ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

Aprendizagem a partir de Observações

Nomes:	Nº USP:
Filipe Hilário Muler	5178043
Frederico Uemura Pimenta	5173607
Rafael Galucci Pereira Passos	5176062

PCS2428
Prof. Anna Helena Reali Costa

Sumário

1 Formas de Aprendizagem

Um agente de aprendizagem contém duas partes:

- O elemento de desempenho – responsável por decidir o próximo passo do agente.
- E o elemento de aprendizagem – que modifica, de alguma maneira, o primeiro para que este tome decisões melhores.

O foco deste documento recai sobre o segundo dos elementos. Existem 4 aspectos importantes para o projeto do elemento de aprendizagem:

- Os componentes do elemento de desempenho que deve ser aprendido.
- A realimentação usada para aprender os componentes citados acima.
- A representação usada nos componentes
- Existência de conhecimento a priori.

Os componentes existentes para o elemento de desempenho podem ser de 6 formas:

- Um mapeamento direto entre condição e ação – esta no estado "2" então vai para o lado esquerdo.
- Deduzir propriedades relevantes do mundo com base numa seqüência de percepções. O agente mantém um estado com base na seqüência de percepções passadas.
- Detêm informações do mundo e como suas ações podem afeta-lo – é o caso de procura e planejamento, por exemplo.
- Pode-se usar uma função de utilidade onde ela poderia classificar as opções que se pode tomar num instante para mudanças de estado.
-
- Objetivos podem listar conjuntos de estados podem maximizar a utilidade do agente.

Com relação ao tipo de realimentação disponível para a aprendizagem, pode-se classificá-la em 3 tipos, todos descritos mais adiante. A terceira característica dos elementos de aprendizagem diz respeito à forma como as informações são representadas. Polinômios para as funções de utilidade, sentenças lógicas proposicionais, discutido em maiores detalhes no documento, lógica de primeira ordem etc. Um último aspecto importante sobre o elemento de desempenho diz respeito à existência de conhecimento a

priori, como o que o agente esta tentando aprender, o que normalmente não é o caso, inclusive nos exemplos estudados aqui.

1.1 Supervisionada

A aprendizagem supervisionada diz respeito ao aprendizado de uma função que se deseja conhecer através do conhecimento de alguns pares de entradas e saídas. Com base nestes "exemplos" para o treinamento, obtém-se uma aproximação para a função dos pontos conhecidos, extrapolando, assim, o comportamento dela para outros pontos.

1.2 Não-Supervisionada

No aprendizado não-supervisionado não são fornecidas saídas para as entradas dos exemplos. Com a ausência de qualquer informação como parâmetro, o aprendizado não-supervisionado puro, não há como o agente decidir o que é uma ação adequada ou um estado desejado. Obviamente este não é o caso normalmente utilizado normalmente.

1.3 Por Reforço

O aprendizado por reforço se caracteriza pela inexistência da saída nos exemplos fornecidos, ou seja, não se da uma indicação de como o agente deve responder ao estímulo. Na verdade, outras características darão um indicativo de como o agente deve se comportar. É o caso de quando uma recompensa ao fim de uma ação indica se tratar de um comportamento desejado.

2 Aprendizagem indutiva

Baseadas na aprendizagem supervisionada, neste caso são alimentadas pares (x, f(x)), sendo que cada um deles é chamado de um "exemplo". Com base nestes dados, a inferência indutiva pura do método utilizado no elemento de aprendizagem retornará uma função h(x) que aproxima f.

Esta função h é chamada de hipótese. A adequação aos exemplos dados, e por conseqüência aos pontos não incluídos no conjunto de exemplos de treinamento, é o que se chama de generalização e no seu caso extremo obtêm-se a própria função f, situação na qual podemos dizer que o "problema da indução" foi resolvido por completo, extrapolando todos os pontos dela. Obviamente que o grau de qualidade das hipóteses encontradas

depende do espaço utilizado para aproximar f. Como na figura de exemplo a seguir, uma reta ou uma função senoidal podem ser usadas para regredir os pontos dados.

