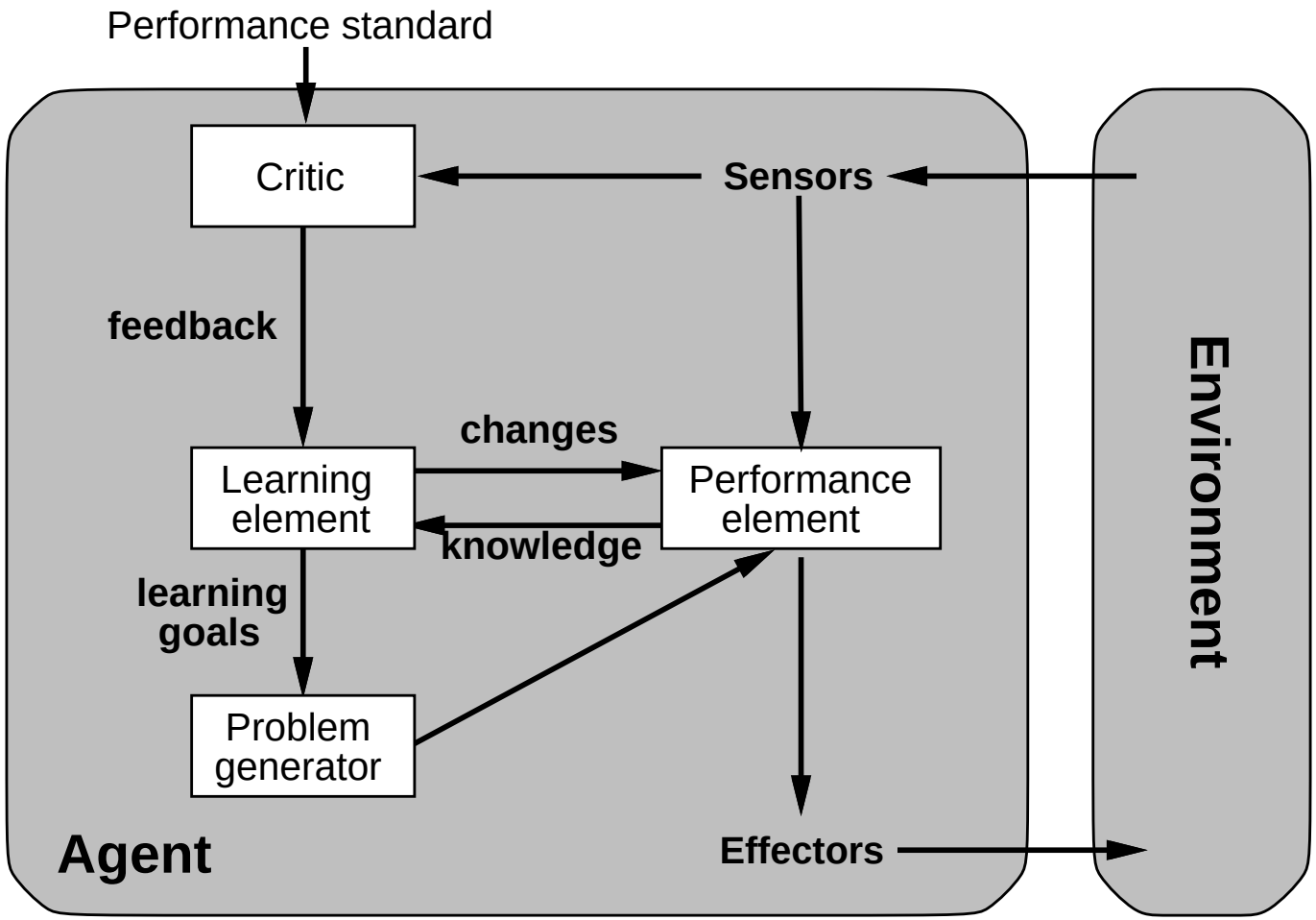
Aprendizagem de Máquina

April 30, 2019

Componentes de um Agente que Aprende

(Fig. 2.15, AIMA book, 3rd ed., page 55)



Aprendendo através de observações

- Projeto de um agente inteligente é influenciado por 4 fatores:
  - identificação de componentes a melhorar
  - representação usada para os dados e para os componentes (lógica?)
  - tipo de retro-alimentação (*feedback*) disponível.
  - informação anterior (conhecimento prévio ou *prior knowledge*) disponível.

Aprendendo através de observações

- Componentes que podem ser aprendidos:
  - função que mapeia as condições do estado corrente para as ações.
  - meios de inferir propriedades relevantes do ambiente através das percepções.
  - info sobre modificações no ambiente
  - info sobre os resultados de possíveis ações
  - info de utilidade do resultado
  - info sobre valores de ações (prioridades) que indiquem o interesse naquela determinada ação em determinado estado.
  - Objetivos que descrevem classes de estados que maximizem a utilidade.

Aprendizagem: definiçao

- Um agente "aprende" se melhora o seu desempenho em tarefas futuras após fazer observações sobre o mundo passado ou atual."

