Logiska symboler:

∧ ∨ →

¬ ⊨ ⊢

Begrepp:

Globalt/lokalt optimum

Engelskans “literal” är ett begrepp i matematisk logik som beskriver en atomär sats. I den här texten så kommer begreppet att användas på sitt originalspråk då det inte finns en passande svensk översättning. Det finns två olika former av literaler:

* En positiv literal är endast en atomär sats. Ex. A.
* En negative literal är en negation av en atomär sats. Ex. ¬ A

Källa: https://en.wikipedia.org/wiki/Literal\_(mathematical\_logic)

Induktiv Logisk Programmering: att lösa klassifikationsproblem med första ordningens logik

Vi skapar regler för klassifikation som formuleras i första ordningens predikatlogik. Reglerna är skrivna som Horn klausuler i implikationsform. En sådan regel har ett vänster led som är en ensam literal eller en konjunktion av literaler. Samt ett högerled som är en ensam literal. En sådan regel kan exempelvis se ut såhär:

A ∧ B ∨ ¬C → D

Källa: <https://en.wikipedia.org/wiki/Horn_clause>

Regler som kan lösas ut med omvänd resolution, vilket gör både input och output data högt integrerbart med andra system. Regler som lätt går att skriva i naturligt språk.

The FOIL stopping condition

* Skriv kanske generellt om FOIL
  + En tidig algoritm i ILP
  + Har många problem men ett bra stopp-condition för området
* Skriv här om pre-pruning
* Skriv om information gain här också
* Separate and conquer
* Sökrymden är alla möjliga kombinationer av variabler i argumenten av de använda

REP

* Introducerar post-pruning
* Delar upp träningsmängden i growing-data och pruning-data
* En regel som täcker alla beskrivningar (exempel) i growingdata
* Tar sedan bort Horn klausuler tills en borttagning resulterar i en försämrad accuracy på pruning-datan.

IREP

* Incremental reduced error pruning
* Istället för att bara utföra pruning på det slutgiltiga regelsettet så utförs pruning varje klausul som skapas, innan den eventuellt läggs in i regelsettet
* Slut-villkoren skiljer sig här också.
* När en individuell klausul inte tillför ytterligare prediction accuracy på pruningmängden än en tom klausul (false) så kommer algoritmen att avslutas.
  + I en tillämpning som kallas IREP2 så utförs även ett mått på ”renheten” i en regel. Detta beräknas som
    - p / p + n
  + Där p (n) är alla positiva (negativa) exempel som täcks av regeln.
* Beskriv sedan hur Cohens IREP är bättre

Det är en hillclimbing algoritm. Genom att iterativt ta fram kandidatlösningar och sedan direkt förbättra dessa lösningar, så når den tillslut en slutgiltig lösning som är mycket bättre än någon av de tidigare dåligt formulerade lösningarna. Försöker hitta en global optimum genom att hitta ett antal lokala optimum. I vårt fall så har vi en samling attribut. För att hitta ett globalt optimum så försöker vi hitta ett lokalt optimum för alla formulerbara regler. Vi kan säga OM voting\_on\_babies == yes and gun\_law == n THEN republican. Hur kan vi ändra den regeln, genom att ta bort någon klausul lokalt för att uppnå en bättre FOIL measure. Vi gör såhär med olika kandidatregler, där vi lokalt optimerar varje kandidatregel innan den läggs in i vår större regelmängd.

Det uppstår emellertid ett problem här. Eftersom att vi för det första har delat upp träningsmängden i growing och pruning set så kommer det finnas exempel som skulle varit informativa för regelbygget som vi går miste om. En annan sak bidrar till samma problematik vi raderar de exempel som redan täcks av regler. Säg att algoritmen har kört ett tag och skapar en regel som inte är lokalt optimal, givet resterande exempel, men den egentligen är globalt optimal (alla exempel finns med, även de i pruning set) så kommer den inte hittas. Anledningen att denna teknik fortarande används är att den är effektiv i minnes och tidskomplexitet (se artikeln om separate and conquer, sid 22).

IREP åtgärdar detta genom att iterativt dela upp growing och pruning set. På det viset fångas fler exempel in.

“The basic problem of this method is its myopia due to the elimination of all

one-step refinements but one. If a refined rule is not locally optimal, but one

of its refinements is the global optimum, hill-climbing will not find it (unless

it happens to be a refinement of the local optimum as well)”

För att vår regel skall vara accepterbar så behöver den uppnå en godtagbar MDL. MDL är ett mått som ger oss en idé om generaliserbarheten i en lösning. Enklare är bättre.

