- Aprendizagem de máquina
    1. Definição

       Aprendizagem de máquina é uma área da inteligência artificial que proporciona aos sistemas a habilidade de aprender automaticamente e melhorar através da experiência sem serem explicitamente programados. Aprendizagem de máquina foca-se no desenvolvimento de programas de computador que consegues aceder a dados e usá-los para aprender com eles mesmos.
    2. Tipos de aprendizagem

       Aprendizagem supervisionada: Tentam modelar relacionamentos e dependências entre a saída da suposição do destino e os recursos de entrada de forma que possamos prever os valores de saída para novos dados com base nas relações que aprendeu com os conjuntos de dados anteriores. Neste tipo de aprendizagem o modelo é previsto, os dados são rotulados e os principais tipos de problemas de aprendizagem supervisionada incluem problemas de regressão e classificação (exs: Vizinho mais próximo, Naive Bayes, Árvores de decisão, regressão linear, Support Vector Machines (SVM) e redes neurais).

       Aprendizagem não supervisionada: A máquina é treinada com dados não rotulados. É usado maioritariamente para deteção de padrões e modelagem descritiva. Não existem rótulos de saída aqui com base nos quais o algoritmo pode tentar modelas relacionamentos. Os algoritmos tentam usar técnicas nos dados de entrada para minerar as regras, detetar padrões e resumir e agrupar os pontos de dados que ajudam a obter valores significativos e descrevem melhor os dados para os usuários. É um modelo descritivo e os principais tipos de algoritmos de aprendizagem não supervisionados incluem algoritmos de clustering e algoritmos de regra de associação (exs: k-means clustering e association rules).

       Aprendizagem semi supervisionada: Nos dois tipos anteriores, não há rótulos para toda a observação no conjunto de dados ou rótulos presentes em todas as observações. A aprendizagem semi supervisionada está entre esses dois. Em muitas situações práticas, o custo para rotular é bastante alto, pois requer especialistas humanos qualificados. Assim, na ausência de rótulos na maioria das observações, mas presentes em poucos, os algoritmos semi supervisionados são os melhores candidatos para o modelo de construção. Esses métodos exploram a ideia de que, embora as associações de grupos dos dados não rotulados sejam desconhecidas, esses dados contêm informações importantes sobre os parâmetros do grupo.

       Aprendizagem reforçada: O método visa usar as observações coletadas da interação com o ambiente para realizar ações que maximizem a recompensa ou minimizem o risco. O algoritmo de aprendizagem reforçada (chamada de agente) aprende continuamente do ambiente de maneira iterativa. No processo, o agente aprende a partir das suas experiências do ambiente até explorar todo o espetro de estados possíveis. Esta aprendizagem permite que máquinas e agentes de software determinem automaticamente o comportamento ideal dentro de um contexto específico, a fim de maximizar o seu desempenho. É necessário um feedback de recompensa simples para o agente aprender o seu comportamento, isso é conhecido como sinal de reforço. Existem muitos algoritmos diferentes que lidam com esse problema.

De facto, a aprendizagem por reforço é definida por um tipo específico de problema, e todas as suas soluções são classificadas como algoritmos de aprendizagem de reforço. No problema, um agente deve decidir a melhor ação para selecionar com base no seu estado atual. Quanto essa etapa é repetida, o problema é conhecido como processo de decisão de Markov. Para produzir programas inteligentes (também chamados agentes), a aprendizagem por reforço passa pelas seguintes etapas:

- O estado de entrada é observado pelo agente;
- A função de tomada de decisão é usada para fazer com que o agente execute uma ação;
- Depois da ação ser executada, o agente recebe a recompensa ou reforço do ambiente e
- As informações do par de ação do estado sobre a recompensa são armazenadas.

Alguns algoritmos que usam este tipo de aprendizagem são Q-Learning, Temporal Difference (TD) e Deep Adversarial Networks.

3. Métricas de avaliação

As métricas de avaliação disponíveis são Precisão de Classificação, Perda Logarítmica, Matriz de Confusão, Área sob Curva, F1 Score, Erro Absoluto Médio e Erro Quadrático Médio.

4. Métodos de avaliação de modelos

Precisão de Classificação:

$$Precisão = \frac{Número\ de\ previsões\ corretas}{Número\ total\ de\ previsões\ realizadas}$$

Perda logarítmica:

$$Perda\ logarítmica = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log p_{ij}$$

y_{ij} = Indica se uma amostra i pertence à classe j

p_{ij} = Indica a probabilidade de uma amostra i pertencer à classe j

Matriz de Confusão:

n=165	**Previsão: Não**	**Previsão: Sim**
Output: Não	50	10
Output: Sim	5	100

Positivos Verdadeiros (PV): Previsão Sim e Output Sim
Negativos Verdadeiros (NV): Previsão Não e Output Sim
Positivos Falsos (PF): Previsão Sim e Output Não
Negativos Falsos (NF): Previsão Não e Output Não

$$Precisão = \frac{PV+NF}{Número\ total\ de\ amostas} = \frac{100+50}{165} = 0.91$$