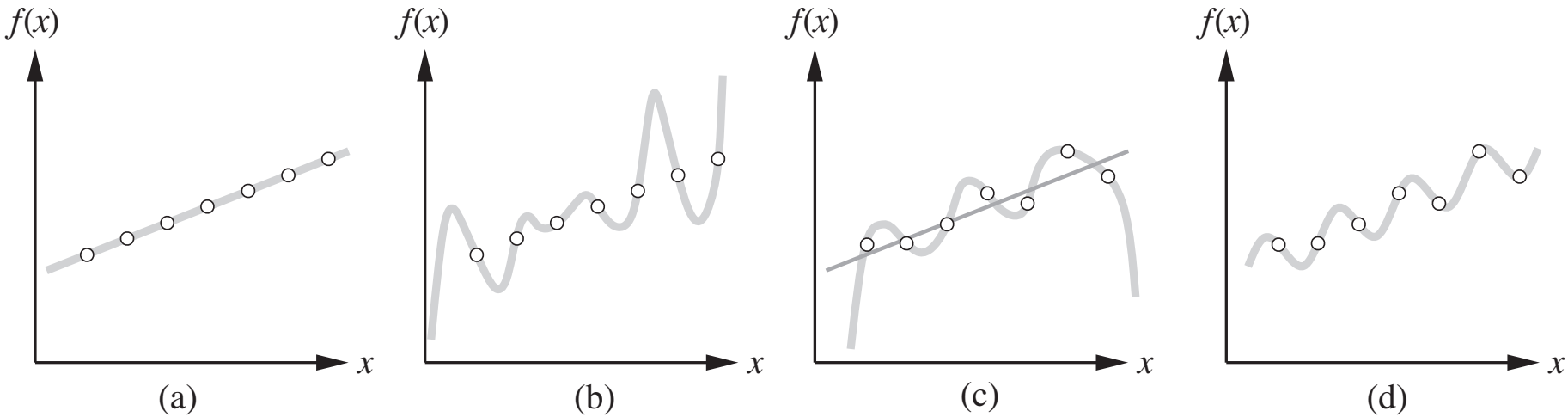
Aprendendo através de observações

- Representação dos componentes: pode ser feita utilizando qq esquema estudado.
- Retro-alimentação:
  - **aprendizagem supervisionada:** dada uma coleção de pares entrada-saída, aprende uma função que prevê a saída de novas entradas.
  - **reinforcement**: agente recebe alguma avaliação de sua ação (penalização ou recompensa). Utiliza esta informação para "julgar" ações futuras.
  - **aprendizagem não supervisionada**: agente aprende padrões sobre as entradas mesmo sem saber os valores de saída.
- Conhecimento prévio (*background/prior knowledge*): descrição das observações, necessária para melhorar a aprendizagem.

Aprendizagem Indutiva

- Em aprendizagem supervisionada o elemento de aprendizagem tem o valor correto ou aproximado da função das entradas.
- Modifica a representação da função para que esta se torne análoga à info fornecida por retro-alimentação.
- **Exemplo**: par $(x, f(x))$, onde x é entrada é f(x) é saída.
- **Inferência puramente indutiva** (ou simplesmente indução): dado um conj de exemplos de f, retorna uma função h (**hipótese**) que aproxima f.
- **Bias**: preferência por uma ou outra hipótese.

Aprendizagem Indutiva



Aprendizagem Indutiva

global *examples* $\leftarrow \{\}$

---

function REFLEX-PERFORMANCE-ELEMENT(*percept*) **returns** an action

if (*percept*, *a*) **in** *examples* **then return** *a*
else
h $\leftarrow$ INDUCE(*examples*)
return *h*(*percept*)

---

procedure REFLEX-LEARNING-ELEMENT(*percept*, *action*)
inputs: *percept*, feedback percept
action, feedback action

examples $\leftarrow$ *examples* $\cup \{($ *percept*, *action*$)\}$

Aprendizagem Indutiva

- Algoritmo atualiza var global *examples* e lista de pares *percepção, ação*.
- Percepção pode ser uma situação no jogo de xadrez.
- Ação: melhor jogada de acordo com um grande mestre enxadrista.
- Se o agente percebe uma situação q já tenha visto antes, executa a ação correspondente.
- Caso contrário, utiliza o algoritmo de aprendizagem INDUCE sobre exemplos que viu até então.
- INDUCE retorna uma hipótese h q é usada para escolher uma ação.

Aprendizagem Indutiva

- Alternativa: **aprendizagem incremental**. Agente tenta atualizar a hipótese anterior sempre q um novo exemplo aparece, sem precisar induzir sobre *todos* os exemplos a cada nova previsão.
- Pode tb receber retro-alimentação (*feedback*) sobre a qualidade das ações escolhidas.
- Forma em que hipóteses são representadas: livre.
- Algoritmos de aprendizagem: mais uma vez, lógica!
- Pelo menos duas abordagens para aprender sentenças lógicas: **árvores de decisão** e **inductive logic programming** (mais geral, menos eficiente).
- Problema: representação da função utilizada para a aprendizagem. É "representável" na linguagem? É eficiente (tratável)?

Aprendizagem Indutiva

- Lógica de primeira ordem: tempo de computação e número de exemplos necessários para aprender um "bom" conjunto de sentenças.
- "Bom" conj de sentenças: prevê corretamente experências futuras e reflete corretamente experiências passadas.
- Problema: como é que podemos saber se um algoritmo de aprendizagem está produzindo uma teoria q prevê (classifica) corretamente observações futuras?

Árvores de Decisão

- Simples e fácil de implementar.
- Recebe como entrada um objeto ou situação descrita por um conj de propriedades (atributos ou variáveis) e produz uma resposta "sim" ou "não". Representam funções booleanas.
- Exemplo: esperar por uma mesa num restaurante.
- Objetivo: aprender a definição do predicado "VouEsperar", com a definição expressa por uma árvore de decisão.

