

Återkallelsen (efter engelskans *Recall*) definieras av förhållandet mellan antalet korrekt klassificerade instanser och det totala antalet instanser klassificerade att tillhöra denna klass:

$$Recall = \frac{TP}{TP + FN} \tag{2.14}$$

Det sista måttet som brukar användas är F-värdet (efter engelskans F-measure). Det är det harmoniska medelvärdet av *Precision* (P) och *Recall* (R) enligt:

$$F = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R} \tag{2.15}$$

där β är en konstant som används för att vikta ihop P och R för att ta fram hur mycket de två måtten influerar varandra. Vanligtvis används $\beta = 1$, det vill säga det vanliga harmoniska medelvärdet, och det är också detta som används i denna rapport. Om man till exempel fokuserar på att ett system främst skall ha en hög precision kan man sätta ett annat β-värde som viktar in att precisionens värde skall bidra mer till F-värdet än återkallelsens, och på så sätt märks förändringar av precisionen av mer i det slutliga F-värdet.

Kapitel 3

Systemarkitektur, tester och resultat

Målet med arbetet är att undersöka användandet av aktiv inlärning vid NER. För att kunna utföra tester och undersöka aktiv inlärning utvecklades först ett system i två versioner, nämligen med och utan aktiv inlärning.
I denna sektion förklaras hur dessa två system är uppbyggda, och vad de olika delarna i systemen gör. Dessutom förklaras vilka typer av attribut som användes, hur målklasserna formulerades samt vilken träningsdata som användes.

3.1 Metod och systemarkitektur

Som nämndes tidigare konstruerades två system med skillnaden att det ena tränades upp på vanligt vis med all tillgänglig träningsdata, och det andra med hjälp av aktiv inlärning.
I denna sektion beskrivs de två olika systemen, vilka delar som de är uppbyggda av och hur varje del är implementerad.
Dock har de typen av träningsdata som används, vilka attribut som systemen tränades på gemensamt och vilka målklasser, eller entitetstaggar, som användes. Dessa genomgås därför här.

Träningsdata

Ett av de stora problemen med NER är bristen på tillgänglig träningsdata, och detta problem visade sig också i detta arbete. Vi försökte till en början att hitta färdiga träningsmängder som använts tidigare, från konferenser som till exempel MUC, men lyckades tyvärr inte att hitta en sådan mängd som var öppet tillgänglig. Efter att inte ha hittat någon tillgänglig annoterad text på engelska byggde jag och Christopher upp en egen annoterad korpus.
Som grund användes en kopia av engelska Wikipedia, som kan hittas på `http://dumps.wikimedia.org/backup-index.html`, [7].
Vi använde oss sedan av ett flertal listor över vanliga och kända personnamn, städer, länder, företag/organisationer och vanligt förekommande datum- och tidsformat (vilket inkluderade namngivna datum såsom högtider med mera). Ett program gick igenom Wikipediatexten och annoterade ut de korrekta entitetstaggarna runt de ord som hittades i nån av listorna, och detta utgjorde sedan vår tillgängliga trän-

ingsdata.
Träningsdata utgjordes alltså endast av annoterad faktatext, och om detta, även vid aktiv inlärning, påverkar det slutliga resultatet diskuteras i slutsatserna.

Använda attribut

De attribut som användes i systemet utgick från de olika grundtyper som togs upp i avsnitt 2.2.4. Systemet använde främst binära-, list- och lexikala attribut, men dessutom attribut som vilka sekvenser av ordklasser som orden före och efter ordet i sig hade.
En fullständig lista av alla attribut finns i Bilaga A: Systemets attribut.

Systemets entitetstaggar

De entitetstaggar, eller målklasser, som användes var de som tillhör standarden för de flesta namnigenkänningssystem. De entitetstaggar som användes var:

Person Detta är personnamn, vilket inkluderar förnamn, mellannamn och efternamn.

Plats Med denna tagg beskriv entiteter såsom länder och städer.

Organisation Detta är helt enkelt organisations- och företagsnamn.

Produkt Namn på produkter som köpes/säljes.

Händelse Namn på händelser såsom sportevenemang och liknande.

Datum Vanliga datum men i olika format.

Tidpunkter Detta inkluderar klockslag och benämningar för specifika klockslag.

Namngivna datum Detta är datum som beskrivs av namn, det vill säga högtider och liknande. Till exempel julafton, midsommar eller påskdagen.

Kvantiteter Detta inkluderar siffror som används för att beskriva mängder av något. Det kan vara mått eller rena antal.

Procenttal Beskriver procenttal som förekommer i texten, (Detta inkluderar inte bråk).

Monetära tal Beskriver monetära tal som förekommer i texten, vilket även inkluderar valutakoder.

Det kan även vara värt att nämna att dessa entiteter i praktiken utökas med fyra tillagda suffixtaggar. Detta behövs för att kunna beskriva entiteter som består av fler ord. De suffixtaggar som användes var:

start Läggs till om ordet i fråga är det första ordet för en entitet.

continue Läggs till om ordet i fråga befinner sig mitt i en entitet, det vill säga varken först eller sist.

end Läggs till om ordet i fråga är det sista ordet i entiteten.

unique Läggs till om ordet i fråga är det enda ordet som entiteten består av.

Alla entitetstaggar innehåller en suffixtagg tillagd som ändelse. Utöver dessa taggar finns det också en klass som kallas för *Annat* som beskriver alla de ord som inte tillhör en entitetsklass.

