IIA

Compitini V
Esame: compitini/scritto, orale facoltativo

Confronto delle strategie (albero)

ALBERO MI RACCOMANDO ALBERO

Criterio	BF	UC	DF	DL	ID	Bidir
Completa?	si	si(^)	*no*	si (+)	*si*	si
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Ottimale?	si(*)	si(^)	no	no	si(*)	si

(*) se gli operatori hanno tutti lo stesso costo
(^) per costi degli archi $\geq \varepsilon > 0$
(+) per problemi per cui si conosce un limite alla profondità della soluzione (se l >d)

Best first/A* f = g + h
Greedy best first f = h (g = 0)
Costo uniforme UC f = g (h = 0)

A è completo con ε>0 come UC

Bilancio su A*

- A* è completo: discende dalla completezza di A (A* è un algoritmo A particolare)
- A* con euristica monotona è ottimale
- A* è ottimamente efficiente: a parità di euristica nessun altro algoritmo espande meno nodi (senza rinunciare a ottimalità)
- Problemi?
  - Quale euristica?
  - e ancora l'occupazione di memoria che nel caso pessimo resta $O(b^{d+1})$, causa frontiera

- Nel caso di ricerca a/su albero l'uso di un'euristica *ammissibile* è sufficiente a garantire l'ottimalità di A*
- Nel caso di ricerca su grafo serve una proprietà più forte: la consistenza (detta anche monotonicità)
  - Cerchiamo quindi condizioni per garantire che il primo *espanso* sia il migliore

Se sono monotone le consideriamo ammissibili

Beam search non completo e non ottimale: best first con k nodi più promettenti (può far perdere soluzioni)

IDA* A*+ID: ricerca in profondità con un limite dato da f (quindi aumenta a ogni interazione)
Ottimizza la memoria rispetto a A* (O(bd))

Min max: Tempo $O(b^m)$ Spazio O(m)
Con potatura α β è $O(b^{m/2})$

Hill climbing: algoritmo di ricerca locale greedy
Sceglie nodo che migliora la valutazione attuale
Migliore -> salita rapida
A caso -> stocastico
Il primo -> prima scelta

Riavvio casuale: se fallisce riparte da un punto casuale
Media 1/p ripartenze con p probabilità di successo
completo

Tempra simulata: stocastico, se migliora viene scelto
- se no (caso in cui $\Delta E = f(n') - f(n) < 0$) quel nodo viene scelto con probabilità $p = e^{\Delta E/T}$ $[0 \leq p \leq 1]$

Se la mossa peggiora molto, p si abbassa molto (inversamente proporzionale a ΔE)
T decresce man mano che l'algoritmo procede (improbabili le mosse peggiorative)
Se T decresce abbastanza lentamente -> ottimale
T determinato sperimentalmente: per valori medi di ΔE, p = 0.5

Local beam: tiene in memoria k stati, si generano i successori, si prosegue con i k migliori di questi
Beam search stocastica: k successori ma migliori con probabilità maggiore

Algoritmi **genetici**: variante della beam search stocastica, successore combinazione genitore

IIA-ML

giovedì 19 maggio 2022 14:51

Lookup table

Può rispondere solo per gli esempi presenti, altrimenti dà errore (unbiased learner)

Find-S
Prende solo esempi positivi , parte da tutto specifico e prende la prima ipotesi, poi generalizza i mismatch per le iterazioni successive

Candidate elimination
Prende tutti gli esempi. Per positivi fa find S e elimina le incongruenze da G,
!! I passaggi successivi li fa partendo da G e non da tutti ??
Per negativi parte da generale e inserisce istanze con mismatch rispetto a S

Regressione lineare

Gradiente formula: $\frac{\partial E(\mathbf{w})}{\partial w_i} = 0,$

$$\frac{\partial}{\partial w}k = 0, \frac{\partial}{\partial w}w = 1, \frac{\partial}{\partial w}w^2 = 2w$$
$$\frac{\partial (f(w))^2}{\partial w} = 2f(w)\frac{\partial (f(w))}{\partial w}$$
Der. sum = sum of der.

Gradient descent algorithm wnew= w + eta*Δw ∀wi
Δw = – gradiente di Ew

Problemi di classificazione: 1 se wTx +w0 = 0, 0 altrimenti

SLT

- Minimize *risk function* $R = \int L(d, h(\mathbf{x}))dP(\mathbf{x}, d)$ True Error Over *all* the data
- Given
  - value from teacher (d) and the probability distribution $P(\mathbf{x}, d)$
  - a loss (or cost) function, e.g. $L(h(\mathbf{x}), d) = (d - h(\mathbf{x}))^2$