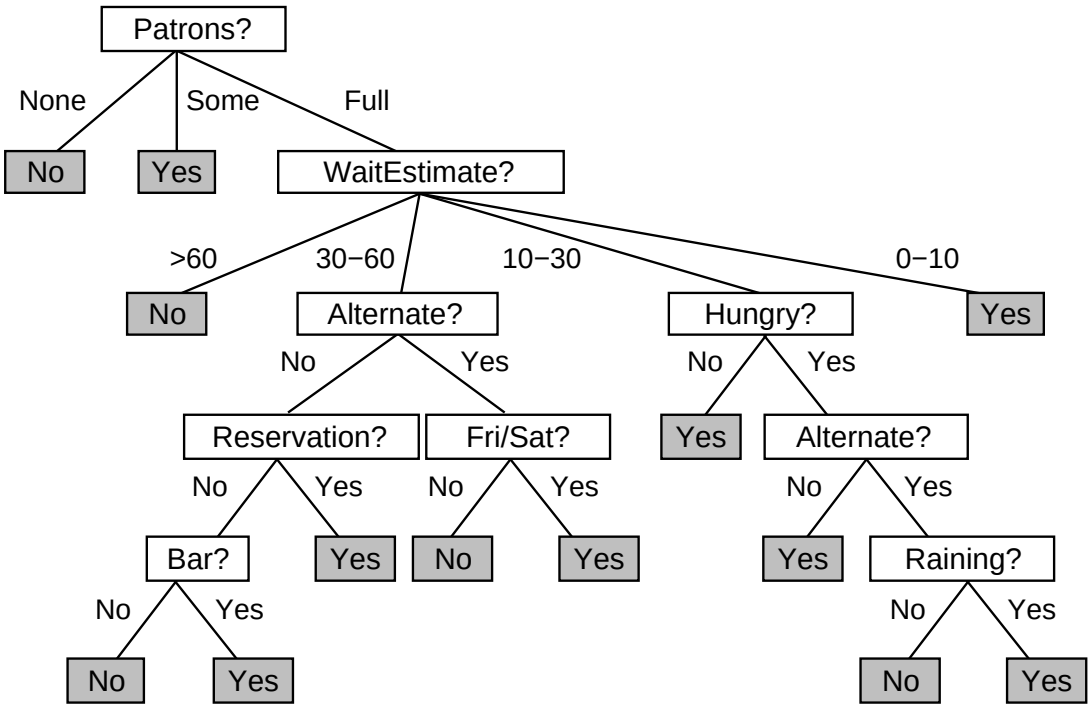
Árvores de Decisão

- Decidimos as propriedades ou atributos (p eqto, pois isto poderia ser decidido pelo algoritmo de aprendizagem):
  - Alternativo: algum restaurante alternativo perto?
  - Bar: restaurante tem uma área de espera?
  - Sex/Sab: V se for sexta ou sábado.
  - ComFome: estamos com fome?
  - Clientes: número de pessoas no restaurante (Nenhuma, Algumas, Cheio).
  - Preço: \$, \$\$, \$\$\$.
  - Chovendo: chovendo do lado de fora.
  - Reserva: temos reserva?
  - Tipo: Francês, Italiano etc.
  - EsperaEstimada: 0–10min, 10–30, 30–60, > 60.
  - Variável de interesse (classe): willWait (dados os valores das outras variáveis, o cliente espera ou vai embora?)

Árvores de Decisão

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Árvores de Decisão



Árvores de Decisão

- Em lógica:
  $\forall\ r\ Clientes(r, Cheio) \wedge EsperaEstimada(r, 10-30) \wedge \neg ComFome(r, N) \Rightarrow VouEsperar(r)$
- Árvores de decisão não conseguem representar testes sobre 2 ou mais objetos diferentes (objetos devem ser "ground").
- Limitações em representação.
- Qq função booleana pode ser representada como uma árvore de decisão.
- No entanto, representação em árvores de decisão deve ser + compacta, pq tabelas verdade têm crescimento exponencial.
  - Por exemplo, com as 10 variáveis do nosso exemplo (assumindo que todas têm somente dois valores), o número de funções booleanas é equivalente a $2^{2^{10}} \approx 10^{308}$!

Árvores de Decisão

- **Exemplos**: valores dos atributos + valor do predicado desejado (variável de classe).
- **Classificação** do exemplo: valor do predicado desejado (classe).
- qdo valor é verdadeiro, exemplo **positivo**. Caso contrário, é um exemplo **negativo**.
- conj completo de exemplos: **conjunto de treinamento**.

Árvores de Decisão

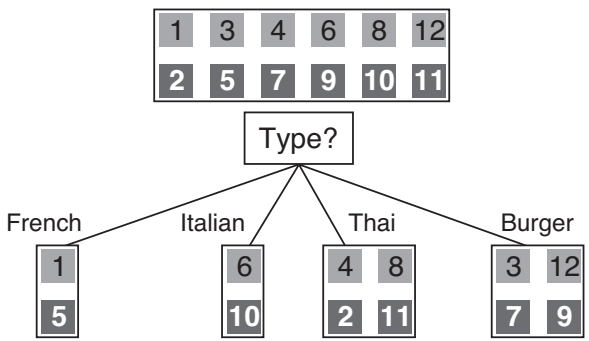
- Como induzir uma árvore de decisão através de exemplos?
- Cada exemplo pode ser um caminho diferente na árvore.
- limitação de representação, não deixa extrair nenhum outro padrão de informação além daquele descrito pelos exemplos já conhecidos.
- Extrair um padrão significa descrever um grande número de casos de forma concisa.
- Princípio geral de aprendizagem indutiva: **Ockham's razor**. "A hipótese mais provável é a mais simples e consistente com todas as observações".
- encontrar a menor árvore de decisão é um problema intratável.
- heurísticas podem ajudar.

