Alex Lövgren | 010719-4896 | alexlov@kth.se
Jacob Gustavsson | 000506-6972 | jacgu@kth.se

2022-12-18
DD1418

Projektrapport

En jämförelse av en maskininlärningsmodell baserad på ordvektorer och en regelbaserad modell i området klassificering av numeriska uttryck

Innehållsförteckning

Introduktion

Named Entity Recognition (NER) kallas den uppgift som går ut på att hämta ostrukturerad data från en text för att sedan klassificera denna enligt angivna fördefinierade klasser, som exempelvis årtal eller namn på personer.[1] Genom denna process kan enorma mängder data med hjälp av olika modeller avväpnas och göras mer lättförståelig. NER-modeller användes till en början främst för extrahering av nyckelinformation i journalistiska och militära syften, men används idag även i flera delar av företags verksamheter, och kommer förmodligen att där få en allt större roll exempelvis inom kundtjänst och MarTech.[2]

I den här rapporten kommer det beskrivas hur två olika metoder kan användas för att lösa en lite mer nischad uppgift inom NER, nämligen klassificering av numeriska uttryck. Till en början tränades och utvärderades en modell baserad på maskininlärning och random indexing, och därefter skapades en enklare modell baserad på reguljära uttryck och det ord som följde det numeriska uttrycket. Dessa två metoder kördes sedan på samma testdata för att jämföra vilken av de två som bäst utförde uppgiften att klassificera numeriska uttryck i en löpande text, och resultatet från detta presenteras i slutskedet av rapporten.

Bakgrund

Det finns flera metoder för NER klassificering, vissa är baserade på maskininlärning och andra inte. Det finns också hybridversioner som försöker kombinera de olika fördelarna av maskininlärning och regelbaserad NER.

Maskininlärning är ett vanligt sätt att implementera NER klassificering. Det är en flexibel metod eftersom att den kan anpassa sig till ny data och relativt enkelt kan utökas till att klassificera nya klasser. Men det krävs också stora datamängder för att maskininlärning ska fungera väl, i princip gäller att desto mer data en modell får tränas på, ju bättre kommer den att bli. Men det krävs också att det finns en bra spridning i datamängden vid klassificering. Om en modell tränas upp mer på vissa klasser än andra så blir den också bättre på att klassificera vissa typer av klasser jämfört andra. Datamängden är avgörande, exempelvis kan en begränsad datamängd innebära att en modell inte kan klassificera för många klasser eftersom att den inte kommer tränas upp tillräckligt på någon av dem.

Ett sätt att implementera Named Entity Recognition i maskininlärning är med hjälp av Random Indexing. Random Indexing är en metod som används för att ord ska kunna beskrivas i vektorform.[3] Metoden går ut på att varje ord i ett korpus först tilldelas en unik slumpgenererad vektor, en så kallad random vektor. Random vektorerna kan sedan summeras för olika datamängder. Ett exempel på implementation är att ett ords kontext kan beskrivas genom att de omkringliggande ordens random

[1] Wikipedia,*Named-entity recognition,*2022, https://en.wikipedia.org/wiki/Named-entity_recognition [hämtad 2022-12-15]
[2] Feedstock, *Unlocking business value from Named Entity Recognition*, u.å, https://www.feedstock.com/quick-reads/unlocking-business-value-from-named-entity-recognition-ner/ [hämtad 2022-12-15]
[3] Johan Boye, *F9 - Lexical Semantics,* 2022, https://www.csc.kth.se/~jboye/teaching/language_engineering/9_word_embeddings.pdf [hämtad 2022-12-15]

vektorer summeras. Den nya vektorn, en så kallad kontext vektor, kommer då innehålla information om kontexten. Ord som har liknande omkringliggande ord kommer att ha liknande kontext vektorer, detta gör det lätt att jämföra vilka ord som förekommer i liknande sammanhang och vilka som inte gör det. Random vektorer kan på samma sätt användas för att jämföra likheter mellan olika dokument. Dokument som har liknande förekomst av vissa ord kommer då att ha "dokumentvektorer" som är lika varandra. Skillnaden blir också desto större ju färre likadana ord olika dokument har. På detta sätt kan dokument sorteras eller klassificeras. Det finns flera olika metoder för att jämföra vektorers likhet. Vissa metoder tar hänsyn till antalet gånger olika ord förekommer i en text, andra tar endast hänsyn till vilka ord som förekommer.