Top-down sökning är grunden för separate and conquer sökning

* Dela upp träningsmängden i growing och pruning set
* Växa fram en kandidatregel
* Förfina kandidatregeln genom pruning
* Ta bort alla (positiva?) exempel som täcks av regeln
  + Varför bara positva?

Alla mått på sida 29 i SEPARATE-AND-CONQUER RULE LEARNING

OPTIMISERING, använder DL för att göra hypoteser mer minimala och generaliserade

FOIL, använder PURITY för att jämföra nya regler, hur många positiva exempel täcks in av alla exempel?

PRUNING, literaler tas bort tills att ACCURACY försämras på pruningdatan

Accuracy ANVÄNDS I PRUNING

A(r) = p + (N - n) / P + N

Purity ANVÄNDS I FOIL

Purity i en regel är kvoten av hur många positiva exempel som täcks av en regel jämfört med all exempel som täcks av regeln.

P(r) = p / p + n

Vi behöver ett mått för att beskriva komplexiteten i en regel

Quinlan 1990 använder en variant av MDL i FOIL som stop-kriterium

Minimum description length

Hittar en avvägning mellan hur mycket information som behövs för att kommunicera en hypotes och hur mycket information som behövs för att beskriva en samling exempel med hjälp av hypotesen.

M DL(H ) = I (H ) + I (EjH )

Beskrivningslängden för en hypotes H är

Först tränar den sen kan den köras

Skriv ett stycke om Noisy Data

Overfitting the data (OVERfitting avoidance bias)

FÖRKLARA ATT REGELMÄNGDEN == HYPOTES == TEORI OCH ANVÄND BEGREPPEN LITE BÄTTRE KANSKE VA

Kanske skriv någonting om REP??? Eller så är det onödigt

Skriv om att sökrymden är jävligt stor, växer exponentiellt med antalet attribut

IREP tidskomplexitet: *n* log2 *n* [Cohen, 1993]

# Inductive Logic programming

**Hur skapar vi en generaliserbar hypotes?**

Många dataset innehåller motsägelsefulla data, där inte alla datapunkter är lika informativa. Detta kallas noisy data. Kongressröster i olika politiska frågor är ett exempel på en sådan domän. Där en stor mängd republikaner röstar ja i en viss fråga, men det också finns en annan grupp republikaner som röstar nej. Det finns en risk när vi tränar en klassificerare på noisy data att den skapar en väldigt komplex teori som täcker alla träningsexempel perfekt. Men den teorin kommer med stor sannolikhet prestera dåligt när vi sedan skall köra algoritmen för att förutsäga klasser i en okänd datamängd. Detta problem är känt som ”overfitting the data”. På svenska kan vi beskriva det som att teorin har en för snäv passform efter träningsdatan. Klassificeraren behöver en teori som är mer ”one-size fits all” än skräddarsydd[[1]](#footnote-1).

Ett sätt att åstadkomma detta, som används i IREP och RIPPER algoritmen är metoden pruning. Pruning används inom flera områden av maskininlärning. Många har säkert stött på begreppet tidigare i beslutsträdsinlärning. Där det används för att simplifiera ett beslutsträd genom att ersätta ett delträd som bara täcker in ett fåtal exempel med en lövnod. Principen är densamma i ILP. Genom att ta bort, negera, begränsa längd eller ändra sammanbindningen av Hornklausuler så görs hypotesen mer generell.

# Incremental Reduced Error Pruning (IREP)

Den här algoritmens mål är att skapa en regelmängd av logiska satser som förklarar maximalt många exempel i träningsdata. Givet att algoritmen körs på kategoriska data så försöker den klassificera datainstanser som positiva eller negativa för en given klass. I fallet med numeriska data så används olikheter (ex. ’ > ’) istället för kategorier.

**Indata och utdata**

Indata till algoritmen är en samling attribut, där varje attribut kan ta olika kategoriska eller numeriska värden. Ett av attributen är också klassattributet. Klassattributet är det attribut för vilket algoritmen lär sig regler. Träningsdata är en samling exempel där varje datainstans har olika attributvärden. Utdata från träning är som tidigare beskrivet logiska uttryck.

Klassificering går till genom att IREP tar en tidigare okänd datamängd som input och ger förutsägelser av klassattributet på individuella datapunkter som utdata. Klassförutsägelser avgörs av den predikatlogiska regelmängden. Vi tar ett exempel från den implementation jag utfört:

Domänen är politiska ämnen som har röstats om i USA:s representanthus (UCI). Varje instans i exempeldata är en individuell politiker. Klassattributet är vilket parti en individ tillhör.