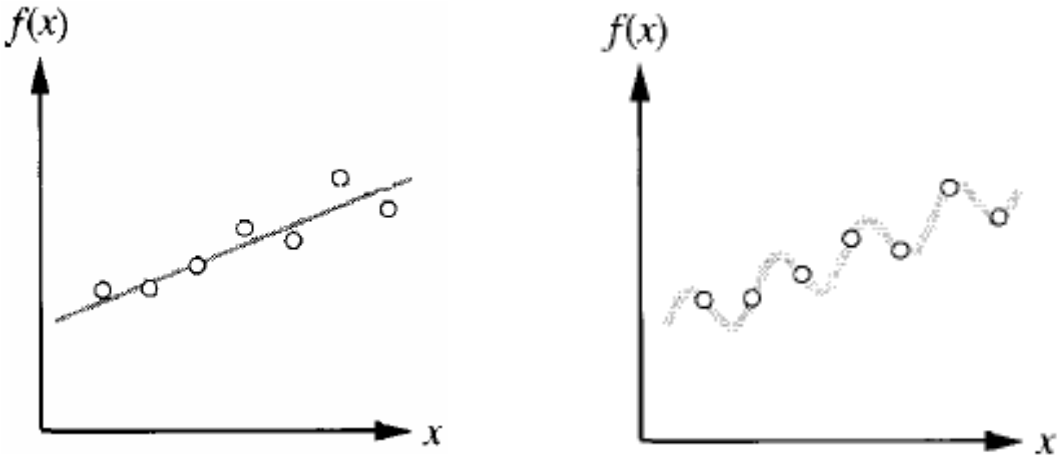

Figura 1: Adequação do espaço de hipóteses.

Quando há múltiplas alternativas para um mesmo conjunto de exemplos de treinamento, deve-se ter optar por algumas delas. A regra intitulada Navalha de Ockham nós diz para preferirmos a hipótese mais simples dentre aquelas que se adéqüem de forma consistente aos dados.

É importante ressaltar que a seleção de um espaço adequado de funções pode não só permitir diferentes graus de complexidade da resposta, como pode permitir, ou não, o encontro da função exata para o aprendizado. Assim, quando a função f se encontra no espaço usado para o treinamento, dizemos que o aprendizado é realizável. Caso contrário, dizemos que ela é irrealizável. Para se evitar que este seja o caso, costuma-se usar alguma informação prévia que se tenha sobre a função que se deseja aprender para poder garantir um espaço que a contenha.

Além disso, um espaço mais simples pode assegurar soluções mais rápidas para problemas computacionalmente de difícil obtenção com o modelo ou função completa, ou mesmo permitir se encontrar uma hipótese mais depressa, do que em espaços mais complexos.

Portanto, no restante do documento, assim como na principal referência em que se baseou para sua escrita, serão descritos exemplos e algoritmos usados principalmente para representações simples de problemas, como é o caso da lógica proposicional.

3 Aprendizagem em Árvores de Decisão

A árvore de decisão indutiva é um dos métodos mais amplamente utilizados e práticos de inferência indutiva. Ele tem uma forma simples e é fácil de se implementar. Abaixo daremos exemplos de árvores de decisão em aprendizado indutivo.

3.1 Árvores de decisão como elemento de desempenho

As arvores de decisão tomam como entrada um conjunto de atributos que descrevem um objeto ou situação e retornam uma decisão, que seria uma saída prevista de acordo com a entrada.

Os valores de entrada e saída podem ser discretos ou contínuos. A aprendizagem de uma função contínua é chamada de regressão e de uma com valores discretos classificação. Cada nó da arvore especifica um teste de algum atributo da instancia e as saídas representam possíveis valores que esse atributo pode vir a tomar. A árvore começa a ser percorrida da raiz até chegar às suas folhas.

Como exemplo, vamos apresentar uma árvore de decisão que terá como objetivo aprender uma definição para o predicativo de objeto *VaiEsperar*. Abaixo temos os atributos juntos com as suas definições:

1. Alternativa: Se há um restaurante alternativo apropriado por perto.
2. Bar: Caso o restaurante tenha um bar confortável em que se possa esperar.
3. Sex/Sáb: Verdadeiro às sextas e sábados.
4. Faminto: Se a fome é grande.
5. Clientes: Quantidade de pessoas. (Valores: Nenhum, Alguns e Cheio)
6. Preço: Faixa de preços do restaurante ($,$$,$$$).
7. Chovendo: Caso esteja chovendo lá fora.
8. Reserva: Caso haja uma reserva feita.
9. Tipo: Tipo do restaurante (francês, italiano, tailandês ou só serve hambúrguer).
10. EsperaEstimada: A espera estimada pelo gerente (de 0 a 10 minutos, de 10 a 30 minutos, de 30 a 60, >60).