Regelbaserade NER metoder har fastställda regler och villkor för att klassificera olika entiteter. Fördelar med regelbaserade klassificeringsmetoder är att de ofta är enklare att implementera och är enklare att förstå jämfört med maskininlärningsbaserade metoder. Att klassificera standardiserade uttryck passar en regelbaserad NER bra, till exempel klockslag på formatet "15:00" eller datum på formatet 24/12-2022. Detta på grund av att de innehåller tecken, eller en syntax, för hur de ska se ut. Nackdelarna med regelbaserad NER ligger samtidigt också i att den är statisk. Om det som ska klassificeras inte kan urskiljas genom att analysera uttrycket och/eller om många klasser ska klassificeras så blir det snabbt ineffektivt och svårt att formulera regler som ska fånga alla dessa klasser. Regelbaserad klassificering kan inte heller anpassa sig till nya data eller förändringar i mönstren i data på ett självständigt sätt, vilket en maskininlärningsbaserad klassificering kan.

För att evaluera klassificeringsmetoder så kan de predicerade och korrekta klasserna beskrivas i en så kallad Confusion Matrix.[4] En sådan kan ge en bra representation över hur bra en modell är på att predicera olika klasser och på vilka sätt den kanske brister. Precision, recall och accuracy är mått som kan användas för att evaluera och jämföra olika modeller. Precision är ett mått på hur många prediktioner av en klass som är korrekta. Recall är ett mått på hur många datapunkter av en viss klass som klassificeras rätt. Accuracy är ett övergripande mått som anger hur stor andel av alla testdatapunkter som klassificerats rätt.

Hypotes

En analys av det egna beteendet rörande klassificering av numeriska uttryck visar att vi i hög grad använder den kontext som finns runt uttrycket för att avgöra vad det representerar.
I de allra flesta fallen säger det numeriska uttrycket i sig inte alls mycket, det är nästan enbart då det är en tid eller ett datum i fråga. Utifrån dessa observationer gick vi vidare med hypotesen att av de två modeller vi skulle konstruera så skulle den som enbart baserades på kontexten vara den som predicerade numeriska uttryck bäst.

[4] Johan Boye, *F7 - Text classification,* 2022, https://www.csc.kth.se/~jboye/teaching/language_engineering/7_text_classification.pdf [hämtad 2022-12-17]

Beskrivning av implementation

I den inledande fasen av projektet var det nödvändigt att begränsa omfattningen av undersökningen genom att välja ut de olika numeriska klasser som våra modeller skulle kunna identifiera. På grund av att storleken på datamängden var begränsad så behövde också antalet klasser begränsas. Följande klasser valdes:

1. Ålder,
2. Avstånd,
3. Datum,
4. Tid,
5. Pengar och
6. Övrigt.

För att skapa ett dataset utifrån de valda klasserna så hämtades meningar som innehöll numeriska uttryck från några olika källor. Först hämtades data från Wikipedia, där faktasidor om länder hämtades för att få meningar som innehöll datum. Faktasidan om Elon Musk hämtades för att få meningar som innehöll uttryck av klassen pengar. För att få uttryck av klassen avstånd hämtades bl.a sidan om marathon. Meningar på Wikipedia visade sig i princip bara innehålla årtal och datum. För att få mer data av klassen pengar så hämtades därför artiklar i form av analyser av aktier från webbplatsen Placera. För att få fler datapunkter som innehöll ålder och tid så användes OpenAI:s ChatGPT för att generera meningar som innehöll uttryck av dessa klasser. Chat GPT användes också för att komplettera och utöka antalet datapunkter av de andra klasserna.

När meningar hade samlats in så klassificerades alla numeriska uttryck manuellt. Ett kort program skrevs för att klassificeringsprocessen skulle effektiviseras. Programmet skrev ut en mening och vilket uttryck som skulle klassificeras i denna och indexet för det uttryckets klass matades då in. Programmet skrev då om uttrycken i meningarna så att de var på formen “N– <uttryck>–<index för klassen>” så att uttrycken och dess klass kunde läsas in av maskininlärningsmodellen när den skulle tränas och utvärderas.