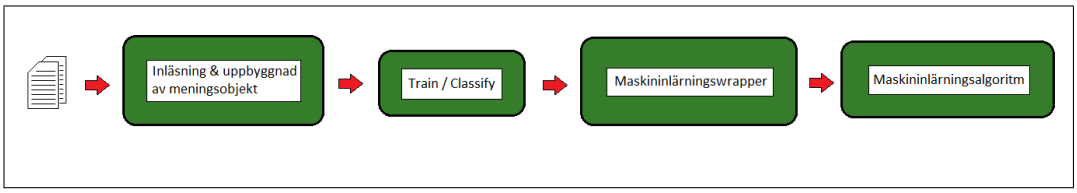
Val av maskininlärningsalgoritm

Maskininlärningsmetoden som användes är Logistisk Regression. Valet av maskininlärningsmetod påverkar till viss del vilken typ av attribut som ger bäst resultat, men eftersom jag i detta arbete inte hade som syfte att ta fram ett system som presterade så bra som möjligt, eller att jämföra olika typer av maskininlärningsmetoder mot varandra, var valet av algoritm inte avgörande. Dock väljs med fördel en algoritm som presterar bra för att studien skalla vara av värde. Som förklarades i avsnitt 2.2.3 ger metoden, för varje instans, en sannolikhetsfördelning över de olika entitetsklasserna.
Vi valde därför att använda oss av en avkodningsalgoritm för att hitta den mest troliga sekvensen av entitetsklasser för varje mening, givet sannolikhetsfördelningarna. Vi använde oss inte av en vanlig avkodningsalgoritm utan en annan metod i vilken avkodningsproblemet reduceras ner till ett grafproblem som sedan löses med Bellman-Fords grafalgoritm, Cormen et al. [4].
Bellman-Fords algoritm används för att hitta den kortaste vägen genom en riktad och viktad graf. Reduceringen till ett grafproblem sker genom att varje mening av taggad data skrivs om som en graf. För varje ord läggs lika många noder som det finns legala entitetsklasser, (inklusive suffix samt klassen för andra ord än entiteter), för ordet, in i grafen. Det läggs sedan till en kant från ordets motsvarande noder till de noder som tillhör det efterföljande ordet i meningen. Detta görs dock endast mellan de noder vilkas entitetsklasser kan efterfölja varandra. Till exempel läggs det inte till en kant från *Person_start* till *Person_unique*, eller från *Person_start* till *Location_continue* etcetera. Kostnaden för en kant ges av den logaritmerade sannolikheten för den nod som kanten går till, och mer specifikt negeras den för att få fram den slutliga kostnaden. Detta då vi med hjälp av Bellman-Ford hittar den kortaste vägen genom grafen, och ju större sannolikheter negerade, ju mindre kostnader.
Det läggs dessutom till en startnod och en slutnod till grafen. Startnoden har en kant till varje nod som tillhör det första ordet, med dessa noders sannolikheter som kostnader, och till sist läggs det dessutom, från alla noder som representerar det sista ordet i meningen, till en kant till slutnoden, vilka har kostnaden noll.
Det som sedan återstår är alltså att använda sig av Bellman-Fords algoritm för att hitta den kortaste vägen mellan startnoden och slutnoden, och därmed hitta den sekvens av efterföljande entitetstaggar som är den mest troliga.
På detta vis hittas en väg genom den riktade grafen, vilken har en så låg kostnad, och därmed hög sannolikhet, som möjligt, där varje nod som valts representerar en entitetstagg som valts för det motsvarande ordet i meningen. Resultatet blir att den sekvens av entitetstaggar som väljs blir den mest sannolika följden av taggar för den givna meningen.

3.1.1 Grundsystem för Named Entity Recognition

Det första systemet byggdes, som nämndes tidigare, tillsammans med Christopher Raschke. Detta system fungerar för vanlig NER-användning, det vill säga det tar emot en mängd träningsdata vilket sedan används i en maskininlärningsalgoritm för att träna fram en klassificerare. Klassificeraren kan sedan sparas i fil, för att kunna laddas igen och användas på icke annoterad text.

Systemet kan delas upp i fyra huvudkomponenter enligt Figur 3.1.



Figur 3.1: Systemets huvudkomponenter

Inläsning och uppbyggnad av meningsobjekt

I denna komponent läses ny text in, och den används vid både uppträning av en ny klassificerare och vid klassificering av ny text. Texten läses in en mening åt gången och det skapas ett meningsobjekt i systemet för meningen. När detta objekt skapas sparas inte bara orden i meningen i objektet, utan också ordens ordklasser och entitetstaggar (om sådana finns, det vill säga vid inläsning av redan annoterad text).

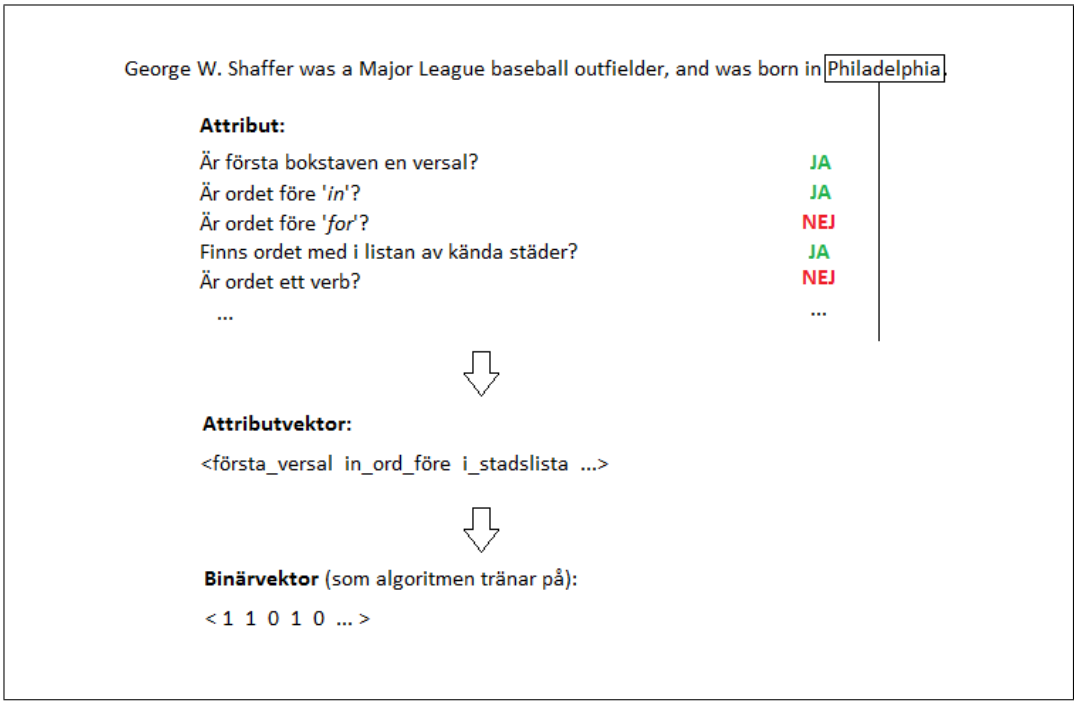
Att ta reda på vilken ordklass ett ord tillhör är ett helt eget problem i sig, vilket inte var aktuellt för denna rapport att lösa. Därför användes ett färdigt bibliotek för att hitta ordklasserna. Biblioteket som användes var OpenNLP (`http://incubator.apache.org/opennlp/`, [5]), vilket är ett Open Source-bibiliotek för Natural Language Processing.

Träning/Klassificering

Denna komponent tar emot meningar i form av de interna meningsobjekten. Hur meningsobjekten sedan behandlas beror på om de ska användas vid träning eller skall klassificeras.

Vid träning gås meningsobjekten igenom ett åt gången, och för alla ord som tillhör en entitetsklass läggs motsvarande attributvektor och entitetsklass till som ett exempel i mängden av träningsexempel. Hur processen att bygga upp en attributvektor för en träningsinstans går till, visas i Figur 3.2. Dessutom läggs attributvektorerna och motsvarande entitetsklass för ett mindre antal ord, som inte tillhör någon klass, till i träningsexempelmängden. Detta därför att systemet också skall lära sig vilka

typer av kontexter som inte tillhör någon entitetsklass. Det kan vara värt att notera att attributvektorerna byggs upp först i detta steg.
När alla exemplen har lagts till startas träningen, och när den avslutats sparas den upptränade klassificeraren ner som en fil.



