INF283 - Project – 1

Intro

I dette prosjektet skulle vi lage en implementasjon av decision trees, der vi lagde både learn- og predict-funksjonalitet, samt metoder for å sjekke nøyaktigheten i predictionsene. I dette dokumentet går jeg igjennom de forskjellige aspektene ved min implementasjon.

Innlesning

Dataen vi skulle bruke i oppgave fire var gitt som en tekstfil. Jeg valgte å bruke denne dataen underveis i arbeidet bla. for debugging. De første jeg gjorde var dermed å lage en funksjon for innlesning. Her brukte jeg den innebygde funksjonen open(), og leste linje for linje. Linjer som inneholdt «?» ble hoppet over. Dataen ble lagret som en list av list, senere numpy arrays

Tre-struktur og -navigering

For å representere decicsion treet valgte jeg en egen trestruktur, gitt som en egen klasse. Her fant jeg en representasjon på stacoverflow. Den ble senere modifisert, først og fremst med flere variabler. Jeg valgte å traversere treet rekursivt , der *base case*en var å nå en leaf-node. I de tilfeller returnerte jeg den ønskelige verdien(e) «oppover». Nodene ble behandlet som *mutable object*s.

Entropy og GINI

For å regne ut entropy og GINI hadde jeg to egne funksjoner. De respektive funksjonene var basert på følgende formler.

$$Entropy = \sum_{i=1}^{c}(-p_i log_2(p_i))$$

$$Gini = 1 - \sum_{i=1}^{c}(p_i)^2$$

Fremgangsmåten ble som følger. Først fant jeg alle unike targets fra attributten jeg regnet på og antallet for hver. Deretter fant jeg totalt antall items som vi regnet på. For entropy brukte jeg så en for-loop for å summere hver utregning for hver decision, p, og returnerte (-resultatet). For GINI hadde jeg også en for-loop for summeringen av hver decision, p, og returnerte (1-resultatet).

Information gain

For å finne information gain brukte jeg funksjonene for entropy og GINI, avhengig av hva som ble gitt som impurity measurement. Formlene for information gain følger.

$$IG(T,X) = Entropy(T) - Entropy(T,X)$$

$$IG(T,X) = GINI(T) - GINI(T,X)$$

Total information gain ble funnet ved å først finne det første leddet i uttrykket. Deretter brukte jeg en for-loop for å summere verdiene for hver decision. Her ble frekvensen ganget med entropy/GINI for hver forekomst av hver decision. Til slutt ble differensen returnert.

Best attribute/largest information gain

For å finne den featuren som gav størst information gain hadde jeg en funksjon som sjekket for alle ledige features, og tok den der inormation gain var størst.

ID3-algorithm

Med de definerte funksjonene kunne jeg lage desicion treet ved hjelp av ID3-algoritmen. Jeg implementerte algoritmen på følgende vis. Funksjonen mottok variablene dataset x, targets y, ledige features feature_list, impurity_measure og parent_node. Det første som skjer i funksjonen er at den sjekker om det resterende settet av items er pure. Hvis det er tilfellet så returnerer vi en leaf-node «opp». Den andre sjekken som skjer er at den sjekker om det er flere features å dele settet på. Hvis det ikke er det så returnerer den «opp» en leaf-node med labelen er den targeten det er flest av i settet. Hvis ingen av disse to tilfellene er sanne så går vi videre til resten av algoritmen. Her finner vi den attributten som det er best å splitte på, i denne iterasjonen. Så lager vi en ny node, som representerer denne iterasjonen. Noden får satt label lik den attributten vi bestemmer å splitte på. Videre lager vi barna til denne noden. Da splitter vi alle items på hver decision de representerer, og lager hvert sett, og setter hver nye child lik id3() og gir det settet med items som hører til hver decision branch. Til slutt i funksjonen returnerer vi «denne» noden. Dermed vi algoritmen gå rekursivt helt ned til vi har leafs, og returnere disse «opp» til der de ble kalt på. Der de blir lagt inn i side children-lister. Første kall på funksjonen vil til slutt returnere root-noden.