Då tillräckligt många datapunkter klassificerats med sin korrekta klass inleddes arbetet med att träna upp modellen baserad på random indexing. Till att börja med skapades ett vokabulär där varje unikt ord sparades i ett set, utöver alla unika ord lades det även till ett element av formen <number> i setet. <number> stod för alla de numeriska uttrycken som förekom i korpuset, och användes för att olika numeriska uttryck inte skulle påverka varandra. När vokabuläret var byggt var nästa steg att skapa ordvektorer, och detta utfördes enligt random indexing - till att börja med assignerades en random vektor till varje unikt ord enligt metoden beskriven i inledningen av rapporten. Precis som för alla unika ord fick <number> en random vektor som då var samma för alla numeriska uttryck. I skapandet av dessa random vektorer var en utmaning att avgöra vilken dimension (längd) som passade det föreliggande problemet. Dimensionen är vanligtvis 2000, men då datasetet i detta fall var mindre valdes istället en dimension på 1000. Den andra parametern som skulle bestämmas vid skapandet av random vektorer var hur många element i varje random vektor som skulle ha ett värde skilt från 0 (dessa element blev då slumpmässigt genererade från mängden {-1,1}). Här var balansen viktig, då både ett för stort antal och för lågt antal nollskilda element kunde göra modellen mindre effektiv. Mängden nollskilda element sätts vanligtvis till 100 då man har en dimension på 2000, med en dimension på 1000 som utgångspunkt och med samma procentsats sattes antalet nollskilda element till 50.

I steget därpå skulle som tidigare beskrivet kontextvektorer skapas och uppdateras, men här skiljde sig vårt tillvägagångssätt något från den som implementerades i Labb 4. Istället för att skapa en kontextvektor för varje unikt ord som sedan uppdateras för varje gång ordet förekommer så skapades en kontext vektor för varje enskilt numeriskt uttryck. Denna kontextvektor parades sedan ihop med sin korrekta label och lagrades i en lista, vilket illustreras i bild 1. Kontextvektorerna i vår implementering baserades på ett fönster på 3 åt vänster, och 3 åt höger från fokusordet där detta var möjligt, d.v.s. i mitten av meningar som var 7 ord eller längre. I fall där det inte fanns 3 grannar åt båda hållen gjordes fönstret så stort som möjligt, se exempel på olika situationer i tabell 1.
Detta gjordes för varje mening i korpuset, så att det till slut fanns en lista med kontextvektor och tillhörande korrekt klass för alla numeriska uttryck i korpuset. Därefter importerades modulen sklearn.neighbors och den färdigimplementerade klassen Nearest Neighbors tränades på alla kontextvektorer genom metoden fit, detta illustreras i bild 2. Genom detta skapades kluster där numeriska uttryck som förekommit i en liknande kontext hamnade nära varandra. Eftersom kontextvektorer inte uppdaterades iterativt utan togs fram för varje gång ett numerisk uttryck förekom, fanns en risk att de kluster som skapats inte var helt felfria. T.ex. skulle 1.83 i “Jag är 1.83 meter lång” kunnat ha en kontextvektor liknande den för 17 i “Jag är 17 år gammal”, och några liknande exempel illustreras i bild 3 (såklart simplifierat då vektorrummet för dessa kluster egentligen är 1000 dimensioner). När en modell tränas enligt NN-klassen måste det avgöras vilket typ av likhetsmått som används för att skapa dessa kluster. Här har “cosine similarity” använts då det är ett mått som inte tar hänsyn till normen av kontextvektorerna då klustrena skapas, utan det är vinkeln som avgör likheten mellan två kontextvektorer.

I den sista delen av modellen togs en ny input emot, antingen som en enskild kort mening eller som en längre text. Så fort ett numeriskt uttryck påträffades så skapades en kontextvektor för det numeriska uttrycket på precis samma sätt som i ovan avsnitt beskrivet. Om ett grann-ord påträffades som inte förekommit i träningskorpuset (och således inte hade någon random vektor) så motsvarades dess random vektor av None. Med kontextvektorn skapad återfanns dess 15 närmaste grannar och deras klasser genom den importerade metoden kneighbors. Anledningen till att inte enbart den närmaste grannen hittades var att undvika att en kontextvektor med felaktig klass skulle ligga närmast kontextvektorn för det inkomna numeriska uttrycket. När de närmaste grannarna hämtats predicerades det numeriska uttrycket tillhöra den klass som var vanligast förekommande bland dessa grannar. Denna process illustreras i bild 4 (i bilden ses enbart tre kluster, i verkligheten fanns ett mycket större antal kluster).