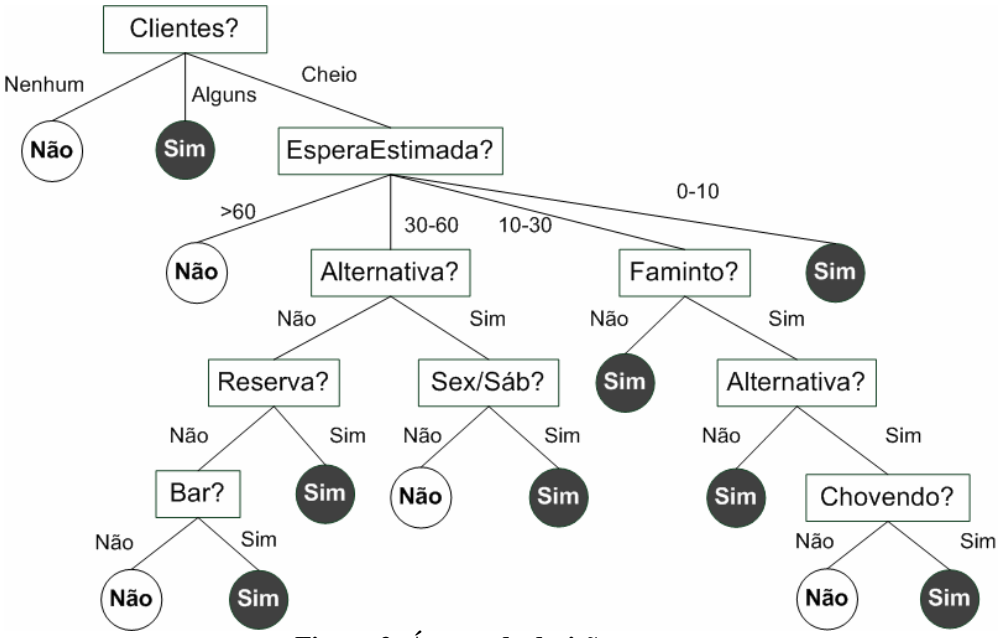

Figura 2: Árvore de decisão

3.2 Expressividade de árvores de decisão

Traduzindo em termos lógicos a hipótese da árvore de decisão obtida na exemplo, temos:

$$\forall s \; VaiEsperar(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \ldots \vee P_n(s))$$

Onde $P_i(s)$ é uma conjunção de testes que corresponde o caminho da raiz até uma folha com resultado positivo. Apesar de parecer uma sentença de primeira ordem, ela na verdade é uma sentença proposicional de uma variável e todos predicados unários.

Como as árvores de decisões se enquadram expressivamente nas classes de linguagens proposicionais, podemos ter qualquer função booleana escrita numa árvore de decisão. Mas para nem todos os problemas ela pode ser a melhor decisão, já que em alguns casos a árvore se torna exponencialmente grande. Como exemplo, temos o caso de uma função de paridade[1] e a função maioria[2].

Resumindo, árvores de decisão são boas para alguns tipos de decisão e não são boas para outros. Para exemplificar, podemos tomar a tabela da verdade com 2^n linhas, já que cada entrada possui n atributos. Se são necessários 2^n para definir a função, então teremos 2^{2^n} funções diferentes de n atributos.

3.3 Induzindo árvores de decisão a partir de exemplos

O problema de se encontrar uma árvore de decisão booleana, onde cada entrada é aceita ou rejeitada, consiste em se passar um vetor de entrada com uma lista de atributos, uma tupla. Para esse conjunto, com seus respectivos valores, a árvore retorna verdadeiro ou falso.

Em outras palavras, os exemplos do conjunto de treinamento serão classificados como verdadeiros, aqueles que retornam valor verdadeiro, ou falsos, quando retornam valor falso. Um algoritmo que retorna uma árvore de pequena profundidade consiste em:

- Se existe exemplos negativos e positivos, escolha o melhor atributo para dividi-los, ou seja, o atributo que melhor filtra os exemplos. Detalhes da escolha serão dados adiante.
- Se todos os exemplos restantes forem positivos, ou negativos, então já se pode tomar uma decisão, o exemplo que cair nesta folha será verdadeiro, ou falso respectivamente.

[1] Função Paridade: retorna 1 se e somente se um número par de entradas é igual a 1.
[2] Função maioria: retorna 1 se mais da metade de suas entradas é igual a 1.