Träningsdata och attribut kan se ut såhär:

Attribut:

Attributnamn: Handicapped-infants

Värden: yes, no

Attributnamn: Synfuels-corporation-cutback

Värden: yes, no

Klassattributnamn: Party

Värden: republican, democrat

Exempeldata:

1. Handicapped-infants, yes; Party, republican
2. Handicapped-infants, no; Party, democrat
3. Synfuels-corporation-cutback, yes; Party, republican

En resulterande regelmängd efter träning kan se ut såhär:

Regel:

IF [{Handicapped infants == yes} AND {Synfuels-corporation-cutback == yes}] THEN republican

ELSE democrat

Utdata från klassifikation kan se ut såhär:

Class Cases Classified

republican 20 14

democrat 24 9

**Framtagning av regler**

IREP är en iterativ hillclimbing algoritm. I varje iteration så försöker den ta fram en ny regel som motsvarar ett lokalt optimum. Genom att sedan optimera och generalisera både varje ny regel och den slutgiltiga regelmängden så försöker den nå ett globalt optimum.

En regel tas fram genom att skapa ett flertal kandidatregler baserade på något attribut i träningsdata och sedan jämföra dem med varandra. I jämförelsen används en funktion för mätvärdet ”purity”. På svenska kan vi säga att algoritmen bedömer hur renodlad en regel är.

Purity: P(*r*) = p / p + n

Där *r* är en ny regel, p är alla positiva exempel som täcks av regeln och n är alla negativa exempel som täcks av regeln. Negativa exempel är datainstanser där literalen för klassattributet är negerat, ex. ¬republican. Av alla kandidatregler i en iteration så väljs den som har högst värde väljs ut och alla exempel som täcks av den nya regeln tas bort från träningsdatan. Vi vill ta bort alla täckta regler så att redundanta regler inte skapas samt att algoritmen avslutas när det inte längre finns några exempel kvar att skapa regler för i träningsmängden.

**Förenkling av regler**

Som tidigare nämnt så är noisy data ett problem i klassifikation. Därför används pruning för att testa och modifiera nya regler innan de inkluderas i regelmängden. På så vis undviks overfitting. I IREP går detta till genom att träningsdata upp i två delar: grow-data och pruning-data (fördelningen är 2/3 grow-data och 1/3 pruning-data). Nya regler skapas för att förklara grow-data och testas sedan mot pruning-data. IREP använder två olika typer av pruning: pre-pruning och post-pruning.

Pre-pruning är heuristiska stoppvillkor som förhindrar algoritmen från att skapa onödiga regler. I IREP och RIPPER så slutar algoritmen skapa regler när en ny regel har en felkvot som är högre än 50%. Felkvoten uppskattas med en funktion som benämns som ”Accuracy”.

A(r) = p + (N - n) / P + N

Där r är en regel, p (n) är alla positiva (negativa) exempel som täcks av regeln och P (N) är samtliga positiva (negativa) exempel. Notera att endast pruning-data är indata till accuracy.

Accuracy används inte bara som ett stoppvillkor, utan tillämpas även som en form av post-pruning. Post-pruning är en metod där en individuell regel, eller ett helt regelset utvärderas och möjliga förenklingar tillämpas. Det här konceptet är det mest centrala med IREP algoritmen. Direkt efter att en regel har skapats så förenklas den iterativt tills att vidare förenklingar försämrar regelns felkvot (accuracy-funktionen) i klassificering av pruningdatan. För att förenkla en regel i IREP så raderas en eller flera literaler i hornklausulens vänsterled. En mängd med förslag till raderingar skapas för den nya regeln. Den radering som maximerar accuracy-funktionen väljs ut. Namnet på IREP: Incremental reduced error pruning, denoterar just processen av att stegvis förenkla individuella regler för att minska felkvoten.

**Algoritmen**

Sedan så prunas regeln. I pruningen så sker en iterativ radering av literaler i regeln för att göra den mer generell. Pruningen avslutas när en radering skulle resultera i en försämring av regelns prediktionsförmåga på prune-data. Denna process upprepas tills det inte längre finns några positiva exempel kvar i träningsmängden eller värdet för en ny regel är lägre än en förbestämd acceptansnivå.