Ett program som utförde regelbaserad klassificering skrevs också. Detta för senare jämförelse av de olika klassificeringsmetoderna. Denna algoritm består egentligen bara av flera villkor som kontrollerar förekomsten av specifika tecken i uttrycket och om ordet efter uttrycket är av en enhet som kan avslöja klassen av uttrycket. Alla villkor gås igenom då ett uttryck ska klassificeras. För varje villkor som uppfylls finns en regel som säger att detta ska ge ett “poäng” till en viss klass. Ett exempel på ett villkor är förekomst av “:” i uttrycket, som har bestämts ska ge ett poäng till klassen Tid. Efter att alla villkor har gåtts igenom för ett uttryck så prediceras klassen som har flest “poäng”. Om det är flera klasser som har lika många “poäng” så väljs en av dessa slumpmässigt, med lika stor sannolikhet för varje klass.

Metod för testning/evaluering

För att testa och jämföra de två metoderna av NER som tagits fram så används samma testdata för de båda modellerna. Eftersom att projektet är begränsat av mängden datapunkter och den ena modellen kräver träningsdata så kommer evalueringen att göras på resultatet av flera körningar av slumpmässigt utvald testdata från hela datamängden. Vid körning av ett test så väljs 15% av datamängden ut som testdata, resterande datapunkter tränas maskininlärningsmodellen upp på. För varje modell lagras sedan varje prediktion av en klass och den korrekta klassificeringen för varje testdatapunkt. Prediktionerna och de korrekta utfallen används sedan för att ta fram en confusion matrix för varje modell med hjälp av klassen confusion_matrix i modulen sklearn.metrics. För att ta fram data för precision, recall och accuracy så skickas samma data in i klassen classification_report som också finns i skleard.metrics modulen.

Vid körning av modellerna så kommer resultaten att skilja sig för olika körningar på grund av att testdatamängden är relativt liten och att testdatat väljs ut slumpmässigt. Därför kommer resultatet för mätvärdena av precision och recall för varje klass samt accuracy att beräknas genom att utfallen för 5 körningar läggs ihop och att måtten sedan beräknas utifrån summan av dessa utfall.

Resultat

Tabell 2 och 3 visar en aggregerad confusion matrix baserad på 5 körningar för den maskininlärningsbaserade respektive den regelbaserade modellen.

Tabell 4 och tabell 5 visar precision och recall för varje klass där tabell 4 baseras på data i tabell 2 och tabell 5 baseras på data i tabell 3.

Tabell 6 visar accuracy för maskininlärningsmodellen respektive den regelbaserade modellen.

Diskussion

Tabell 6 visar att det övergripande kvalitetsmåttet, accuracy, är något högre för maskininlärningsmodellen (0.74) jämfört med den regelbaserade modellen (0.71). Generellt beskrivet betyder det att maskininlärningsmodellen i snitt är något bättre på att predicera rätt klass, en vidare undersökning kring orsaker till detta kan initieras genom att undersöka måtten precision och recall.

Tabell 4 visar att den maskininlärningsbaserade modellen har en relativt jämn precision och recall för de olika klasserna. Detta kan bero på att mängden data av varje klass är relativt jämnt fördelad. Modellen har särskilt bra precision och recall för klassen pengar, detta kan tänkas bero på att meningarna som innehåller denna klass ser relativt lika ut, många meningar beskriver aktiepriser eller priser på saker. De numeriska uttrycken av klassen pengar efterföljs också ofta av en valuta som avslöjar mycket för modellen. Dessutom är antalet olika valutor i datasetet få, vilket ger goda möjligheter för modellen att tränas upp och predicera denna klass.

Klassen tid har låg precision och recall för maskininlärningsmodellen. Varför denna klass är svårare för modellen att predicera kan bero på flera saker. En orsak kan vara att väldigt många av meningarna som innehåller tid kommer från Wikipedia och att många av dessa har inte någon enhet omkring sig.

Flera uttryck skrivs till exempel på formen HH:MM:SS och formen 15,67 där den sista formen anger sekunder men utan enheten sekunder (eller s). Detta medför att denna modell, som predicerar baserat på kontext, inte kan urskilja likheter i kontexten mellan dessa meningar som innehåller en tid. Vidare så har de meningar av klassen tid som har en enhet efter sig många olika enheter som bl.a sekunder, timmar, dagar och år. En sista orsak som troligtvis leder till de låga värdena för denna klass är att klassen tid omfattar såväl tid i form av tidsintervall (t.ex 5 timmar) som klockslag (15:00) i datasetet. Klockslag och tidsintervall förekommer ofta i olika kontexter vilket gör att kontextvektorerna blir olika och man kan tänka sig att det blir som två kluster av vektorer inom denna klass.