- Se não resta nenhum exemplo, então não se pode tomar uma decisão com base nos exemplos usados para treinamento, toma-se uma decisão padrão.
- Quando se acabam os atributos, mas existem exemplos negativos e positivos num mesmo ramo dizemos que há ruído nos dados, pois há dois exemplos com os mesmos dados, atributos, mas com classificações distintas, ou não há exemplos suficientes para inferir todos os possíveis casos ou trata-se mesmo de um caso de indeterminismo.

Como ilustração, expomos os dados de exemplos na tabela abaixo para o mesmo problema do restaurante já apresentado anteriormente. A árvore que segue foi gerada com o algoritmo apresentado.

Exemplo	Atributos										Objetivo Esperar
	Alternativa	Bar	Sexta-feira	Com fome	Clientes	Preço	chovendo	Reserva	Tipo	Estimativa	
X1	*Sim*	*Não*	*Não*	*Sim*	*Alguns*	*$$$*	*Não*	*Sim*	*Francesa*	*0-10*	*Sim*
X2	*Sim*	*Não*	*Não*	*Sim*	*Cheio*	*$*	*Não*	*Não*	*Tailandesa*	*30-60*	*Não*
X3	*Não*	*Sim*	*Não*	*Não*	*Alguns*	*$*	*Não*	*Não*	*Fast-food*	*0-10*	*Sim*
X4	*Sim*	*Não*	*Sim*	*Sim*	*Cheio*	*$*	*Sim*	*Não*	*Tailandesa*	*10-30*	*Sim*
X5	*Sim*	*Não*	*Sim*	*Não*	*Cheio*	*$$$*	*Não*	*Sim*	*Francesa*	*>60*	*Não*
X6	*Não*	*Sim*	*Não*	*Sim*	*Alguns*	*$$*	*Sim*	*Sim*	*Italiana*	*0-10*	*Sim*
X7	*Não*	*Sim*	*Não*	*Não*	*Nenhum*	*$*	*Sim*	*Não*	*Fast-food*	*0-10*	*Não*
X8	*Não*	*Não*	*Não*	*Sim*	*Alguns*	*$$*	*Sim*	*Sim*	*Tailandesa*	*0-10*	*Sim*
X9	*Não*	*Sim*	*Sim*	*Não*	*Cheio*	*$*	*Sim*	*Não*	*Fast-food*	*>60*	*Não*
X10	*Sim*	*Sim*	*Sim*	*Sim*	*Cheio*	*$$$*	*Não*	*Sim*	*Italiana*	*10-30*	*Não*
X11	*Não*	*Não*	*Não*	*Não*	*Nenhum*	*$*	*Não*	*Não*	*Tailandesa*	*0-10*	*Não*
X12	*Sim*	*Sim*	*Sim*	*Sim*	*Cheio*	*$*	*Não*	*Não*	*Fast-food*	*30-60*	*Sim*

Figura 3: Tabela de exemplos.

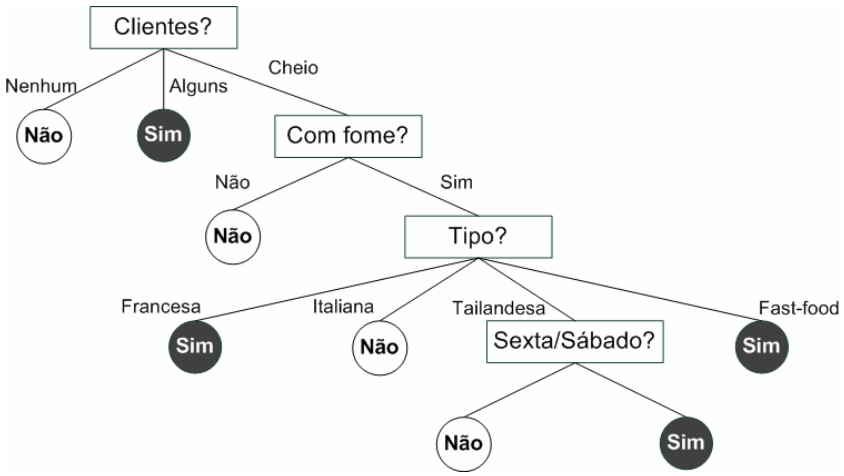

Figura 4: Árvore resultante da aplicação do algoritmo.