Learn

Funksjonene learn kaller på id3-funksjonen. Hvis vi ikke skal prune returnerer vi bare resultatet av id3. Hvis vi skal prune blir settet delt inn i et traning-sett og et prune-sett. Modellen blir bygget med learn-settet, så traverserer vi treet og finner en error-verdi for hver node med prune-settet og til slutt pruner vi.

Predict

Predict tar inn en item, og returnerer den targeten som er mest sannsynlig basert på modellen. Den begynner på root-noden, og beveger seg nedover ved å hele tiden velge den branchen som passer for hver attributt. Til slutt returnerer den predicted value når den kommer til et leaf, evt. label til noden hvis den kommer til en node der ingen branches matcher med itemets atributt.

Error

Error-funksjonen blir kjørt før prune-funksjonen for å gi hver node en error-value. Vi sender en og en item, og sjekker target i itemet.

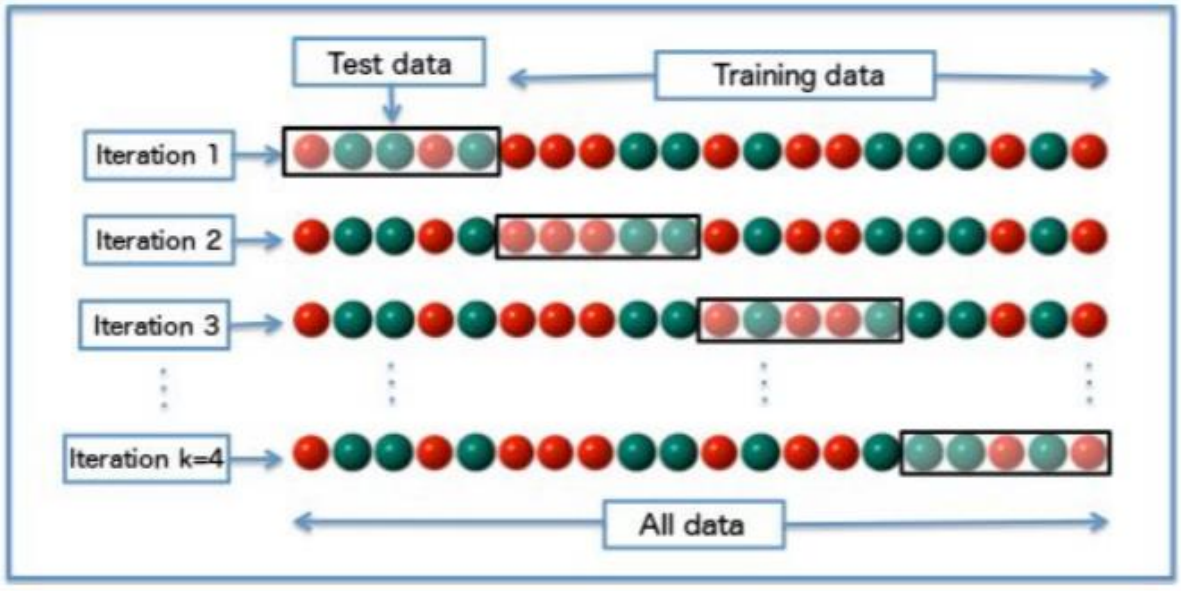
Et mulig problem som ligger i min implementasjon er at items som aldri kommer til en leafnode (dvs. training settet ikke har laget et tre som har branches som passer itemet), vil denne noden legge inn error hele veien ned til den noden der det stopper. Det gjør at barna til denne noden kan ende opp med å betydelig lavere error rate, hvis mange items alle kommer til denne noden, men ikke videre. Dette er måten jeg har valgt å lage implementasjonen på, men et alternativ kunne vært å ikke legge inn error hvis vi aldri kommer til et leaf.