Tabell 5 visar att den regelbaserade modellen har en genomgående hög precision på samtliga klasser förutom övrigt där den är så låg som 0.40, samtidigt som den har en mellan klasserna relativt volatil recall-nivå. En möjlig anledning till att den får så hög precision på alla klasser förutom övrigt är att för att klassificeras som något “icke-övrigt” så måste den uppfylla några väldigt specifika kriterier, vilket minskar sannolikheten att t.ex. ett avstånd klassificeras som en ålder. Detta är troligtvis även orsaken till att övrigt får en så låg precision - alla de uttryck som inte fångas upp av de väldigt specifika kriterierna blir klassificerade som övrigt. Då recall istället undersöks så kan konstateras att recall-nivån är hög, där de två klasser som drar ned snittet rejält är datum och tid. Som tidigare diskuterat kan den låga recallen av tid troligtvis förklaras av att det är en så mångfacetterad klass - det är en klass som förekommer i många olika former i datasetet och som dessutom innehåller tidsintervall såväl som klockslag, och därför blir den svår att klassificera genom explicita kriterier. Detsamma kan sägas om datum, som även den är en klass som förekommer på många olika sätt i korpuset. Från tabell 3 kan avläsas att datum 91 av 127 gånger klassificeras som övrigt, något som stöder förklaringen att det för sådana klasser är svårt att lyckas fånga alla uttryck med hjälp utav specifika kriterier. Den regelbaserade modellen hade kanske kunnat förbättras genom att kolla på ord framför uttrycket eftersom att många datum skrivs som “den 4/7” t.ex. Detta visar samtidigt på att den regelbaserade modellen är statisk och att det har vissa nackdelar.

Den främsta orsaken till att vissa resultat av recall och precision var särskilt låga för vissa klasser verkar (som tidigare nämnt) dels bero på mängden data. För den maskininlärningsbaserade modellen så leder brist på data till att den inte blir tillräckligt tränad och därmed att den helt enkelt blir sämre på att predicera. Hade antalet klasser minskats men fler datapunkter av varje klass använts så hade det kunnat vara fördelaktigt för modellen. Fler datapunkter hade också kunnat göra modellen bättre, men i detta projekt så lades mycket tid på att ta fram datasetet och mer tid kunde inte ha lagts på detta.

Vid körning av testerna av modellerna så varierade resultaten också ganska mycket. De körningar som noterades i rapporten för maskininlärningsmodellen blev något sämre än vad de vanligtvis brukade bli. Att resultaten varierade relativt mycket för olika körningar beror troligtvis på att datamängden var relativt liten. Detta leder till att datasetet som maskininlärningen tränas på kan variera mycket för varje körning eftersom att andelen som olika klasser utgör skiftar mycket mellan olika körningar. 15% slumpmässig testdata av hela datasetet kan medföra att ett par klasser kanske inte tränas alls men att dessa klasser samtidigt testas desto mer. För den regelbaserade modellen så kan skiftandet av testdata medföra att klasser som den är väldigt bra på att predicera utgör en stor majoritet för en specifik körning och tvärtom för en annan körning. Därför skulle den kanske ha kunnat köras för hela datasetet också eftersom att detta kanske fångar fler olika fall än att köra 5 slumpmässiga körningar där 15% av datasetet används. För att säkerställa hur andelen av olika klasser som testas och tränas påverkar resultatet så hade fler körningar behövt noteras med fler detaljer kring andelen av olika klasser och liknande.

Sammanfattningsvis så visar resultaten på att den regelbaserade modellen är mer stelbent än den maskininlärningsbaserade. Det som modellen klassificerar som exempelvis tid och datum är nästan alltid korrekt eftersom att den kan urskilja specifika syntax för dessa uttryck som inte finns för någon annan klass. Samtidigt så ligger det en svaghet i att inte kolla lika mycket på kontexten. Då det saknas tecken som särskiljer en klass på sättet som datum och tid gör så kan modellen inte göra bra prediktioner alls. Den maskininlärningsbaserade modellen som är baserad på kontext har istället mer jämna resultat för hur bra den predicerar de olika klasserna. Den är inte utmärkt på att predicera en speciell klass, men relativt bra på alla. Resultatet visar dock på inlärningsprincipens fördelar. En klass som förekommer i flera olika format och kontexter är den bättre på att predicera än den enklare modellen vilket syns i resultaten för recall och precision för datum. Resultaten lär har påverkats mycket av storleken på datasetet inte var särskilt stort. Detta resulterade i lite träningsdata och testdata vilket medförde stora variationer mellan körningarna på grund av att testdatat valdes ut slumpmässigt.