3.4 Escolha de testes e atributos

Ao montar a árvore de decisão, busca-se selecionar atributos que minimizem a profundidade da árvore. Um atributo perfeito dividiria os exemplos em conjuntos positivos e negativos apenas. Para realizar a escolha entre os diversos atributos, pode-se utilizar como medida de comparação a quantidade de informações fornecidas por cada atributo. Essa quantidade de informações pode ser medida em bits, dado por:

$$I\left(P(v_1), \ldots, P(v_n)\right) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$$

Onde P(vi) corresponde à probabilidade de ocorrência de vi.

Em uma árvore de decisão, a quantidade de informações contidas em uma resposta correta é:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Após o teste de um único atributo qualquer A, é necessário descobrir a quantidade de informações restantes para classificar o exemplo. Se A divide o conjunto de treinamento em v subconjuntos (diferentes valores possíveis para A) e cada subconjunto Ei tem pi exemplos positivos e ni exemplos negativos, o restante de informações necessárias para classificar o exemplo após o teste de A é dado por:

$$Restante\,(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

Deste modo, o ganho de informações para um atributo pode ser representado por:

$$Ganho(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - Restante\,(A)$$

3.5 Avaliação do desempenho do algoritmo de aprendizagem

O desempenho do algoritmo de aprendizagem é analisado preparando dois tipos de conjuntos disjuntos: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento é responsável pela geração da hipótese que procura aproximar a função de saída. Uma vez obtida a hipótese h a partir do conjunto de treinamento, mede-se a porcentagem de acerto da hipótese para o conjunto de teste. Este processo pode ser repetido diversas vezes para conjuntos de treinamento e teste com diferentes composições de exemplos e tamanhos. O resultante deste processo é um conjunto de dados que pode ser

representado por uma curva de aprendizado, que mostra a eficiência do algoritmo com relação ao tamanho do conjunto de treinamento.

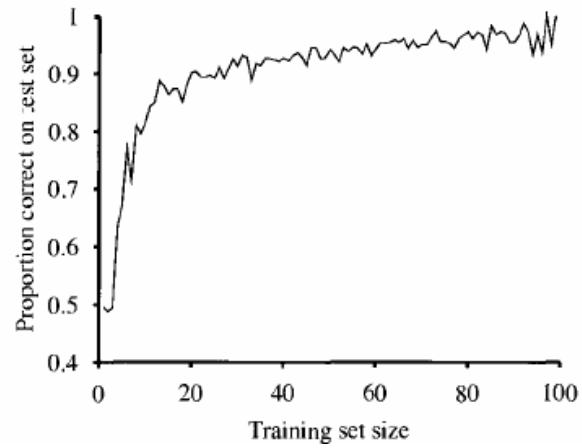

Figura 5: Curva de aprendizado

Um problema que pode decorrer desta abordagem é a ocorrência de espionagem do conjunto de teste, isto é: para cada novo conjunto de treinamento, utilizam-se dados do desempenho da hipótese anterior sobre o conjunto de testes, que por sua vez é reutilizado. O resultado disso é que as hipóteses geradas são otimizadas para responder a um teste específico, não para generalizar a função de saída.

3.6 Ruído e superadaptação

Mesmo quando não existem informações para realizar a decisão, é provável que o algoritmo de aprendizagem de árvore de decisão encontre uma árvore consistente para os exemplos, utilizando atributos irrelevantes, o que gera ruídos. Um exemplo de ruído seria considerar o atributo *Cor do Dado* em exemplo de lançamento de dados. Quanto maior a quantidade de dados, maior a probabilidade de serem inseridos ruídos na árvore de decisão. Outro problema que pode acontecer é a superadaptação, na qual o algoritmo encontra padrões sem significado para os dados. Métodos utilizados para eliminar a superadaptação incluem a Poda χ^2 e a Validação Cruzada.

4 Aprendizagem por Agrupamento

4.1 Aprendizagem por agrupamento

A aprendizagem indutiva pode tornar-se mais robusta através da utilização de mais de uma hipótese para realizar previsões. Um modo de realizar tal aprimoramento é através da aplicação de métodos de aprendizagem por agrupamento, que consiste em selecionar um agrupamento de hipóteses a partir do espaço de hipóteses.