Aprendizagem Indutiva

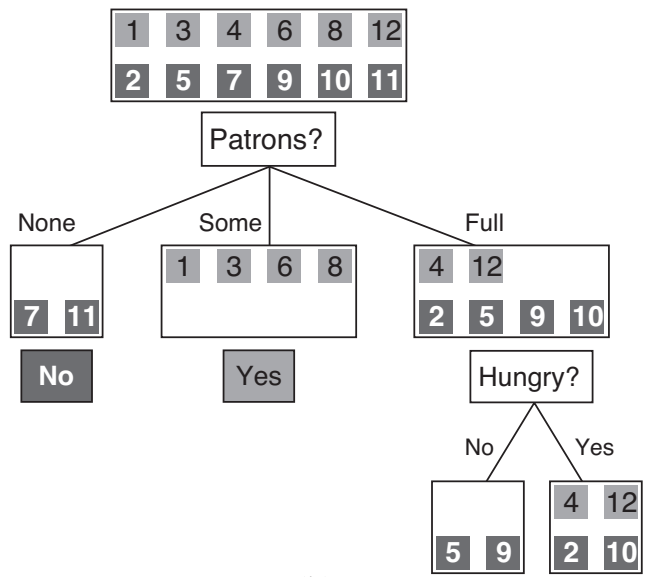
- idéia básica do algoritmo: testar os atributos "mais importantes" primeiro.
- O que é um atributo "mais importante"? É aquele que influencia mais a classificação do exemplo.
- Exemplo: 12 conjuntos de treinamento, separados em exemplos positivos e negativos.
- *Clientes* é um atributo importante: se valor igual a Nenhuma ou Algumas, o predicado sempre tem valor definido (Não e Sim).
- *Tipo*: atributo fraco: não ajuda a decidir se um cliente vai esperar ou não.
- algoritmo escolhe o atributo mais forte e coloca como raiz da árvore.

Aprendizagem Indutiva

Escolha entre dois atributos, Type e Patrons: neste caso, Patrons é escolhido porque discrimina melhor entre exemplos positivos (willWait=Yes) e negativos (willWait=No).



(a)



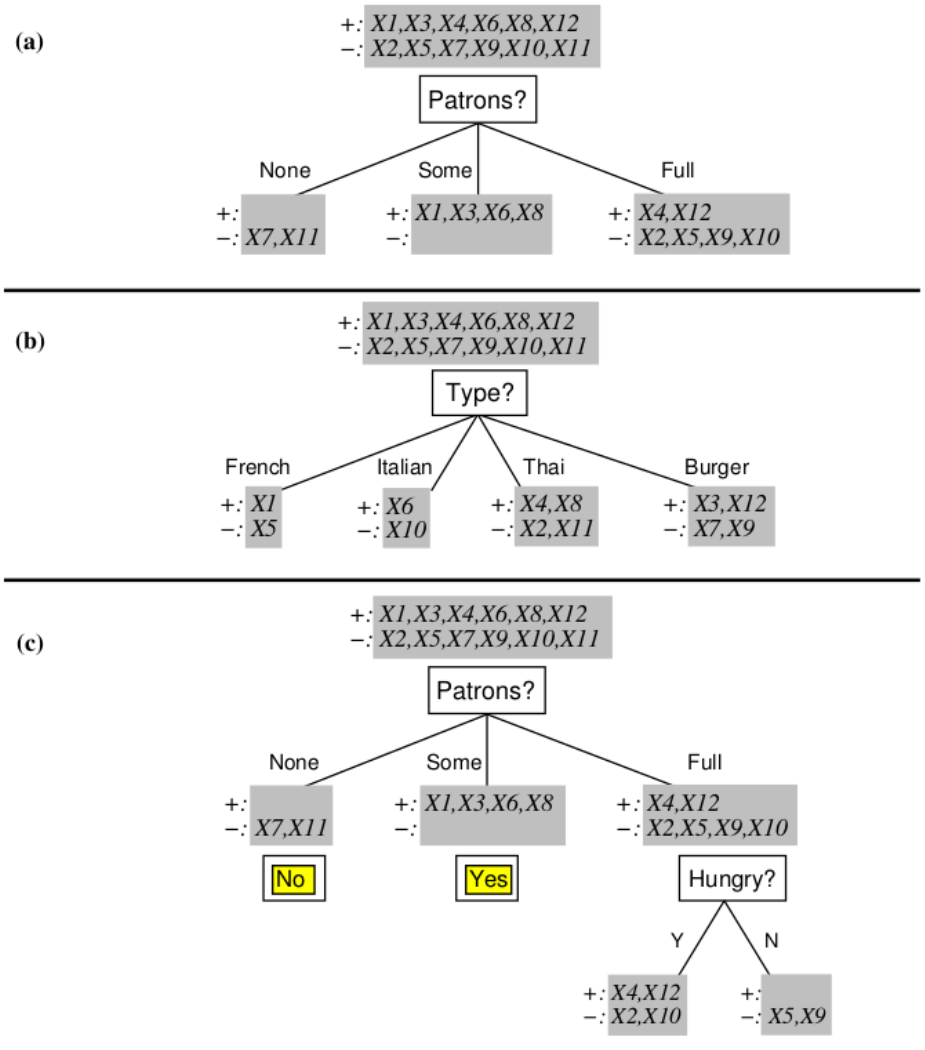
(b)