Área sob Curva:

Sensibilidade: Proporção de pontos de dados positivos que são corretamente considerados como positivos.

$$Sensibilidade = \frac{PV}{NF + PV}$$

Especificidade: Proporção de pontos de dados negativos que são erradamente considerados positivos.

$$Especificidade = \frac{PF}{PF + NV}$$

F1 Score:

$$F1 = 2 * \frac{1}{\frac{1}{Precisão} + \frac{1}{Recall}}$$

$$Precisão = \frac{PV}{PV + PF}$$

$$Recall = \frac{PV}{PV + NF}$$

Erro absoluto médio:

$$EAM = \frac{1}{N}\sum_{j=1}^{N} |y_j - y'_j|$$

Erro quadrático médio:

$$EQM = \frac{1}{N}\sum_{j=1}^{N} \left(y_j - y'_j\right)^2$$

5. BIAS e variância de modelos

BIAS é a diferença entre a previsão média do modelo e o valor correto que estamos a tentar prever. Modelos com alto BIAS prestam pouca atenção aos dados de treino e simplifica demais o modelo. Isso leva sempre a um alto erro nos dados de treino e teste. Variância é a variação do modelo de previsão para um determinado ponto de dados ou um valor que diz a propagação dos nossos dados. Modelos com alta variância prestam muita atenção aos dados de treino e não generalizam os dados que não foram vistos antes. Como resultado, esses modelos têm um desempenho muito bom nos dados de treino, mas apresentam altas taxas de erro nos dados de teste.

Deixar a variável que estamos a tentar prever como Y e as outras co variáveis como X. Assumimos que existe uma relação entre os dois de forma que

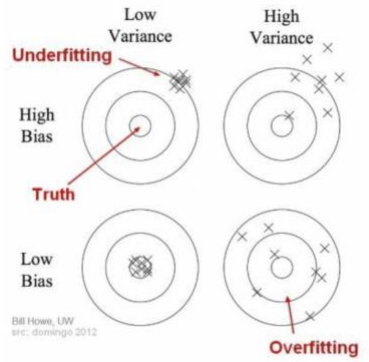
$$Y = f(x) + e$$

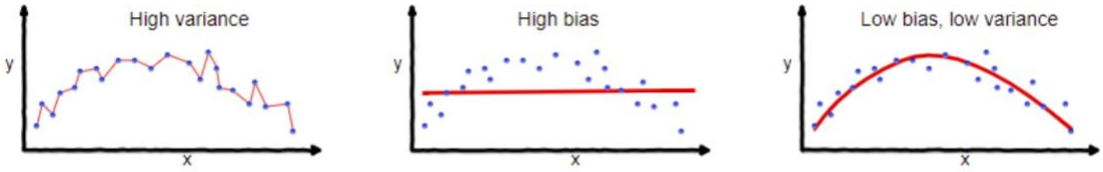
Onde o termo de erro e é normalmente distribuído com uma média de 0.
Então o erro é:

$$Err(x) = \left(E[f'(x)] - f(x)\right)^2 + E[(f'(x) - E[f'(x)])^2] + \sigma_e^2$$

$$Err(x) = BIAS^2 + Variância + Erro\ irredutível$$

É possível analisar diagramas e gráficos para verificar se os valores são "*Underfitting*" ou "*Overfitting*".





overfitting underfitting Good balance

Se o nosso modelo for simples demais e tiver poucos parâmetros então poderá ter alto valor de BIAS e baixo de Variância. Por outro lado, se o nosso valor tiver um número elevado de parâmetros então terá um alto valor de variância e baixa BIAS. Tem de ser encontrado um valor equilibrado sem dar "Overfitting" ou "Underfitting" aos dados.

Essa desvantagem na complexidade é a razão pela qual existe uma compensação entre o BIAS e a variância. Um algoritmo não pode ser mais e menos complexo ao mesmo tempo.

- Árvores de decisão
  1. Como representar uma árvore de decisão

     Uma árvore de decisão é uma ferramenta de suporte à decisão que usa um gráfico ou modelo semelhante à arvore de decisões e suas possíveis consequências, incluindo resultados de eventos futuros, custos de recursos e utilidade. É uma maneira de exibir um algoritmo que contém apenas instruções de controle condicional.