Figur 3.2: Uppbyggnaden av en attributvektor

Vid klassificering laddas först en valfri klassificerare in från fil, och därefter tas texten som skall processeras in som meningsobjekt från komponenten innan i systemet. Alla meningsobjekt bearbetas och deras motsvarande attributvektorer byggs upp, för att sedan klassificeras av klassificeraren. När alla orden i meningen har klassificerats genomförs en avkodning på sannolikhetsfördelningarna för alla orden, för att hitta den mest troliga sekvensen av entitetsklasser för meningen. Till slut sparas den klassificerade texten ned i en ny fil.

Maskininlärningsalgoritm

Som nämndes innan användes multinomial logistisk regression som maskininlärningsalgoritm. Ett färdigt bibliotek med maskininlärningsalgoritmer, *Mallet*, användes istället för en egen implementering. Detta då syftet med arbetet inte var att utveckla en sådan algoritm, utan att undersöka aktiv inlärning inom NER. Mer information om Mallet finns att hitta på `http://mallet.cs.umass.edu/index.php`, [11].

Maskininlärningswrapper

Denna komponent hanterar kommunikationen mellan *Mallet* och det resterande systemet. Här instansieras maskininlärningsalgoritmen och både vid träning och klassificering tas träningsexempel och instanser som indata från komponenten innan, för att sedan göras om till det format som Mallet jobbar med, innan träningen eller klassificeringen utförs.
I denna komponent göms också eventuella inställningar för maskininlärningsalgoritmen i *Mallet*. Dessa är standardiserade för systemet och behövs alltså inte kunna ändras av de andra komponenterna.

3.1.2 System för Named Entity Recognition med Aktiv Inlärning

Detta system utgår från den tidigare versionen och innehåller exakt samma komponenter som det första systemet. Skillnaden ligger i hur träningskomponenten är uppbyggd, då detta system använder sig av aktiv inlärning istället för traditionell maskininlärning.

Den typ av aktiv inlärning som används i systemet är detsamma som nämndes i bakgrunden, närmare bestämt i avsnitt 2.3.1. Det är alltså en implementering av den allmänna algoritmen för aktiv inlärning.
Det behövs, som nämndes i bakgrunden, två mängder av träningsdata där den första (kallad mängd A härefter) utgör de exempel som en ny klassificerare tränas med i varje iteration. Den andra mängden (kallad mängd B härefter) klassificeras i varje iteration och de mest osäkra instanserna rättas av det mänskliga oraklet, och läggs till i A. Mer om hur de två mängderna av träningsdata togs fram beskrivs senare.

Antalet iterationer som körs ges av ett övre tak på antalet iterationer som man vill att systemet ska köra, eller tills då träningsmängden B inte innehåller några fler instanser. Inget stoppkriterium används i systemet då testerna som utförs alla körs ett givet antal iterationer. Detta då en jämförelse av hur de olika testernas prestationsmått förändras under samma antal iterationer skall kunna utföras.

Använt osäkerhetsmått

I varje iteration tränas alltså en klassificerare upp som sedan klassificerar alla icke annoterade instanser i B. Därefter tas de mest osäkra, (enligt ett givet mått), instanserna samt meningarna som de tillhör ut och presenteras för rättning.
Måttet som används för att beskriva hur osäker den tillfälliga klassificeraren är på en instans, bygger på sannolikhetsfördelningen över klasserna av entiteter som klassificeraren ger tillbaka. Mer specifikt beräknas hur nära denna sannolikhetsfördelning är till den likformiga sannolikhetsfördelningen över alla entitetsklasserna. Det är de m instanser som har kortast avstånd till den likformiga sannolikhetsfördelningen som presenteras, där m kan varieras.

Avståndet mellan de två sannolikhetsfördelningarna beräknas med hjälp av formeln för det euklidiska avståndet mellan två vektorer. Avståndet mellan vektorerna **p** och **q** ges av:

$$Dist = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + ... + (q_n - p_n)^2} \tag{3.1}$$

För detta system finns det elva olika entitetsklasser, och alla dessa har ett av fyra suffix, som förklarades tidigare i avsnitt 3.1. Utöver det finns fallet då ett ord inte tillhör en entitetsklass och då benämns med *Annat*-taggen. Detta betyder att systemet använder $(11 * 4) + 1 = 45$ möjliga taggar, och därmed 45 sannolikheter i de sannolikhetsfördelningar som klassificeraren ger vid en klassificering. Vektorerna med sannolikheter vars avstånd mellan varandra beräknas, innehåller alltså 45 värden, och den vektor som innehåller den uniforma sannolikhetsfördelningen har alltså $\frac{1}{45}$ som värde på varje position i vektorn.
När de m instanser som enligt ovanstående mått anses vara de mest osäkert klassificerade har hittats, presenteras de samt hela den mening som instansen tillhörde för den mänskliga annoteraren. Alla orden i meningarna rättas och förflyttas sedan från B till A, för att träna upp klassificeraren i nästa iteration.
På detta sätt förflyttas i varje iteration m meningar från B till A, som hela tiden växer. På så sätt utökas den tillgängliga träningsmängden och förhoppningsvis förbättras de tillfälliga klassificerarna för var iteration som går.

Träningsdata och indelning

Den träningsmängd som användes utgick från den egenannoterade faktatexten, och som grund valdes tusen slumpmässigt utvalda meningar ut. Av dessa bildade de hundra första meningarna träningsmängden A, och de resterande niohundra meningarna den andra mängden B.

3.2 Testöversikt

Det utfördes fem stycken tester totalt. Det första testet undersökte hur bra en traditionellt tränad klassificerare presterade på den givna testmängden. De följande tre testerna bestod av varianter av aktiv inlärning, där alla använde samma aktiva inlärningsmetod som beskrevs tidigare, men med olika parametrar. Avslutningvis utfördes ett test på den högst presterande, aktivt inlärda, klassificeraren, (från de tre föregående testen), på olika typer av text.

Testmängden

Den testmängd som användes för alla testerna består av ett antal texter av olika karaktär. Närmare bestämt utgörs testmängden av fem texter vilka är korta utdrag

ur olika romaner, fyra nyhetstexter och till sist en större text med faktakaraktär. Testmängden annoterades manuellt av oss (Johan Wessman & Christopher Raschke).

3.2.1 Test av traditionellt tränad klassificerare

För detta test användes det gemensamt byggda systemet som beskrevs i avsnitt 3.1 och avsnitt 3.1.1. Systemet använder sig av all den tillgängliga träningsdatan, och alla ord som är utmärkta med en entitetstagg används som träningsinstanser. Dessutom läggs också ett antal slumpmässigt utvalda ord som inte tillhör en entitetsklass till som träningsinstanser.
En klassificerare, som baseras på ME som algoritm, tränades med hjälpa av *Mallet* upp på alla träningsexemplen. För att utvärdera klassificeraren användes den ovan nämnda testmängden, och klassificeraren applicerades på ett ord åt gången. Den svarade med sannolikhetsfördelningen över entitetsklasserna för det aktuella ordet, och för att få ut den slutliga entitetsklassen användes avkodningsalgoritmen som beskrivs i avsnitt 3.1. Därefter jämfördes de klassificerade entitetsklasserna med de manuellt annoterade över hela testmängden, och de statistiska prestationsmåtten, (som nämndes i bakgrunden), *Precision*, *Recall* och *F-measure* beräknades.