Aprendizagem Indutiva

- Restam subconjuntos de exemplos (para o valor Full de Patrons), algoritmo é aplicado recursivamente. 4 casos possíveis:
  - Se há alguns exemplos positivos e negativos, escolher o melhor atributo.
  - Se todos os exemplos restantes são positivos (ou todos negativos), podemos responder diretamente Sim ou Não.
  - Se não há mais exemplos, significa que nenhum exemplo foi observado para aquele caminho. Retorna valor default Sim ou Não dependendo da maioria das classificações do pai.
  - Se não há mais atributos, mas temos exemplos positivos e negativos, significa que estes exemplos têm exatamente a mesma descrição, mas diferentes classificações. Solução simples: voto majoritário.

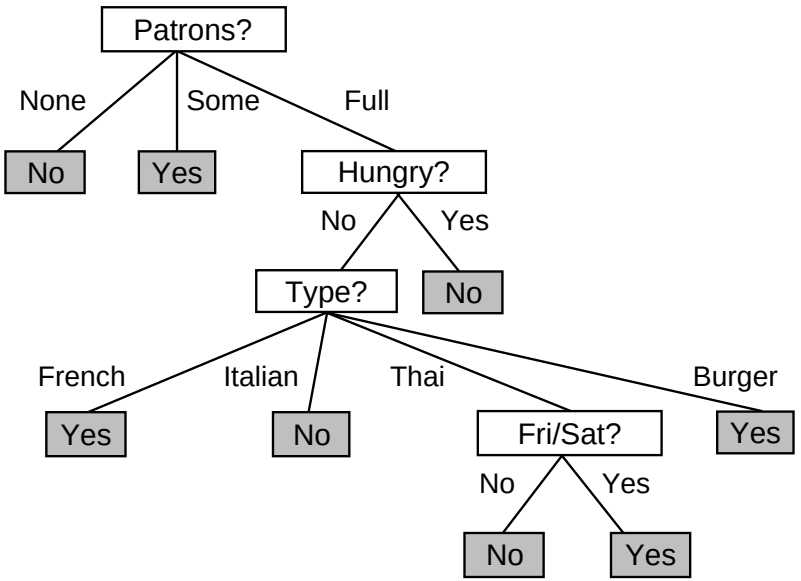
Aprendizagem Indutiva

Escolha do atributo Patrons (Clientes) e continuação da construção da árvore com a escolha do atributo Hungry (c).



Aprendizagem Indutiva

Possível árvore (mais compacta) gerada por um algoritmo de indução de árvores de decisão.