Ao selecionar este grupo de hipóteses, a classificação de um exemplo é feita por meio de um sistema de votação por maioria simples. A justificativa da maior robustez deste método é que para um exemplo ser classificado incorretamente é necessário que a maioria das hipóteses que constituem o agrupamento o classifique de forma incorreta. Vale notar que para obter tal ganho na robustez é necessário que as hipóteses selecionadas tenham algum grau de diferença entre si, para evitar que os erros induzidos pelos exemplos de treinamento sejam os mesmos.

O agrupamento de hipóteses pode ser estendido de forma a construir conjuntos de agrupamentos, cada um desses conjuntos podendo ser visto como sendo uma hipótese por si própria.

4.2 Aceleração

Um método de agrupamento é o da aceleração, que consiste na utilização de conjuntos de treinamento ponderados de forma a gerar hipóteses que priorizem a influência dos exemplos de treinamento corretamente classificados. Em conjunto de treinamento ponderado, cada exemplo possui um peso w_j associado a si. Este peso representa sua influência sobre a geração da hipótese.

O algoritmo de aceleração inicia a partir do conjunto de treinamento inicial, atribuindo $w_j=1$ para todos os exemplos do conjunto, a partir do qual é gerada a primeira hipótese h1. A cada iteração, observa-se a classificação feita pela hipótese gerada, diminuindo os pesos dos exemplos classificados corretamente e aumentando o peso dos classificados de forma incorreta. A idéia é que nas próximas iterações, sejam geradas hipóteses que priorizem estes exemplos classificados incorretamente para que a ao final se obtenha um conjunto expressivo de hipóteses, permitindo uma melhor taxa de acerto na classificação.

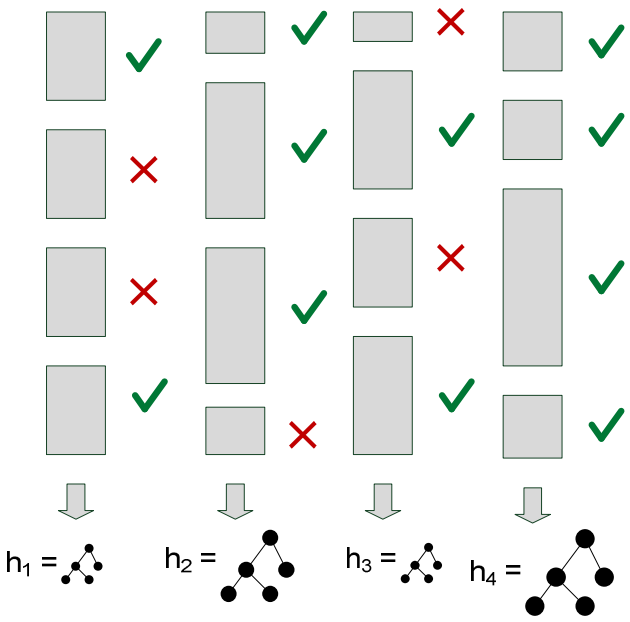

Figura 6: Funcionamento do algoritmo de aceleração

Na figura anterior pode-se observar o funcionamento do algoritmo. O tamanho de cada retângulo representa o peso do exemplo na geração da árvore e os tamanhos das árvores na parte inferior da imagem representam o peso da hipótese no conjunto final.

5 Teoria da aprendizagem computacional

5.1 Teoria da aprendizagem computacional

A justificativa para o funcionamento da aprendizagem é baseada no princípio de que se uma hipótese que esteja bastante errada será facilmente identificada, pois falhará na classificação após a execução de apenas alguns exemplos. Quando a hipótese consegue obter um comportamento consistente para um grande número de exemplos, isto é, consegue prever corretamente a saída para um grande número de exemplos de teste, esta hipótese é chamada de Provavelmente Aproximadamente Correta e um algoritmo que retorne hipóteses deste tipo é chamado de algoritmo de aprendizagem PAC. Para que a hipótese seja aproximadamente correta tanto para o conjunto de treinamento – que a gera – como para o conjunto de teste, supõe-se que ambos são extraídos de forma aleatória e independente da população de exemplos (Hipótese de Estacionaridade).

5.2 Complexidade da Amostra

Complexidade da amostra diz respeito ao número de exemplos necessários para compor o conjunto de treinamento, em função do erro da função h com relação à f que é visado e da probabilidade de se obter uma hipótese consistente pertencente ao espaço de

hipóteses não adequadas (Hruim). É possível deduzir esse valor, chegando na seguinte expressão:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$$

Onde:

N: número de exemplos necessários

ϵ: erro de h em relação à f

H: conjunto de hipóteses possíveis

δ: probabilidade de se obter uma hipótese consistente pertencente ao conjunto de hipóteses ruins (isto é, que não generalizam bem).