3.2.2 Tester av aktivt inlärd klassificerare

Denna sektion beskriver tre tester med endast en minimal, men viktig, variation. Dessa utfördes på det aktivt inlärda systemet som beskrevs i avsnitt 3.1.2, och samma testmängd som vid föregående test användes för att utvärdera klassificerarna. För träningsprocessen användes träningsmängden som också beskrivs i avsnitt 3.1.2, det vill säga uppdelad i mängderna A och B.
Skillnaden mellan testerna bestod i träningsutförandet. Som nämndes innan användes inte något stoppkriterium, men dock ett tak på maximalt antal iterationer som träningen kördes, satt till 40.
I varje iteration tränades en klassificerare upp på träningsexemplen i A och denna klassificerare klassificerade mängden B, varefter de m mest osäkra instansernas meningar sparades undan för rättning. Det var m som varierade mellan de olika testerna och antog värdena 5, 10 respektive 15.

Dessa rättades manuellt, och förflyttades från B till A. För att beskriva hur de upptränade klassificerarna förändrades mellan iterationerna applicerades också klassificeraren i varje iteration på testmängden. Efter att avkodningsalgoritmen hittade den mest troliga sekvensen av entitetsklasser för meningarna i testmängden, jämfördes de med de manuellt annoterade svaren och *Precision*, *Recall* samt *F-measure* beräknades.
Förutom den historik över hur *Precision*, *Recall* och *F-measure* förändras över iterationerna, sparades också för varje test den klassificerare som presterade bäst över testmängden. Det sista testet baserades på den bäst presterande av dessa tre sparade klassificerare.

3.2.3 Test på olika typ av text

Som nämndes tidigare består testmängden av tre olika typer av texter, nämligen romantext, nyhetstext och faktatext. I det sista testet användes den klassificerare som presterade bäst över hela testmängden, för att klassificiera de separata delmängderna av testdata som innehåller endast romantext, nyhetstext respektive faktatext.

Klassificeraren applicerades på en delmängd åt gången och avkodningsalgoritmen hittade, precis som tidigare, den mest sannolika sekvensen av entitetstaggar för meningarna i delmängden. Därefter jämfördes dessa taggar med de manuellt annoterade och *Precision*, *Recall* och *F-measure* beräknades.

3.3 Resultat

Resultaten av de olika testerna delas, som i föregående avsnitt, in i tre delar. Den första delen innehåller resultaten för den traditionellt upptränade klassificeraren över testmängden. Den andra delen innehåller resultaten för de klassificerare som tränades upp med aktiv inlärning, och till dessa resultat hör hur deras prestationsmått över testmängden förändras, och en graf över denna förändring. Den sista delen innehåller resultaten som den bästa klassificeraren, som tränades upp med aktiv inlärning, fick på de olika typerna av text som ingick i testmängden.

3.3.1 Traditionellt tränad klassificerare

Den traditionellt tränade klassificeraren tränades upp på all tillgänglig träningsdata. Resultatet för den syns i Tabell 3.1, och det säger att denna typ av klassificerare har bättre *Recall* än *Precision*, på testmängden.

Andel %	Precision	Recall	F-measure
Traditionellt tränad	43,1	60,3	50,3
5 rättade / iteration	68,1	35,1	46,3
10 rättade / iteration	58,1	50,9	54,3
15 rättade / iteration	60,0	54,1	56,8

Tabell 3.1. Prestationsmått för de olika klassificerarna

3.3.2 Klassificerare - aktiv inlärning

Tre tester utfördes under denna del, alla med aktiv inlärning som uppträningsmetod. Det som varierade mellan dem var antalet meningar som rättades mellan varje iteration, och detta antal var först 5 stycken meningar, sedan 10 och till sist 15 stycken.

I det första av dessa tre test rättades 5 meningar per iteration. Hur de upptränade klassificerarnas prestationsmått över testmängden ändrades under iterationerna ges av Tabell 3.2, och hur förändringen ändras grafiskt kan ses i Figur 3.3. Värdena förändras långsamt under iterationerna, men *Recall* stiger sakta men säkert, medan *Precision* oscillerar mellan olika värden, och till slut håller sig mellan 60-70%. Detta betyder att det ihopviktade värdet, *F-measure*, stiger med ungefär samma takt som *Recall*. Eftersom så pass lite ny rättad data introduceras till träningsmängden i varje iteration stiger måtten långsamt, och en prestationstopp nås inte förren de 40 iterationerna tagit slut. Då antar måtten värdena i Tabell 3.1. Dock är det möjligt att prestationsmåtten skulle fortsatt att öka om fler iterationer körts.

I det andra testet rättades 10 meningar per iteration. Hur prestationsmåtten över testmängden ändrades för de olika klassificerarna ges av Tabell 3.3, och av Figur 3.4. I detta fall stiger måtten i snabbare takt än i testfallet innan, men takten med vilken de ökar avtar efter 30 iterationer ungefär. Både *Recall* och *F-measure* stiger i jämn takt utan stora förändringar medan *Precision* i detta test också har ett oscillerande beteende. Det är mer meningar som rättas per iteration, och därmed mer tillgängliga träningsexempel i varje iteration, vilket leder till att en prestationstopp nås innan de 40 iterationerna har körts klart. Detta händer vid iteration 39 och måtten har då värdena i Tabell 3.1. Dock är detta endast det lokala maximat för de 40 iterationerna.

För det tredje testet ökades antal meningar som rättades i varje iteration till 15. Prestationsmåttens förändringar över de 40 iterationerna finns i Tabell 3.4 och i Figur 3.5. Fler meningar rättas mellan varje iteration än i någon av de två tidigare testerna, vilket leder till att *Recall* och *F-measure* stiger i en högre takt än tidigare. Dock växer de fortfarande med jämna steg. Det oscillerande beteendet för *Precision* återfinns endast till en början varefter det konvergerar till ett intervall mellan 58-60%. Värdena för det lokala maximat i detta test ges i Tabell 3.1. Det inträffar vid iteration 36 och beror till stor del på att mängden tillgängliga träningsexempel ökar i större takt än tidigare.

Som kan ses i resultaten för fallen med fem och femton rättningar per iteration börjar *Precision* på en väldigt hög nivå och minskar därefter. Detta beror på att klassificeraren till en början ej har lärt sig särskilt många exempel, vilket får effekten att den inte klassificerar speciellt många ord, men de som väl klassificeras är den väldigt träffsäker på. Detta stöds dessutom av den låga nivån av *Recall*.
Att detta inte inträffar för fallet med tio rättningar per iteration kan förklaras med att de exempel som klassificeraren ursprungligen tränas upp på, innan den första iterationen, väljs ut slumpmässigt, och att de skiljer sig tillräckligt mycket åt för att *Precision* skall börja på en lägre nivå.