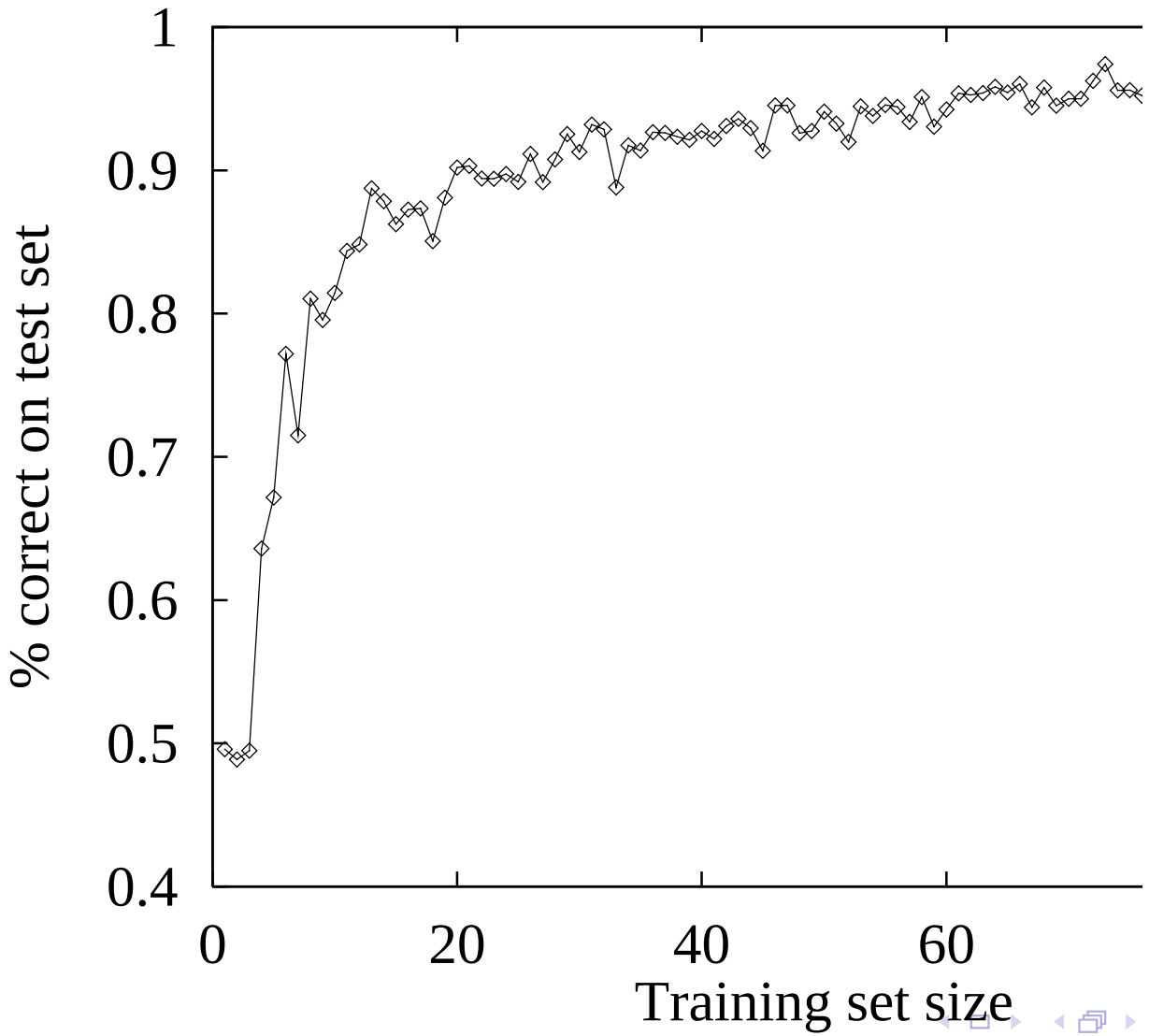
Aprendizagem Indutiva

- Observações:
  - árvore induzida é diferente da árvore original, apesar do agente ter utilizado exemplos gerados por aquela árvore.
  - algoritmo pode concluir fatos que não estão muito evidentes dos exemplos: sempre esperar por um restaurante Tailandês, se for um fim de semana.
  - Por causa destes fatos: muito tempo gasto em depuração procurando por erros não existentes!
  - qto mais exemplos mais detalhada será a árvore de decisão.
  - Neste exemplo, a árvore pode induzir erros já que nunca viu um caso onde o tempo de espera é 0-10mins, mas o restaurante está cheio.
- Questão: se o algoritmo induz uma árvore consistente, mas incorreta, através dos exemplos, quão incorreta é a árvore?

Desempenho de um Algoritmo de Aprendizagem

- Um algoritmo de aprendizagem é bom se produzir hipóteses que classificam bem **exemplos ainda não vistos**.
- duas formas de avaliar desempenho: após os factos, em avanço.
- Método após os factos: verificação das previsões de acordo com as classificações corretas num **conjunto de teste**.
  1. Escolher um conjunto grande de exemplos.
  2. Dividir este conj em conj de treinamento e conj de teste.
  3. Usar o algoritmo com o conj de treinamento como exemplo para gerar o modelo (a hipótese H).
  4. Calcular a porcentagem de exemplos no conj de teste corretamente classificados por H.
  5. Repetir passos 1 a 4 para tamanhos diferentes de conjs de treinamento e conjs de treinamento selecionados aleatoriamente para cada tamanho.
- Resultado do método: conj de dados q podem ser processados para produzir a **curva de aprendizagem**.

Desempenho de um Algoritmo de Aprendizagem



Evaluation Metrics

Most metrics used for model evaluation in classification tasks are based on the confusion (or contingency) matrix. For binary classification problems, we define one class as positive and the second as negative. We then define:

- **TP**: True Positives (number of instances that are positive and were predicted as positive)
- **TN**: True Negatives (number of instances that are negative and were predicted as negative)
- **FP**: False Positives (number of instances that are negative and were predicted as positive)
- **FN**: False Negatives (number of instances that are positive and were predicted as negative)
- $TP + FN$ corresponds to the total number of positive instances
- $TN + FP$ corresponds to the total number of negative instances
- $TN + TP$ is the number of correctly classified instances

Evaluation Metrics

An example of a confusion matrix for a binary classification problem:

Class/Predicted	+	-	Total
+	10	1	11
-	2	35	37
Total	12	36	48

Evaluation Metrics

From this table, we can extract:

- Total number of instances: 48, from which 11 are positive and 37 are negative.
- The classifier misclassified 1 positive and 2 negative instances (secondary diagonal shows the errors).
- The classifier correctly classified 10 out of the 11 positives and 35 out of the 37 negatives (main diagonal shows the correct classified instances).
- The classifier predicted 12 instances as being of class positive and 36 instances as being of class negative.

Evaluation Metrics

NOTE: if we have more than 2 classes, the confusion matrix will have more rows and columns. For example, assume the iris dataset (with classes setosa, virginica and versicolor):

```
=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 48  2 |  b = Iris-versicolor
  0  4 46 |  c = Iris-virginica
```

This was generated by running the weka toolkit with a Naive Bayes model.
In this matrix, all setosa examples were correctly classified by the model. The model missed 2 versicolor cases (these 2 were incorrectly mistaken with virginica). From the 50 cases virginica, 4 were misclassified as versicolor. In these cases,

Evaluation Metrics

- Recall = True Positive Rate (TPR) = Sensitivity =

$$\frac{TP}{TP + FN}$$

  Meaning: from all positives, how many were actually predicted as positive?

- True Negative Rate (TNR) = 1 - FPR =

$$\frac{TN}{TN + FP}$$

- Specificity = False Positive Rate (FPR) = 1 - TNR

Evaluation Metrics

- Accuracy = Correctly Classified Instances (CCI) =

$$\frac{TP + TN}{TP + FN + FP + TN}$$

  Meaning: from all instances, how many were actually correctly predicted?