Per cercare h bisogna minimizzare il rischio empirico (errore di training)

$$R_{emp} = \frac{1}{l}\sum_{p=1}^{l}(d_p - h(\boldsymbol{x}_p))^2$$

VC-dim (misura della complessità)
VC-bound: probabilità 1-∂ che R≤Remp + ϵ(1/l, VC, 1/∂)
ϵ dir prop a VC-dim, inv prop a l(nr dati), ∂(confidenza, probabilità che valga il bound)

Ridge regression

Def Error data term + Regularization/penalty term

$$Loss(h_w) = \sum_{p=1}^{l}(y_p - h_w(\boldsymbol{x}_p))^2 + \lambda \|\boldsymbol{w}\|^2 \longrightarrow \sum_i w_i^2$$

Lambda λ is called the regularization coefficient (a constant parameter or hyperparameter)

- Effect : **weight decay** (basically add $2\lambda w$ to the gradient of the Loss)

$\boldsymbol{w}_{new} = \boldsymbol{w} + eta^*\Delta\boldsymbol{w} - 2\lambda\boldsymbol{w}$

EXERCIZE: derive it

ID3

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|Sv|}{|S|} Entropy(Sv)$$

$Values(A)$ possible values for A
Sv subset of examples S for which A has value v *(weighted sum)*
Cerchiamo A con gain più alto, aka vogliamo ridurre l'entropia

Entropy (S) = – p+ log2 p+ – p–log2 p–

Assume 0log0 = 0
Log1/2 = -1

Se i due insieme sono equivalenti per nr di ipotesi + e -, entropy = 1

In caso di casi non significativi può essere necessario usare:

Def $$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Where
$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

SVM
Hard margin: non accetta errori
Vettori di supporto: quelli sul bordo del margin
Obiettivo: minimizzare |w|²/2

Training problem (primal form) :

minimize $|\boldsymbol{w}|^2/2$ (i.e. $\boldsymbol{w}^T\boldsymbol{w}$)

such that $(\boldsymbol{w}^T\boldsymbol{x}_p + b)\, y_p \geq 1$ for all $p = 1..l$

Duale serve a calcolare α che è un vettore di supporto
L'iperpiano dipende solo dai valori di supporto

Soft margin: ammette qualche errore o tolleranza di punti su margine

Primal training problem:

minimize $|\boldsymbol{w}|^2/2 + C \cdot \Sigma_p \xi_p$

such that $(\boldsymbol{w}\boldsymbol{x}_p + b)\, y_p \geq 1 - \xi_p$ and $\xi_p \geq 0$ for all p

C definito da user, low underfitting, hi overfitting
ξ indica errori o margine piccolo se positivo

Kernel

Applicazione di LBE in caso di problemi non linearmente separabili
Mantiene la complessità bassa anche in spazi grandi

Modello finale con uso di kernel in h(x)

$$h(\mathbf{x}) = sign\left(\sum_{p \in SV} \alpha_p y_p \boxed{K(\mathbf{x}_p, \mathbf{x})}\right)$$

Kernel notevoli (RBF): gaussiana con iperparametro σ, se piccolo = overfitting, grande = underfitting

possono portare l'overfitting: C, kernel, parametri del kernel, etc

K-NN
Prendo l'average dei k punti più vicini, con 1 se avg>0.5, 0 altrimenti
Curse of dimensionality: calcolo di tutte le distanze dai k-1 input, esponenziale

IA COMPLESSITà

giovedì 19 maggio 2022 14:33

Regole congiuntive:
Capacità espressiva AND, poco complesso
Candidate:
Esprime AND e OR, complessità bassa
Modelli lineari:
Complessità bassa
Se errore empirico (training) basso: *Overfitting*
VC-dim aumenta con + variabili -> *Overfitting*
Gradient descent
Troppe iteraziini -> overfitting
LBE
Complessità cresce con il numero di dim introdotte -> *overfitting*
Ridge regrssion
λ alto underfitting possibile basso *overfitting*
    Regola il valore di M (grado del polinomio)
DT
Esprimono qualsiasi f finita a valori discreti (disgiunzione di congiunzioni)
Complessi -> prone to *overfitting*
    Non si può evitare, si ricorre a pruning o a fermare la ricerca prima
    (non troppo prima, no rules on that)
    Pruning basato su osservazione di tr e vl

SLT
Errore training basso -> *overfitting*
Tanti dati abbaassano la complessità
Vc dim inv prop a e traing
Epsilon alto -> *overfitting*

SVM
VC-dim inv prop a margine
Margine più stretto -> tendenza a overfitting

<u>Soft margin:</u>
C basso -> *underfitting* (margine elevato)
C alto -> *overfitting* (pochi errori considerati)
C=0 vuol dire che posso fare 100% errori
Hard margin: c alto errori ->0
<u>Kernel</u>
Regolarizza la lbe -> abbassa la complessità
<u>RBF</u>
σ piccolo -> *overfitting*

Cose che possono fare overfitting: c, kernel, parametri del kernel
K-NN
K = 1 *overfitting*
K = l *underfitting*
esponenziale