Slutsats

Så vilken modell predicerade numeriska uttryck bäst? Det tråkiga svaret är att det beror på, här hade hypotesen behövt varit mer specifik i sin mening. Beroende på vad modellen ska användas till kan den ena modellen vara bättre än den andra; Om användningen av modellen har strikta krav på att en icke-övrig klass inte felaktigt får prediceras så är den regelbaserade modellen bättre, med en precision > 0.9 för samtliga icke-övriga klasser. Ställer användningen istället krav på en över samtliga klasser jämnare prestation så är maskininlärningsmodellen att föredra. Något som med säkerhet kan konstateras är att kontexten till ett numeriskt uttryck avslöjar väldigt mycket om dess klass, vare sig det används som den enda parametern eller som en del i ledet vid en klassificering.

För att se till möjliga felkällor kan man titta på de avgränsningar som gjorts - de klasser som valts samt mängden data. När det gäller klasser så hade en möjlig lösning kunnat sett ut på två sätt; Å ena sidan hade mängden klasser kunnat reducerats så att varje klass fick fler datapunkter att träna på och således känt igen fler kontexter, vilket möjligtvis lett till att modellen oftare predicerat rätt klass. Å andra sidan hade de klasser som användes kunnat utvidgats, så att var klass fått än mer specifika kontexter. Exempelvis hade “tid” kunnat delats upp i “klockslag” och “tidsintervall”, och “datum” delats upp i “datum” och “årtal”. Detta andra tillvägagångssätt hade dock kunnat leda till en större slumpmässighet vid bildandet av kluster (som tidigare diskuterats) och således lett till sämre prediceringar, särskilt vid en begränsad mängd data. Mängden data spelade vidare troligtvis en stor roll för de resultat som togs fram i detta projekt, och för att testa modellens potential så hade det varit intressant att ha ett mycket större och varierat dataset. Trots det uppfyllde projektet sannerligen sin roll i att för utförarna introducera projekt inom maskininlärning.