**begin**

Ruleset := Ø

**while** Pos ≠ Ø

/\* grow and prune a new rule \*/

split (Pos,Neg) into (GrowPos,GrowNeg)

and (PrunePos,PruneNeg)

Rule := GrowRule(GrowPos,GrowNeg)

Rule := PruneRule(PrunePos,PruneNeg)

**if** the error rate of Rule on

(PrunePos,PruneNeg) exceeds 50% **then**

**return** Ruleset

**else**

add Rule to Ruleset

remove examples covered by Rule

from(Pos,Neg)

**endif**

**endwhile**

**return** Ruleset

**end**

Styrkan i den här algoritmen kommer ur att den med olika metoder förenklar, optimiserar och generaliserar varje beståndsdelarna i den slutgiltiga lösningen.

IREP är en hillclimbing algoritm. Under varje steg i sökningen så försöker den ta fram en regel som motsvarar ett lokalt optimum. För att undvika overfitting så används pruning. Träningsdata delas upp i ett grow-dataset och ett prune-dataset (split 2/3). En regel tas fram genom att skapa ett flertal kandidatregler baserade på exempel grow-data och sedan jämföra dem med en funktion som uppskattar varje individuell regels värde. Regeln som har högst värde väljs ut och alla exempel som täcks av den nya regeln tas bort från träningsdatan. Sedan så prunas regeln. I pruningen så sker en iterativ radering av literaler i regeln för att göra den mer generell. Pruningen avslutas när en radering skulle resultera i en försämring av regelns prediktionsförmåga på prune-data. Denna process upprepas tills det inte längre finns några positiva exempel kvar i träningsmängden eller värdet för en ny regel är lägre än en förbestämd acceptansnivå.

Genom att iterativt ta fram kandidatlösningar och sedan direkt förbättra dessa lösningar, så når IREP tillslut en slutgiltig lösning som är mycket bättre än någon av de tidigare individuella reglerna

Det finns två olika typer av pruning, pre-pruning och post-pruning. IREP och RIPPER använder en kombinerad version av bägge som förklaras i nästa avsnitt.

IREP åtgärdar detta genom att iterativt dela upp growing och pruning set. På det viset fångas fler exempel in.

Algoritmen kan växa fram en exponentiellt stor mängd regler. Men tiden det skulle krävas att utforska hela sökrymden är oönskvärt stor. Därför så introducerar vi en heuristik i form av ett stoppvillkor. Pre-pruning är stoppvillkor som förhindrar algoritmen från att växa fram onödiga regler. I IREP och RIPPER så slutar algoritmen växa fram regler när en ny regel skapas som har en felkvot som är högre än 50%. Felkvoten uppskattas med en funktion som benämns som ”Accuracy”.

A(r) = p + (N - n) / P + N

Där r är en regel, p (n) är alla positiva (negativa) exempel som täcks av regeln och P (N) är samtliga positiva (negativa) exempel i hela pruning-datamängden. Accuracy funktionen fungerar inte bara som ett stoppvillkor, utan den tillämpas även som en form av post-pruning. Post-pruning är en metod där en individuell regel, eller ett helt regelset utvärderas och möjliga förenklingar tillämpas. Det här konceptet är det mest centrala med IREP algoritmen. Direkt efter att en regel har skapats så förenklas den iterativt tills att vidare förenklingar försämrar regelns felkvot (accuracy-funktionen) i klassificering av pruningdatan. För att förenkla en regel i IREP så raderas en eller flera literaler i hornklausulens vänsterled. En mängd med förslag till raderingar skapas för den nya regeln. Den radering som maximerar accuracy-funktionen väljs ut. Namnet på IREP: Incremental reduced error pruning, denoterar just processen av att stegvis förenkla individuella regler.

**Problem med Pruning**

Det uppstår emellertid ett problem här. Eftersom att vi för det första har delat upp träningsmängden i growing och pruning set så kommer det finnas exempel som skulle varit informativa för regelbygget som vi går miste om. En annan sak bidrar till samma problematik vi raderar de exempel som redan täcks av regler. Säg att algoritmen har kört ett tag och skapar en regel som inte är lokalt optimal, givet resterande exempel, men den egentligen är globalt optimal (alla exempel finns med, även de i pruning set) så kommer den inte hittas. Anledningen att denna teknik fortarande används är att den är effektiv i minnes och tidskomplexitet (se artikeln om separate and conquer, sid 22).

IREP åtgärdar detta genom att iterativt dela upp growing och pruning set. På det viset fångas fler exempel in.

