Regel-gebaseerde regressie

Capita selecta computerwetenschappen
Artificiële intelligentie (|H05N0a|)

Philippe Tanghe Li Quan

6 april 2011

Inhoudsopgave

1 Inleiding

Het doel van de opgave is een algoritme te schrijven dat een regressiemodel leert. Dit regressiemodel moet gebaseerd zijn op een verzameling regels. De individuele regels worden geleerd met behulp van constraint programming via het Gecode systeem. Het samenstellen van de verzameling regels gebeurt in Java.

2 Constraint programming

De individuele regels worden via Gecode als volgt geleerd: een itemset moet de data zo goed mogelijk in twee groepen splitsen (nl., voorbeelden die de itemset bevatten en die dat niet doen), waarbij de kwaliteit van de splitsing gemeten wordt door de mean squared error.

Voor het propageren wordt er een ondergrens gezocht voor deze error. Deze ondergrens wordt gebruikt om de zoekruimte te verkleinen (itemsets met een hogere error kunnen genegeerd worden).

3 Regressiemodel

3.1 Opbouwen beslissingsboom

Voor het opbouwen van het regressiemodel (zie algoritme 1) wordt het constraint programming systeem herhaaldelijk opgeroepen. Er wordt een beslissingsboom opgesteld door recursief op de gecoverde en niet-gecoverde voorbeelden het constraint programming systeem opnieuw los te laten tot de set van voorbeelden klein genoeg is (parameter MAX_NB_LEAF).

Algorithm 1: buildRegressionModel(data, itemsets, items)

```
if nb instances of data ≤ MAX_NB_LEAF then
    prediction ← average (OR mean) of class values of instances;
    add items to itemsets (as last rule);
else
    items2 ← find best pattern with rimcp on data;
    items2 ← items2 ∪ items;
    dataC ← instances of data that cover items in items2;
    dataNC ← instances of data that don't cover items in items2;
    buildRegressionModel(dataC, itemsets, items2);
    buildRegressionModel(dataNC, itemsets, items);
end
```

Van de itemsets die door een set voorbeelden gecoverd zijn, wordt de unie genomen en zo bekomt men dan een regel voor die verzameling van

voorbeelden. Er wordt zo een geordende regelset samengesteld (doordat de boom depth-first wordt opgebouwd). Dit heeft als gevolg dat de voorbeelden, die geen enkele itemset coveren, onder de laatste regel vallen. Dit is de else-regel, die een lege itemset voorstelt. Ter verduidelijking is het misschien nodig op te merken dat een regel enkel oplegt welke items gecovered moeten zijn en niet welke niet gecoverd mogen zijn.

Voor elke leaf van de beslissingsboom (dit komt overeen met een regel/itemset) wordt een voorspelling gemaakt. Deze voorspelling is het gemiddelde (of mediaan) van de labels van de voorbeelden in de leaf.

3.2 Pruning

Het regressiemodel kan nog vereenvoudigd worden. Hiervoor kunnen bepaalde regels van dit model worden gepruned (zie algoritme 2 en 3 — data stelt de trainingset met de oorspronkelijke labels voor, predOr de trainingset met de voorspelde labels, itemsets de gevonden itemsets).

Algorithm 2: pruneRules(data, predOr, itemsets)

```
newPredOr ← predOr;
repeat
    itemsets ← pruneRule(data, newPredOr, items);
    apply rules of itemsets on newPredOr;
until size new ruleset == size old ruleset ;
return itemsets;
```

Algorithm 3: pruneRule(data, predOr, itemsets)

```
bestRulesSoFar ← itemsets;
tempData ← predOr;
bestError ← error of predOr on data;
for each rule in itemsets do
    tempRules ← itemsets \ rule;
    apply rules of tempRules on tempData;
    error ← error of tempData on data;
    if (error - RULECOST) < bestError then
        bestError = (error - RULECOST);
        bestRulesSoFar ← tempRules;
    end
end
return bestRulesSoFar;
```

Het model moet zonder deze regels een minstens zo goede accuracy hebben op de trainingsvoorbeelden. Dit laatste criterium is in de implementatie nog verzacht. In de implementatie worden regels een voor een gepruned met als

voorwaarde dat de error op de trainingsvoorbeelden niet te sterk toeneemt (de error mag namelijk met een constante waarde toenemen—in het algoritme aangeduid als *RULECOST*—die ervoor zorgt dat eenvoudigere modellen met minder regels geprefereerd worden in overeenstemming met het Occam's razor-principe).