Prune

Hvis pruning er «aktivert» i learing, splitter vi opp datasettet i et learing-set og et prune-set, og evt. shuffler. Vi kan velge hvor stor del av settet som skal brukes til pruning, men default er 40 %. Det er fordi standard prunestørelse pleier å ligge på mellom 20% og 40 %. Deretter kjører vi id3-algoritmen på settet for å lage en modell, før vi pruner modellen med prune-settet. Pruningen skjer rekursivt, der base-casen er at vi har kommet til en leaf-node. I det tilfellet returnerer vi «opp» nodens error. Dette brukes på noden (der kallet kom fra) over for å summere alle barnas error. Hvis barnas error er større en parent-nodens error (og barna ikke har barn igjen), så blir barna slettet, ergo *denne* noden blir gjort til leaf. Størrelsen på treet vil bli mindre hvis vi ser at det blir større prediction accuracy ved å fjerne noder. Treet kan også få forskjellig størrelse mellom kjøring på samme sett pga. shufflingen i inndataen (i main), men særlig også ved shuffling før pruningen.

Cross validation

Cross validation er en metode som returnerer et tall mellom 0 og 1, som representerer hvor godt modellen treffer basert på et treningssett. Den tar inn hvor mange biter du skal dele opp treningssettet i, samt hva slags modell den skal lage (entropy/gini, prune/ikke prune, shuffle pruning/ ikke shuffle pruning). Funksjonen teller opp antall korrekte predictions vi har for hver iterasjon. I hver enkelt iterasjon lager en modell basert på all dataen den fikk (untatt det som i denne iterasjonen er validation-data), og validerer denne modellen med det som i denne iterasjonen er validation-data (test-data). Se bilde for illustrasjon av hva som er train-/learn-data og hva som er validation-/test-data. For hver iterasjon er det dataen som er markert i grått som er validation-data, og resten er train-data.



Fra ukesoppgave 3

Main

Her kaller vi på de funksjonene vi ønsker å bruke. Innlesningen fra fil skjer også her. Det første som skjer er at vi shuffler hele settet vi har lest inn (bør kommenteres ved ut for debugging). Shufflingen gjøres fordi det er en mulighet for at datasettet vi får inn er sortert på en eller annen måte. Deretter deler vi opp datasettet i et traning-sett og et test-sett. Jeg valgte å 60/40, da det er viktig å ha et stort

nok sett for learning, men allikevel nok for testing. Training-settet brukes for å teste forskjellige modelltyper (med cross validation, entropy/gini, prune/ikke prune, shuffle pruning/ ikke shuffle pruning). Når vi har funnet ut hvilken modelltype vi skal bruke så lager vi en endelig modell (et decision-tre) med train-settet vårt, og bruker så test-settet (som hittil har vært urørt) til å teste modellen. Da får vi et tall på hvor godt modellen vår faktisk fungerer. Hvis en selv vil predicte target-value for en item manuelt, så kan dette gjøres fra main ved å bruke predict()-funksjonen.

Sammenlikning med sk learn

Jeg valgte å sammenligne min implementasjon med sk learn sin. Jeg lagde sk-treet og testet det i en egen funksjon, get_sk_tree_accuracy(). Deres implementasjon støtter ikke attributt-verdier som char/strings (kategoriske data). Derfor begynte jeg med å konvertere alle chars til deres tilsvarende tallveridi (med ord(c)). Deretter lagde jeg modellen med training-settet, og testet det med test-settet.

Da jeg skulle velge ut min beste modell falt valget på en modell basert på entropy og pruning. Det er fordi den jevnt over gav precision på 1.0 over mange tester. Verdien fra sk learn sitt tre, med defaultverdier på hvordan treet skulle lages, fikk jeg også 1.0 som accurency. Det betyr at begge modellene klarte å finne riktig verdi på alle items jeg testet dem på. Sk learn sin implementasjon har flere parametre man kan spesifisere, og er generelt mer kompleks. Derfor tror jeg at min modell hadde gitt dårligere accuracy hvis testdataen hadde vært mangelfull.

```
Cross val test, entropy                  1.0
Cross val test, gini                     0.9991139988186651
Cross val test, entropy, pruning         1.0
Cross val test, gini, pruning            0.9636739515652688
Cross val test, entropy, pruning, shuffle 1.0
Cross val test, gini, pruning, shuffle   0.9781453041937389
```

```
Accuracy for my model (entropy, pruning) 1.0

Accuracy for sk learn tree implementation 1.0
```