Para o caso em que H é o conjunto de todas as funções booleanas, a complexidade da amostra tem crescimento exponencial (2n). Para evitar esse crescimento, pode-se limitar o espaço de funções a linguagens mais restritas, uma vez que nem sempre é necessária a utilização de todo o poder de expressão proporcionado por elas. Uma forma de aprendizagem utilizando linguagem mais restrita é a aprendizagem por listas de decisão.

5.3 Aprendizagem em listas de decisão

A lista de decisão pode ser descrita como uma expressão lógica em uma forma restritiva. Ela é composta por testes, cada um uma conjunção de literais. Se um teste tem sucesso, ele especifica o valor a ser retornado. Caso falhe, continua com o próximo teste na lista. As listas lembram árvores de decisão, mas sua estrutura é mais simples.

Abaixo temos um exemplo com o caso do restaurante:

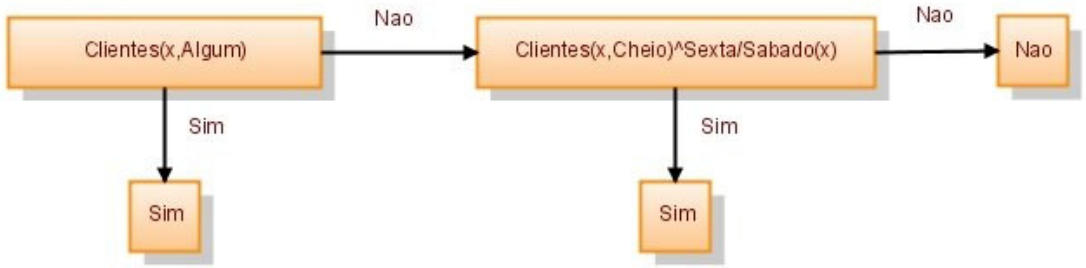

Figura 7: Lista de decisão para o problema do restaurante

Temos que se restringirmos o numero de testes para no máximo k literais, o algoritmo de aprendizagem poderá generalizar com sucesso a partir de um pequeno número de exemplos. Essa é a linguagem k-LD.

O k-LD inclui no seu subconjunto a linguagem k-AD, que é o conjunto de todas as árvores de decisão de profundidade no máximo k.

Temos que o numero de exemplos necessários para a aprendizagem de PAC de função k-LD é polinomial em n:

$$N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O(n^k \log_2(n^k))\right)$$

Abaixo, temos a curva de aprendizagem para o algoritmo de lista de decisão e o de árvore de decisão para comparação:

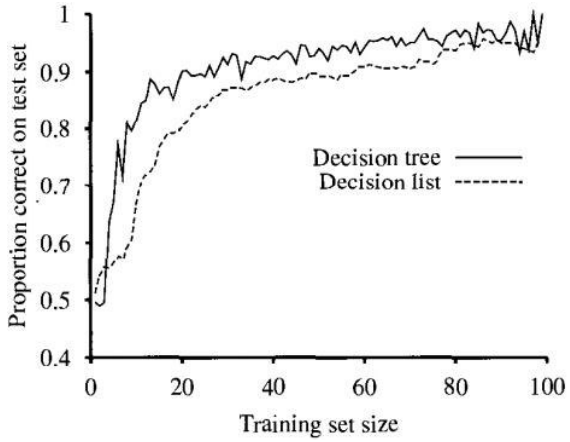

Figura 8: Comparação entre o algoritmo de aprendizagem de árvore de decisões e lista de decisões

6 Conclusão

A aprendizagem indutiva pura é geralmente difícil, pois há uma série de problemas que podem prejudicar a geração de uma hipótese adequada. Entre elas podemos citar:

- Árvores de decisão complexas, com crescimento exponencial com o problema.
- Ruído nos dados, que comprometem soluções determinísticas, graças a existências de dados, exemplos, muitas vezes inconsistentes.
- Superadaptação, gerada por atributos desnecessários que geram árvores de decisões que distorcidas, com padrões irrelevantes ou inexistentes na prática.

7 Referências

- Russel, Stuart; Norvig, Peter. Inteligencia Artificial. 2.ed. Ed. Campos, 2004. 1021p.