4 Experimenten invloed parameters

Om ons regressiemodel te evalueren, werd de gegeven dataset *auto93bin* opgesplitst in een training- en een testset, die respectievelijk 2/3 en 1/3 van de originele dataset bevatten.

Tabel 1 toont verschillende parameterkeuzes en hun invloed op het aantal regels, en de bijbehorende L_1 afstand tussen de voorspelling en de werkelijke labelwaarde (zowel voor de training- als de testset). De labelwaarden zijn bekomen door de gemiddeldes te nemen als voorspelling.

prune	*RULECOST*	*MAX_NB_LEAF*	nb of rules	error trainingset	error testset
false	/	3	33	67.33	158.77
true	5	3	13	87.90	158.47
false	/	5	20	85.92	143.73
true	5	5	12	99.04	142.38
false	/	7	15	92.35	141.78
true	5	7	10	106.33	139.90
false	/	10	11	102.05	139.84
true	5	10	9	109.06	137.96
true	0	3	33	67.33	158.77
true	0	5	20	85.92	143.73
true	0	7	15	92.35	141.78
true	0	10	11	102.05	139.84
true	3	3	15	80.13	158.47
true	3	5	14	92.03	143.73
true	3	7	12	99.35	141.78
true	3	10	11	102.05	139.84
true	10	3	12	94.9	158.47
true	10	5	8	124.32	142.38
true	10	7	8	118.53	143.39
true	10	10	8	114.29	137.96

Tabel 1: Voorspelling met gemiddelde. Invloed van de parameters op de training en test set error.

Een grotere waarde voor *MAX_NB_LEAF* zorgt uiteraard voor minder regels; de error op de trainingset wordt over het algemeen hoger terwijl de error op de testset lichtjes daalt. Dit betekent dat er aanvankelijk overfitting gebeurde. Bij te grote leaves zal uiteindelijk de fout op de testset weer stijgen (aangezien er dan te weinig regels zijn voor een goed model).

De parameter *RULECOST* bepaalt de trade-off tussen het aantal regels

en de fout op de trainingset: bij grotere waarden wordt er meer gepruned. Minder regels leidt echter meestal tot een grotere fout op de trainingset, dus is het nodig om een voldoende hoge *RULECOST* te hebben om eenvoudigere modellen te bekomen. Vaak lijkt pruning een iets kleinere error te hebben op de testset, wat te verklaren valt doordat het gevonden model eenvoudiger is.

Tabel 2 toont enkele resultaten waarbij de mediaan werd gebruikt als voorspelling. Uit de resultaten lijkt de mediaan een betere keuze te zijn voor het regressiemodel.

prune	*RULECOST*	*MAX_NB_LEAF*	nb of rules	error trainingset	error testset
false	/	3	33	54.20	127.00
true	5	3	11	89.30	128.20
false	/	5	20	67.60	125.55
true	5	5	11	89.65	126.90
false	/	10	11	91.10	118.15
true	5	10	10	99.04	126.40

Tabel 2: Voorspelling met mediaan. Invloed van de parameters op de training en test set error.

Voor de *jak2*-dataset kon Gecode aanvankelijk geen itemset vinden: dit kwam doordat de dataset enkele extreem grote waarden bevatte. Na normalisatie van de waarden, kon er echter wel een itemset gevonden worden. Wegens tijdsrestricties werden de experimenten voor deze dataset niet verder uitgevoerd in Java zelf.

5 Conclusie

Het optimalisatiecriterium voor het Gecodesysteem dat zoekt naar één itemset gebruikt de mean squared error (van de gecoverde en niet-gecoverde). Voor het regressiemodel werd een beslissingboom opgesteld waarbij de leaves de uiteindelijke itemsets zijn, waarbij de bijbehorende voorspelling gebeurt aan de hand van het gemiddelde—of de mediaan. Het zo bekomen model kan uiteindelijk nog gepruned worden, waarbij elke verwijderde regel een bepaalde winst oplevert.

De parameters zijn uiteraard sterk afhankelijk van de gegeven dataset (zo is er onder andere de beperkte grootte van de *auto93bin* dataset). De conclusies getrokken uit de experimenten zijn dus moeilijk te veralgemenen.