- Error rate = 1 - CCI =

$$\frac{FP + FN}{TP + FN + FP + TN}$$

Evaluation Metrics

- Precision = Positive Predictive Value (PPV) =

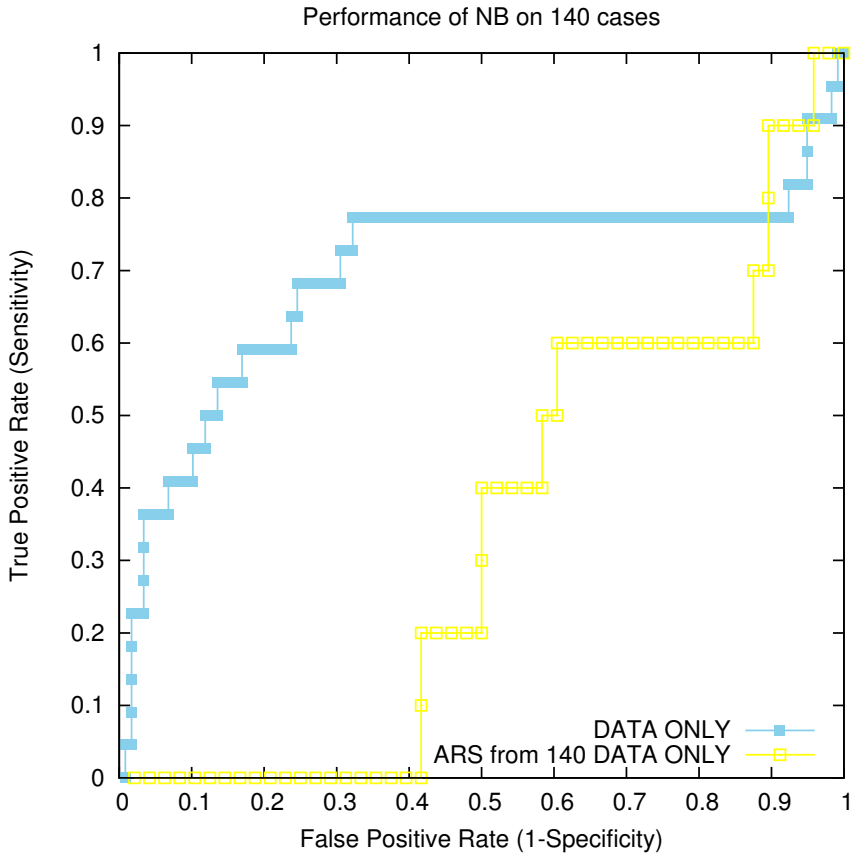
$$\frac{TP}{TP + FP}$$

  Meaning: from all instances predicted as positive, how many are actually positive?
- $F_\beta measure = (1 + \beta^2) \frac{Precision \times Recall}{(\beta^2 \times Precision) + Recall}$
  When $\beta = 1$, the F_1 measure is the harmonic mean between Precision and Recall.

Evaluation Metrics

- Receiver Operating Characteristic curve: values in the curve between 0 and 1, according to threshold variation.
- X axis represents False Positive Rate (FPR = 1 - Specificity). Best point is zero.
- Y axis represents True Positive Rate (TPR). Best point is 1.
- Curve is plotted with respect to a given class value (positive or negative).
- Used to help specialists to choose cut points related with false positive rate and true positive rate.
- ROC curves are useful when the model can predict numerical values.

ROC example



- This figure shows two ROC curves, one for each trained model.
- Points of the ROC curve are obtained by thresholding.
- The ideal point in an ROC curve is TPR=1 and FPR=0.

ROC example: thresholding

- For example, suppose we have the following expected and predicted values (where P is a positive class and N is negative)
- Also assume that your model gives you a number as a prediction:

Expected	Predicted
P	0.8
P	0.6
P	0.2
N	0.1
N	0.9
N	0.7
N	0.6
N	0.5

ROC example: thresholding

- The algorithm to calculate the curve points is:

```
Initalize arrays TP, FN, TP, FP with zeros
for i = 0 to 1 {
    if Expected == P and Predicted >= i TP[i]++
    if Expected == P and Predicted < i FN[i]++
    if Expected == N and Predicted >= i FP[i]++
    if Expected == N and Predicted < i TN[i]++
}
```

- This cycle may also vary around the predicted values: 0.1, 0.2, 0.5, 0.6, 0.7, 0.8, 0.9, but it needs to contain points 0.0 and 1.0.

ROC example: thresholding

- For this example, our arrays will be:

Threshold	# TP	# FP	TPR	FPR
0.0	3	5	1	1
0.1	3	4	1	4/5
0.2	3	4	1	4/5
0.3	2	3	2/3	3/5
0.4	2	2	2/3	2/5
0.5	2	1	2/3	1/5
0.6	2	1	2/3	1/5
0.7	1	1	1/3	1/5
0.8	1	1	1/3	1/5
0.9	0	0	0	0
1.0	0	0	0	0

ROC: area

- The ROC curve defines an area
- The area under the curve is also a very popular metric (AUC or AUCROC)
- This area varies between 0 and 1
- The closest to 1 the better

ROC: analysis

- When analysing an ROC curve, a specialist may decide which threshold to use in the model
- The specialist use cut points by using verticals that passes through different points in the X-axis
- Depending on the domain, specialists will look for thresholds that minimise either FP or FN