Iteration	Precision	Recall	F-measure
1	29,6	3,0	5,5
2	93,8	2,8	5,4
3	84,2	3,0	5,8
4	75,0	2,8	5,4
5	71,0	4,1	7,8
6	73,5	4,8	8,8
7	60,0	3,9	7,4
8	62,2	4,3	8,1
9	58,1	4,7	8,7
10	61,4	5,1	9,4
11	59,2	6,1	11,0
12	50,9	5,5	10,0
13	63,1	7,8	13,9
14	64,2	8,2	14,5
15	70,1	10,2	17,9
16	64,4	11,1	19,0
17	62,6	13,0	21,5
18	59,1	12,6	20,8
19	62,5	14,5	23,6
20	66,1	15,3	24,9
21	63,6	15,8	25,3
22	67,4	17,4	27,7
23	64,0	16,9	26,7
24	61,9	17,8	27,7
25	64,5	19,3	29,7
26	66,7	20,8	31,7
27	65,7	21,7	32,6
28	70,6	22,4	34,0
29	65,6	19,4	30,0
30	63,4	23,1	33,8
31	63,0	22,9	33,6
32	69,3	24,5	36,2
33	65,3	24,8	36,0
34	67,0	25,9	37,4
35	67,3	27,3	38,9
36	65,6	27,8	39,1
37	67,3	30,4	41,9
38	66,5	31,8	43,1
39	68,6	33,0	44,6
40	68,1	35,1	46,3

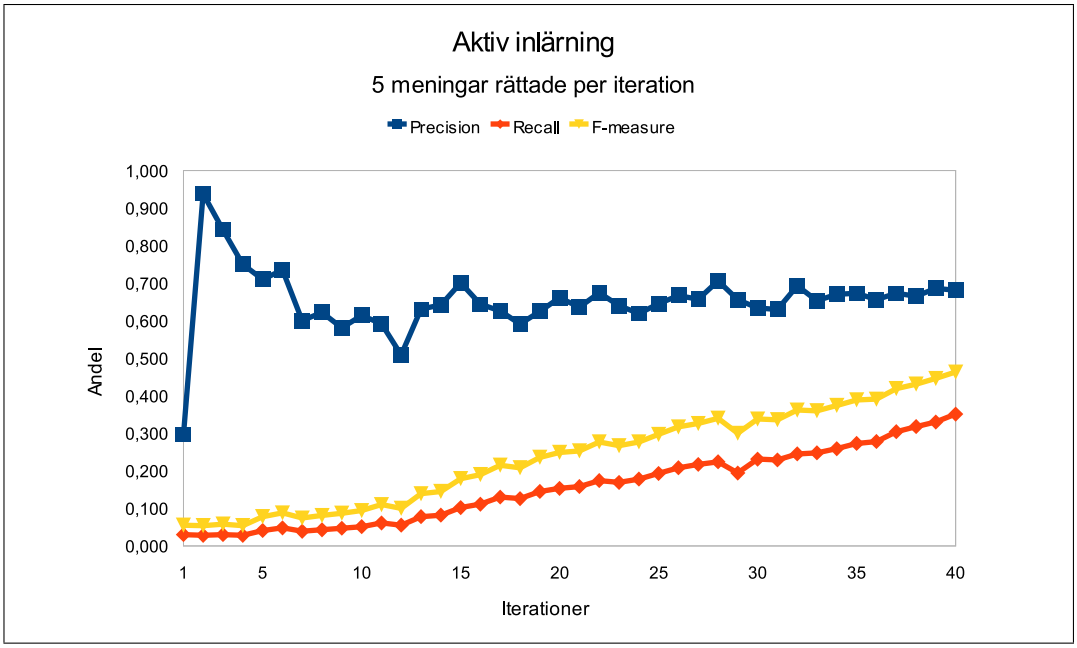
Tabell 3.2. Prestationsmått för klassificerare som tränats upp med hjälp av aktiv inlärning. Fem meningar rättades i varje iteration, och värdena ges i %

Iteration	Precision	Recall	F-measure
1	30,1	3,6	6,4
2	29,2	3,6	6,4
3	30,4	4,0	7,0
4	35,1	4,9	8,6
5	38,0	5,7	9,9
6	41,3	6,3	10,9
7	43,8	6,6	11,5
8	45,7	9,2	15,4
9	48,1	9,9	16,5
10	53,1	11,5	18,9
11	51,2	12,4	19,9
12	48,0	11,6	18,7
13	47,5	11,2	18,1
14	45,7	11,5	18,3
15	47,9	13,5	21,0
16	48,7	14,3	22,1
17	52,9	15,9	24,4
18	52,9	18,1	27,0
19	49,5	18,5	26,9
20	50,3	20,2	28,9
21	51,9	22,7	31,6
22	48,9	21,9	30,2
23	51,1	24,6	33,2
24	54,8	29,6	38,4
25	55,7	32,1	40,7
26	54,8	32,3	40,6
27	55,8	35,4	43,3
28	56,1	36,4	44,2
29	56,9	39,2	46,4
30	56,9	41,2	47,8
31	56,7	42,8	48,8
32	55,4	43,3	48,6
33	54,2	44,8	49,0
34	54,1	45,9	49,7
35	53,9	47,4	50,5
36	54,8	49,0	51,8
37	55,0	49,1	51,9
38	55,0	50,1	52,4
39	58,1	50,9	54,3
40	56,4	50,1	53,1