**Att optimiera en samling regler**

Skriv här om RIPPER

**Min implementation**

Beskriv här vad jag gjort och vilka slutsatser jag kan dra

**Diskussion och slutsatser**

FOIL – pre pruning (purity)

Post Pruning

IREP – post pruning (accuracy)

Förklara kort vad regel-induktion är

* Hur det fungerar
* Varför det är bra
* Vad det har använts till/ kan användas till
* Kanske kort om begränsningar generellt

Beskriva vilken form av data man får ut i ILP, alltså Horn Clauses. Regler som kan lösas ut med omvänd resolution, vilket gör både input och output data högt integrerbart med andra system. Regler som lätt går att skriva i naturligt språk.

Jag kommer i den här texten redogöra för grunderna i en av de enklaste, effektivaste, men också högpresterande algoritmen för induktiv framtagning av predikatlogiska regler: RIPPER*k*. Algoritmen kom till som en utveckling av IREP som i sin tur var en förbättring av REP.

FOIL-stopp kriterium (sluta skapa regler när det inte finns några **positiva** exempel kvar) är en form av pre-pruning. Somliga träningsexempel (negativa i FOIL), ignoreras för att regeln inte skall perfekt matcha mot all träningsdata. Med andra ord så ignoreras här potentiellt sett **negativa** exempel i datan.

REP-algoritmen introducerade post-pruning. Med denna teknik så finns större möjligheter för at tf

Inductive rule

Beskriva REP (reduced error pruning)

Idén med reduced error pruning är att tillämpa en sorts bottom-up greedy hillclimbing algoritm. Algoritmen börjar med en tom regelmängd och sedan tar den fram en regel som täcker in samtliga beskrivningar.

Beskriva IREP (Incremental reduced error pruning)

Beskriva FOIL

Beskriva IREP\* och RIPPERk

Förklara min implementation

Beskriva algoritmen?

Top down process i att ta fram regler.

* Generate-then-test apporoach
* Start with short, general clause
* Specialise by applying substitutions and/or adding literals

Istället för bottom-up där man börjar med en regel som täcker in alla data och sedan skär ner från den.

REP algoritmen tendererar att overfitta datan. Därför så modifierar Cohen 1995 algoritmen

Det som skiljer RIPPER från IREP\* är att RIPPER gör

# Incremental reduced error pruning (IREP)

# Implementation

Jag har använt algoritmen RIPPER*k* för att utföra ett antal experiment där algoritmens träffsäkerhet och regellängd har undersökts.

Stopvillkoret i IREP förä

RIPPER*k* optimiserar regler *k* ggr genom att skapa revisioner och ersättningar för det regelset som returneras av IREP. För varje regel i Regelmängden så skapas två nya regler. En regel som är en ersättning av originalregeln och en revision av regeln. Revisionsregeln skapas genom att girigt lägga till villkor med originalregeln som utgångspunkt. Ersättningsregeln är en regel som skapas från en tom regel och sedan prunas för att minimera felkvoten för hela regelsettet på pruning-data. Sedan så väljs regeln med lägst DL-värde.

Träningsmängd (95%). Testmängd (5%). Pruning på, k=2 (två rundor med optimisering).

**Class Cases Classified**

republican 8 6

democrat 14 8

Accuracy: 63.6363636364%

Det dataset jag har använt är tillgängligt genom UCI Machine Learning Repository (Schlimmer, 1987).

Använda min implementation för att förklara vad RIPPER-algoritmen gör.

OVERFITTING AVOIDANCE BIAS!!!!!

* I vissa domäner så undviker den overfitting FÖR MYCKET!

# Nyckelord

Inductive Logic Programming (ILP), Noise, Noisy data, FOIL, REP, IREP, RIPPER, Pruning, Post-pruning, Pre-pruning

FOIL = First Order Inductive Learning

Foil, greedy hillclimbing in hypothesis space.

Stopping criteria i FOIL är en form av Pre-pruning:

* Varför?
* Vad är pre-pruning,

Post-pruning introducerades till ILP i och med REP:

* Varför?
* Vad är post-pruning

FOIL:

* En regel får inte täcka några negativa exempel

Börjar med en tom regel

Tills det endast återstår positiva exempel.

Sedan skapar man en första hypotes

Ta bort alla positiva exempel som täcks av regeln

**Källförteckning**

Schlimmer, J. (1987). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. Hämtad Januari 04, 2018, från https://archive.ics.uci.edu/ml/datasets/Congressional Voting Records

Russel & Norvig. Artificial Intelligence a Modern Approach. Section 19.5 (sida 788) “Inductive Logic Programming”.

1. Den saknar uppenbarligen klädsmak! [↑](#footnote-ref-1)