ROC: problem

- If the classes are imbalanced the ROC curve may show optimistic results
- If the positive class is much smaller than the negative, an error in the negative class will be much less significant than an error in the positive class
- In these cases, another curve is used: the Precision-Recall (PR) curve
- In the PR curve, the X-axis has the TPR (Recall) values and the Y-axis has the Precision values
- In the ROC curve we plot $\frac{TP}{TP+FN}$ against $\frac{FP}{FP+TN}$
- In the PR curve we plot $\frac{TP}{TP+FN}$ against $\frac{TP}{FP+TP}$
- For the same TP, the denominator of the PR curve will not dominate as much as the denominator of the ROC curve

Validation

- Models need to be validated and an estimate of the error needs to be calculated
- In order to do that, we usually divide our entire dataset in **training** and **testing** data
- There exists at least to methods for model validation, which requires iterative training and testing
  - cross-validation
  - bootstrapping

Cross-Validation

- In cross-validation, the dtaset is divided in k partitions (folds) of approximately the same size
- Training is performed k times, each time using one of the partitions for testing and the reamining for training
- Usually, cross-validation is **stratified**, meaning that, each fold will have approximately the same number of positive and negative examples...
- ...except if it is **leave-one-out**, where the dataset of size n is divided in $n - 1$ partitions and each test set has exactly one example
- leave-one-out is normally used when the dataset is small
- Care needs to be taken when calculating performance metrics in the context of cross-validation (see paper `Papers/v12-1-p49-forman-sigkdd.pdf`)

Bootstrapping

- In bootstrapping, the dataset is divided in training set partition and test set partition k times
- Usual values for partitioning may be 70%/30%, 80%/20%, 67%/33%
- In that case, metrics need to be calculated per each one of the k samples and an average is reported

Desempenho de um Algoritmo de Aprendizagem

- GASOIL, BP, sistema escrito a mão levaria 10 anos para ser completado. Usando algoritmo baseado em indução de árvores de decisão, foi desenvolvido em 100 dias! Considerado melhor do que um expert.
- Aprendendo a voar: C4.5 utilizado para extrair a árvore de decisão através de 90.000 exemplos obtidos por pilotos experientes num simulador. Árvore resultante convertida em C aprende e voa melhor do que os instrutores.

Teoria da Informação

- Encontrar medidas formais para classificar atributos como "bom" ou "razoável" ou "pobre" etc.
- **Entropia**: se um atributo tem como valores possíveis v_i com probabilidade $P(v_i)$, sua entropia será definida como: $I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) log_2 P(v_i)$

Teoria da Informação

- Considerando exemplos positivos e negativos: $I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$, estimativa da info contida em uma resposta correta.
- **Ganho de informação**: diferença entre a info original e a info introduzida pelo novo atributo da árvore $Ganho(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Restante(A)$
- Heurística usada por CHOOSE-ATTRIBUTE escolhe o atributo com maior ganho (menor entropia).
- Ex: $Ganho(Clientes) =$ $1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] \approx 0.541$ bits.
- O "1" da fórmula vem da info inicial: temos 6 exemplos da classe positiva (willWait=yes) e 6 exemplos da classe negativa (willWait=no), portanto info inicial = $-\frac{6}{12} log_2 \frac{6}{12} - \frac{6}{12} log_2 \frac{6}{12}$

Algoritmo ID3 para Indução de Árvores de Decisão

```
ID3(Examples, Target_Attribute, Attributes)
    Create a root node for the tree
    If all examples are positive,
        Return the single-node tree Root, with label = +.
    If all examples are negative,
        Return the single-node tree Root, with label = -.
    If number of predicting attributes is empty,
        Return the single node tree Root,
           with label = most common value of the
           target attribute in the examples.
    Else
        A = Attribute that best classifies examples
        Decision Tree attribute for Root = A
        For each possible value, vi, of A,
            Add a new tree branch below Root,
              corresponding to the test A = vi.
            Let Examples(vi) be the subset of examples that
              have the value vi for A
            If Examples(vi) is empty
               below this new branch add a leaf node with
                  label = most common target value in the examples
            Else
               below this new branch add the subtree
                  ID3 (Examples(vi), Target_Attribute, Attributes - {A})
            EndIf
        EndFor
    EndIf
    Return Root
```

Algoritmo ID3 para Indução de Árvores de Decisão

Nota: é fortemente recomendado dar uma olhada e entender também o algoritmo do livro AIMA (Russell and Norvig, 3a. edição), página 702, que é uma versão mais abstrata do algoritmo mostrado no slide anterior.

Algoritmo ID3 para Indução de Árvores de Decisão

- ID3 somente trata variáveis de classe binárias e atributos categóricos
- E se a variável tiver valores contínuos?
  $\rightarrow$ Discretização ou *binary splitting*

Atributos com valores contíunos

Possíveis abordagens:

- Discretização
  - intervalos podem ser gerados baseados em divisão igual de intervalos ou divisão igual de frequências (percentis) ou usando alguma forma de *clustering*
    - Divisão estática: discretiza uma única vez no início
    - Divisão dinâmica: repete a discretização a cada nó da árvore
- Decisão binária: $(A < v)$ or $(A \geq v)$
  - considera todas as possíveis divisões (*splits*) para encontrar a melhor
    $\rightarrow$ pode gastar muito recurso computacional

Complementary slides, from page 1 to 48, from Data Mining book by Tan, Steinbach, Karpatne and Kumar