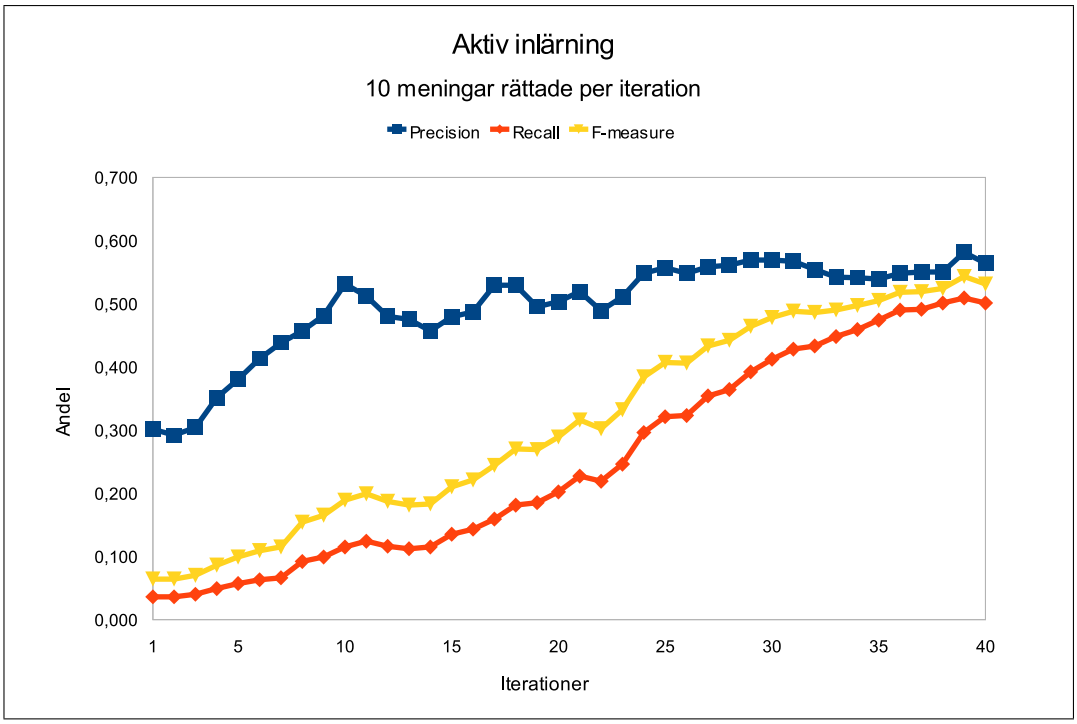
Tabell 3.3. Prestationsmått för klassificerare som tränats upp med hjälp av aktiv inlärning. Tio meningar rättades i varje iteration, och värdena ges i %

Iteration	Precision	Recall	F-measure
1	30,0	1,1	2,2
2	65,5	6,8	12,3
3	76,3	11,0	19,2
4	71,7	13,6	22,9
5	71,6	15,0	24,8
6	68,7	13,2	22,1
7	66,4	15,9	25,6
8	66,0	20,1	30,8
9	60,6	20,9	31,1
10	66,8	25,3	36,8
11	68,1	25,6	37,2
12	68,1	27,7	39,4
13	61,6	27,6	38,0
14	63,2	30,5	41,2
15	64,7	31,1	42,0
16	63,3	35,3	45,3
17	64,9	35,7	46,0
18	63,7	37,6	47,3
19	62,0	40,0	48,4
20	61,8	41,3	49,5
21	64,5	41,7	50,7
22	60,8	42,8	50,2
23	60,1	42,1	49,5
24	60,5	41,7	49,4
25	58,2	43,0	49,4
26	59,0	44,9	51,0
27	58,9	45,9	51,6
28	57,8	48,5	52,7
29	57,8	46,0	51,2
30	58,7	47,5	52,6
31	58,8	48,9	53,4
32	59,4	50,3	54,5
33	58,0	50,8	54,2
34	57,8	51,9	54,7
35	58,1	52,0	54,9
36	60,0	54,1	56,8
37	59,0	54,3	56,6
38	57,7	54,2	55,9
39	56,8	54,2	55,5
40	57,0	53,8	55,3

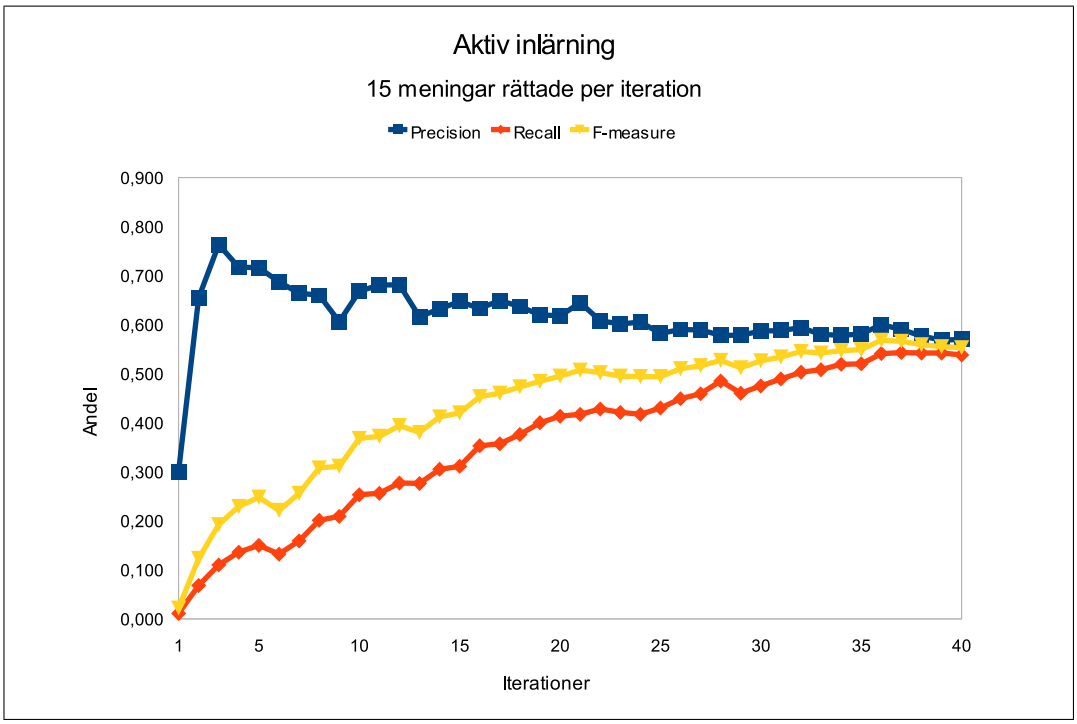
Tabell 3.4. Prestationsmått för klassificerare som tränats upp med hjälp av aktiv inlärning. Femton meningar rättades i varje iteration, och värdena ges i %



Figur 3.3: Förändring mellan iterationer för klassificerare med aktiv inlärning.



Figur 3.4: Förändring mellan iterationer för klassificerare med aktiv inlärning.



Figur 3.5: Förändring mellan iterationer för klassificerare med aktiv inlärning.

3.3.3 Olika typer av text

Det avslutande testet berodde på de tidigare tre testerna av aktiv inlärning. För varje test sparades den klassificerare vars prestationsmått var högst på testmängden, vilket kunde ses som den lokalt bästa klassificeraren. Av de tre som sparades undan användes den bäst presterande klassificeraren på de olika typerna av text som testmängden bestod av. Den bäst presterande var den från uppträningsprocessen där 15 meningar rättades i varje iteration.
Denna klassificerare användes för att klassificera de tre delmängderna av den totala testmängden vilka bestod av romantext, nyhetstext respektive faktatext. Resultatet av dessa testkörningar ges i Figur 3.6. Som man kan se presterade klassificeraren bäst på ren faktatext och en aning sämre på nyhetstext. Dock var resultatet betydligt sämre för text tagen från romantext.

Texttyp	Precision	Recall	F-measure
Romantext	43,1	48,1	45,5
Nyhetstext	60,4	56,8	58,5
Faktatext	73,9	55,0	63,1

Tabell 3.5. Den bästa klassificerarens resultat på de olika typerna av text. Värdena ges i %.