Appendix

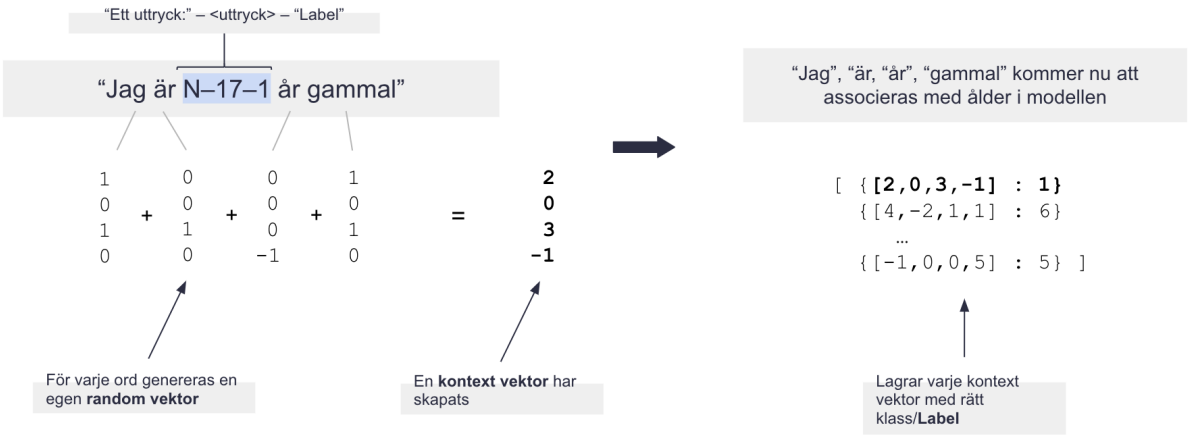

Bild 1: Skapandet och lagrandet av kontextvektorer och dess korrekta klass

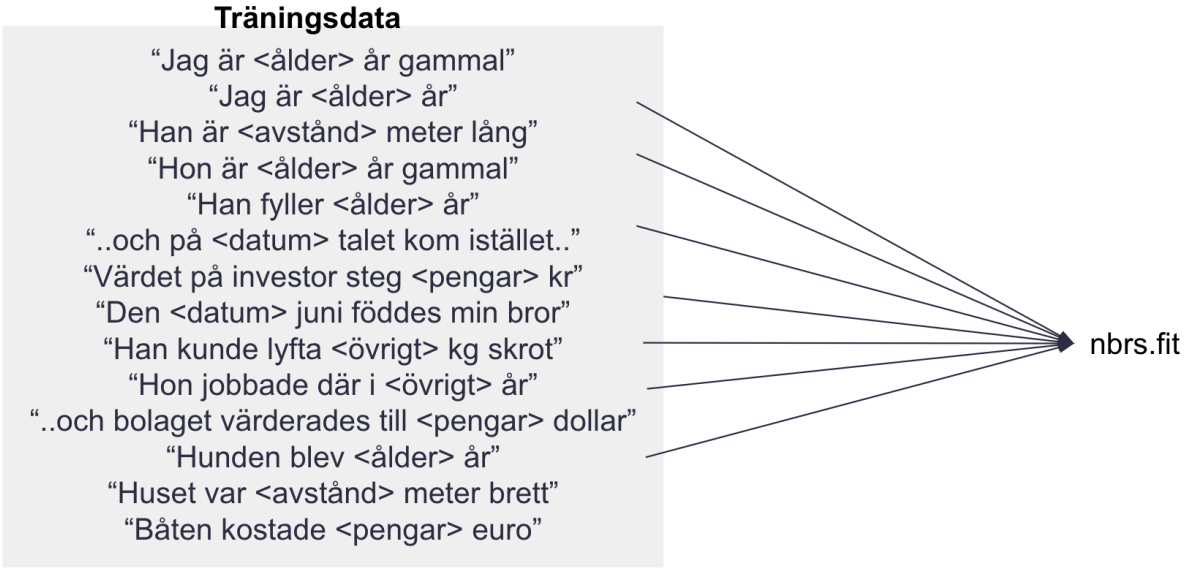

Bild 2: Uttryck med sin korrekta klass som tränar modellen genom nbrs.fit (uppmålat för illustrativa syften, notera att det egentligen är kontextvektorn och dess korrekta label för uttrycket som körs i nbrs.fit)

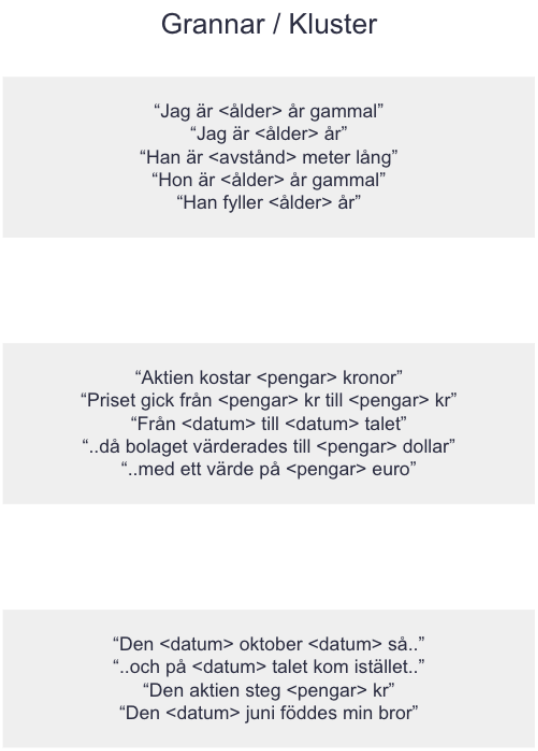

Bild 3: Grannar/Kluster som skapas vid träning av modellen med cosine similarity som likhetsmått