Kapitel 4

Slutsatser

I denna rapport har användandet av aktiv inlärning som uppträningsmetod inom NER undersökts. Det var tre frågor som ämnades bli besvarade. Den första var om ett system som tränats upp med hjälp av aktiv inlärning kan prestera på samma nivå som ett traditionellt tränat system. Hur påverkas resultatet vid aktiv inlärning om antalet instanser som annoteras i varje iteration varierar var den andra huvudfrågan. Och till sist utreddes om en klassificerare uppträned med hjälp av aktiv inlärning, precis som en traditionellt tränad, fortfarande presterar bättre på den typ av text som den tränats på jämfört med annan typ av text.
Slutsatserna som kan dras från de erhållna resultaten svarar förhoppningsvis på dessa frågeställningar och kan delas in i ett par delar.

4.1 Aktiv inlärning och traditionell inlärning presterar likvärdigt

Klassificerare som har tränats upp med aktiv inlärning kan ge resultat som är jämförbara med en traditionellt tränad klassificerares resultat. Enligt resultaten från avsnitt 3.3.1 och avsnitt 3.3.2 håller nästan alla de tre bästa aktivt inlärda klassificerarna samma nivå, eller bättre, som det traditionellt tränade systemet.
Om för få rättningar görs per iteration kommer den resulterande klassificeraren inte upp till samma nivå som det traditionella systemet. Detta beror på att den bredd av exempel som den totala träningsmängden består av inte kan beskrivas med endast den data som finns tillgänglig då fem meningar rättas per iteration. Detta resonemang förstärks vidare av att det traditionellt tränade systemet har högre *Recall* än vad systemet med fem rättningar per iteration har, och tvärtemot vad gäller *Precision*. Det aktivt inlärda systemet håller en högre *Precision* då den har färre exempel att tillgodose, och därmed till större del kan lära sig att särskilja dessa exempel. Detsamma gäller om man jämför den traditionellt tränade med den bästa klassificeraren som har uppdaterats med tio rättningar per iteration.
Att det traditionella systemet har en högre *Recall* tyder på att det har tränats på tillräckligt många exempel för att ha lärt sig många av de möjliga specialfall där

entiteter kan förekomma i en text. Dock håller dess *Precision* en lägre nivå vilket tyder på att mängden exempel som den tränats på har fått effekten att den inte kan särskilja mellan entiteterna på samma sätt, utan istället har lärt sig i vilka kontexter en entitet i allmänhet existerar.

Det är först när den aktivt inlärda klassificeraren uppdateras med femton rättade meningar per iteration som den presterar bättre än det traditionella systemet. Den får se tillräckligt många träningsexempel att *Recall* höjs till nästan samma nivå som det traditionella systemet, men fortfarande på en sådan nivå att den kan särskilja de olika entitetsklasserna åt i större utsträckning än det traditionella systemet.

4.2 Fler rättningar per iteration ger bättre resultat

Frågan: ”Hur många rättningar som ska göras i varje iteration när man använder aktiv inlärning” verkar ha svaret; fler är bättre. I alla fall om man tittar på de tre fall som undersöktes i denna rapport. Ju fler meningar som rättas i varje iteration, desto snabbare växer prestationsmåtten över hela testmängden.
Om man jämför mellan att uppdatera med fem respektive femton rättade meningar per iteration, ser man att systemet med fem rättningar når en *F-measure* på 30% först vid iteration 26, medan systemet med femton rättningar når denna nivå på *F-measure* redan efter iteration 8. Att samma *F-measure* nås för något mindre antal rättade exempel beror troligtvis på att den ursprungliga mängden träningsexempel som klassificeraren började med valdes ut slumpmässigt, vilket ger olika startpunkter i de olika fallen.
Om måtten växer snabbare skulle antalet iterationer som behövs kunna sänkas, och då samtidigt dra ned den tidsåtgång som krävs för att träna upp systemet. Värdena efter iteration 30, då systemet uppdateras med femton rättade meningar per iteration, visar på endast en mindre förändring i *F-measure* och det framförallt efter iteration 35. Man skulle därför kunna ställa sig frågan om de få extra procenten som fås efter flera iterationer är värt den tid som måste avsättas för att utföra dessa rättningar.

Dock finns det nackdelar med att rätta för många meningar per iteration. Om för många rättningar måste utföras i varje iteration kommer tidsåtgången för träningsprocessen att höjas och mer tid måste spenderas av en mänsklig annoterare för att rätta systemet under träningen, vilket kan visa sig vara kostsamt.
Om för mycket ny träningsdata introduceras mellan iterationer finns möjligheten att man missar det globala maximat för klassificerarens prestanda. Till slut får man samma beteende som för det traditionellt tränade systemet där *Recall* växer medan *Precision* försämras. Denna typ av problem kan dock undvikas om man inför ett stoppkriterium som stannar träningsprocessen när oscillerande beteende eller en sänkning i prestationsmåtten upptäcks.

4.3 Typen av text vid träning är viktigt

Vid traditionell träning är det känt att den typ av text som används som träningsdata till stor del påverkar hur bra klassificeraren presterar på ny text. Är det av samma typ fås bättre resultat än om den nya texten är av helt annan art.
Resultaten i avsnitt 3.3.3 säger att detta också gäller vid aktiv inlärning. Träningsmängden som användes för rapporten består i grunden av annoterad text från Wikipedia, vilket betyder att det är faktatext som använts. Det är också faktatexten som systemet presterar bäst på enligt resultaten och nästan likvärdiga resultat fås på nyhetstexten. Detta beror troligtvis på att nyhetstext är relativt lik faktatext till strukturen. Nyhetstext kan ses som en form utav faktatext.
Klassificeraren presterar sämre på romantexter vilket är förståeligt då de skiljer sig från faktatext i uppbyggnad och i vilka typer av ord som används.

4.4 Eventuell osäkerhet i resultaten

Eftersom de olika testerna endast utfördes med parametervärdena en gång uppstår en fråga i hur pass statistiskt säkra resultaten är. Hur de ursprungliga träningsexemplena valdes ut påverkar också resultatet till viss del. Hade en vida använd träningsmängd använts hade resultaten varit mer jämförbara med andra resultat inom området.
De slutsatser som drogs i denna rapport bör därför ses som preliminära slutsatser, som är troliga, men inte helt säkra.
För att kunna ta fram resultat och basera slutsatser på dem som är helt statistiskt säkra, bör testerna utföras ett antal gånger för att erhålla ett medelvärde, vilket är ett mer säkert resultat.

Kapitel 5

Framtida arbete

I denna rapport undersöktes hur bra aktiv inlärning står sig i jämförelse med traditionell inlärning. Ett framtida arbete skulle kunna fokusera på att använda sig av en annan typ av metod för aktiv inlärning, till exempel en kommitté av klassificerare som nämndes i avsnitt 2.3.1. Det skulle dessutom vara intressant och värt fortsatt arbete att undersöka hur ett mer raffinerat osäkerhetsmått påverkar resultaten. Som ett exempel skulle man kunna ta i beaktande att alla de instanser som väljs ut för rättning alla signifikant skiljer sig åt och att de valda instanserna representerar hela mängden av tillgänglig icke annoterad data. Man skulle också kunna prioritera de instanser som särskiljer sig mest från de som redan finns i träningsmängden.

Det skulle också vara intressant att titta på avbrottskriterier och implementera ett sådant. Istället för att hålla sig till ett fast antal iterationer kan man stanna träningsprocessen när prestationsmåtten avtar eller när de skiftar upp och ner inom ett visst intervall.

Ett annat intressant område att undersöka är valet av attribut. Istället för att alltid använda alla attribut som man har tillgängliga kan man först göra en utredning för att hitta de attribut som är mest relevanta för den typ av text som systemet skall användas på. Man går alltså från en större attributmängd till en mindre bestående av attribut som ger tillräcklig information. På så sätt förminskas komplexiteten i systemet och de attribut som inte ger tillräcklig information för en viss typ av text används inte alls.

Litteraturförteckning

[1] Oliver Bender, Franz Josef Och, and Hermann Ney. Maximum Entropy Models for Named Entity Recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4*, pages 148–151, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1119176.1119196. URL `http://dx.doi.org/10.3115/1119176.1119196`.

[2] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. NYU:Description of the MENE Named Entity System as Used in MUC-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7*, New York, NY, USA, 1998. Url (Sep 2010): `http://www.dfki.de/~neumann/esslli04/reader/NE/borthwick98nyu.pdf`.

[3] Andrew Eliot Borthwick. *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, New York University, New York, NY, USA, 1999. Url (Sep 2010): `http://portal.acm.org/citation.cfm?id=930095&coll=Portal&dl=GUIDE&CFID=104776302&CFTOKEN=48627610`.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7.

[5] The Apache Software Foundation. OpenNLP, Natural Language Processing. Url (April 2011): `http://incubator.apache.org/opennlp/`.

[6] Simon Haykin. *Neural Networks, A Comprehensive Foundation, Second Edition*. Prentice Hall, Inc, 1999. ISBN 0-13-273350-1.

[7] Wikimedia Foundation Inc. Wikimedia Dump Service. Url (April 2011): `http://dumps.wikimedia.org/backup-index.html`.

[8] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education Inc, 2008. ISBN 978-0-13-504196-3.

[9] Katharina Kaiser and Silvia Miksch. Information Extraction, A Survey, 2005. Url (Sep 2010): `http://ieg.ifs.tuwien.ac.at/techreports/Asgaard-TR-2005-6.pdf`.

[10] Florian Laws and Hinrich Schätze. Stopping Criteria for Active Learning of Named Entity Recognition. In *COLING '08: Proceedings of the 22nd International Conference on Computational Linguistics*, pages 465–472, Morristown, NJ, USA, 2008. Association for Computational Linguistics. ISBN 978-1-905593-44-6. Url (Sep 2010): `http://www.aclweb.org/anthology/C/C08/C08-1059.pdf`.

[11] Andrew Kachites McCallum. Mallet, MAchine Learning for LanguagE Toolkit, 2002. Url (April 2011): `http://mallet.cs.umass.edu/index.php`.

[12] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Book Co, 1997. ISBN 0-07-115467-1.

[13] Fredrik Olsson. *Bootstrapping Named Entity Annotation by Means of Active Machine Learning*. PhD thesis, Göteborgs Universitet, 2008. Url (Sep 2010): `http://gupea.ub.gu.se/dspace/bitstream/2077/18722/2/gupea_2077_18722_2.pdf`.

[14] Dan Shen, Jie Zhang, Jian Su, Guodong Zhou, and Chew-Lim Tan. Multi-criteria-based Active Learning for Named Entity Recognition. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 589, Morristown, NJ, USA, 2004. Association for Computational Linguistics. Url (Sep 2010): `http://acl.ldc.upenn.edu/P/P04/P04-1075.pdf`.

[15] Andreas Vlachos. A Stopping Criterion for Active Learning. *Comput. Speech Lang.*, 22(3):295–312, 2008. ISSN 0885-2308. Url (Sep 2010): `http://www.cl.cam.ac.uk/~av308/vlachos_active_csl07.pdf`.

Bilaga A

Systemens attribut

Detta är de attribut som användes i systemet, och de är grupperade i fyra grupper.

A.1 Ordklassattribut

Dessa attribut aktiveras på alla de former av ordklasser som förekommer runt omkring ordet i fråga. Området är fem ord framför samt fem ord bakom, inom samma mening. Alla förekommande kombinationer av ordklasser aktiveras som ett separat attribut.

A.2 Listattribut

Dessa attribut aktiveras på om ordet självt eller upp till fem ord framför eller bakom, i samma mening, hittas i någon av de olika listorna med entiteter. Till exempel ett förnamn följt av ett efternamn i en lista med personnamn.
De listor som användes var över för- och efternamn för personer, kända personnamn, städer, länder, företag, universitet, datum, dagar, månader och en lista med några allmänna klockslag.

A.3 Meningsattribut

Dessa attribut är av binär karaktär och beskriver relationen mellan ordet och meningen där det förekommer.

- Är ordet det första ordet i meningen?
- Är ordet det sista ordet i meningen?
- Är meningen endast ett ord lång?
- Är meningen under fem ord lång?

- Är meningen över tjugo ord lång?
- Innehåller ordet före en siffra?
- Består ordet före av endast siffror?
- Är ordet före *in* eller *at*?
- Är ordet före *says*?
- Är ordet före *tells*?
- Är ordet före *writes*?
- Är ordet före *on*?
- Är ordet före *is*?
- Är ordet före *married*?
- Är ordet före *a*?
- Är de tre orden före *what*, *is* och *now*?

A.4 Ordattribut

Dessa attribut är av binär karaktär och beskriver hur ordet i fråga ser ut.

- Är första bokstaven i ordet en versal?
- Finns det någon versal i ordet?
- Består ordet av endast versaler?
- Innehåller ordet någon siffra?
- Består ordet av endast siffror?
- Innehåller ordet en punkt?
- Innehåller ordet ett kolon eller ett semikolon?
- Innehåller ordet en parentes?
- Innehåller ordet ett utropstecken eller frågetecken?
- Innehåller ordet en framåt- eller bakåtsnedsträck?
- Är ordets längd ett?
- Är ordets längd två?

- Är ordets längd tre?
- Är ordets längd fyra?
- Är ordets längd mellan fem och sju?
- Är ordets längd åtta eller mer?
- Är sista bokstaven i ordet ett *s*?
- Innehåller ordet endast siffror och är två karaktärer långt?
- Innehåller ordet endast siffror och är fyra karaktärer långt?

TRITA-CSC-E 2011:118
ISRN-KTH/CSC/E--11/118-SE
ISSN-1653-5715

www.kth.se