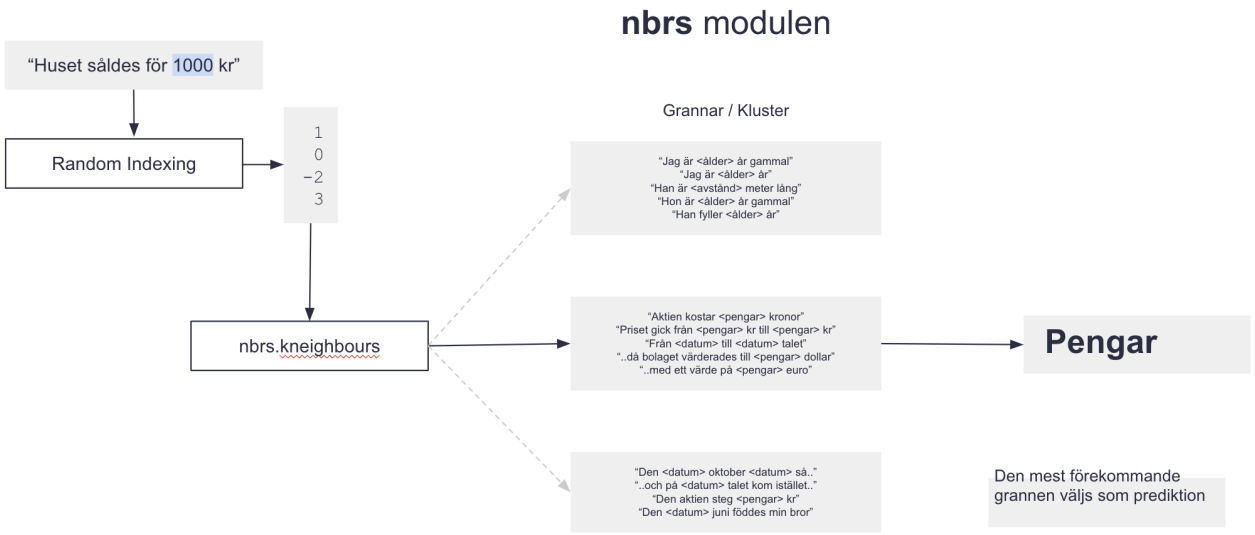

Bild 4: Exempel på körning av en mening i maskininlärningsmodellen

Mening	Grannar åt vänster	Grannar åt höger
“Hon var bara 17 år gammal då”	3 st [Hon, var, bara]	3 st [år, gammal, då]
“13 kr kostade huset”	0 st []	3 st [kr, kostade, huset]
“Det var på 2000-talet”	3 st [Det, var, på]	1 st [talet]

Tabell 1: Exempel på storleken på fönster i olika situationer

122,00	0,00	2,00	10,00	2,00	0,00
3,00	93,00	2,00	13,00	1,00	0,00
9,00	10,00	86,00	13,00	4,00	5,00
16,00	27,00	4,00	81,00	2,00	5,00
0,00	0,00	2,00	5,00	107,00	3,00
16,00	14,00	12,00	10,00	7,00	75,00

Tabell 2: Confusion matrix för modellen baserad på maskininlärning

121,00	0,00	0,00	0,00	4,00	11,00
0,00	100,00	0,00	0,00	1,00	11,00
0,00	0,00	36,00	0,00	0,00	91,00
12,00	0,00	0,00	73,00	2,00	48,00
0,00	0,00	0,00	0,00	88,00	29,00
1,00	8,00	0,00	0,00	0,00	125,00

Tabell 3: Confusion matrix för den regelbaserade modellen

	Precision	Recall
Ålder	0,73493976	0,89705882
Avstånd	0,64583333	0,83035714
Datum	0,7962963	0,67716535
Tid	0,61363636	0,6
Pengar	0,8699187	0,91452991
Övrigt	0,85227273	0,55970149

Tabell 4: Precision och recall för modellen baserad på maskininlärning

	Precision	Recall
Ålder	0,90298507	0,88970588
Avstånd	0,92592593	0,89285714
Datum	1	0,28346457
Tid	1	0,54074074
Pengar	0,92631579	0,75213675
Övrigt	0,3968254	0,93283582

Tabell 5: Precision och recall för den regelbaserade modellen

	Accuracy
Maskininlärning	0,74113009
Regelbaserad	0,71353482

Tabell 6: Accuracy för respektive modell

Referenser

Feedstock, *Unlocking business value from Named Entity Recognition*, u.å, https://www.feedstock.com/quick-reads/unlocking-business-value-from-named-entity-recognition-ner/ [hämtad 2022-12-15]

Johan Boye, *F7 - Text classification,* 2022, https://www.csc.kth.se/~jboye/teaching/language_engineering/7_text_classification.pdf [hämtad 2022-12-17]

Johan Boye, *F9 - Lexical Semantics,* 2022, https://www.csc.kth.se/~jboye/teaching/language_engineering/9_word_embeddings.pdf [hämtad 2022-12-15]

Wikipedia,*Named-entity recognition,*2022, https://en.wikipedia.org/wiki/Named-entity_recognition [hämtad 2022-12-15]