     Uma árvore de decisão é uma estrutura semelhante a um fluxograma na qual cada nó interno representa um "teste" num atributo, cada ramo representa o resultado do teste e cada nó folha representa um rótulo de classe (decisão tomada após computador todos os atributos). Os caminhos da raiz para a folha representam regras de classificação.
  2. Construir uma árvore de decisão a partir de observações

     Escolher o atributo mais importante, aquele que influencia mais a classificação do exemplo. Depois de escolhido o atributo, restam os subconjuntos de exemplos em que é retornado um dos quatro tópicos:

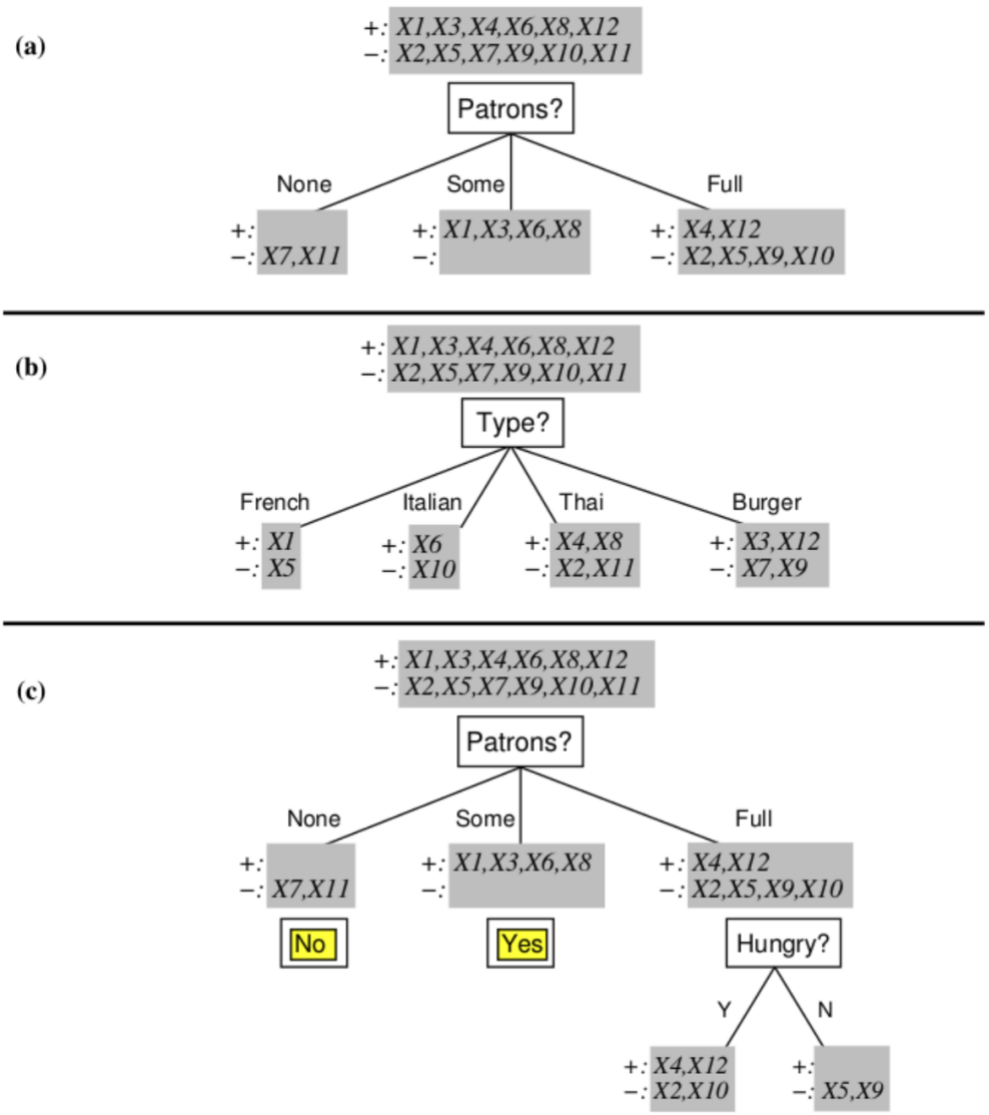
     - Se há alguns exemplos positivos e negativos, escolher o melhor atributo;

- Se todos os exemplos restantes são positivos (ou todos negativos), 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 ou

- Se não há mais atributos, mas temos exemplos positivos e negativos, significa que estes exemplos têm exatamente a mesma descrição, mas diferentes classificações. Solução: voto majoritário.

Exemplo:



3. Algoritmo de construção de árvore de decisão

O algoritmo usado é o ID3 e CART:

```
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
```

4. Manipulação de variáveis numéricas

Quando uma variável é numérica, é realizado uma discretização ou binary splitting.

Discretização:

- Os 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) ou (A >= v)

- Considera todas as possíveis divisões (splits) para encontrar a melhor
    - Pode gastar muitos recursos computacionais

5. Manipulação de dados em falta

Existem vários métodos usados por várias árvores de decisão. Simplesmente ignora-se os valores em falta (como o algoritmo ID3) ou tratar o valor em falta como outra categoria (no caso de um recurso nominal) não são manipulações reais dos valores perdidos. No entanto, essas abordagens foram usadas nos estados iniciais do desenvolvimento das árvores de decisão. As abordagens de manipulação reais para dados ausentes não usam pontos de dados com valores ausentes na avaliação de uma divisão. No entanto, quando nós filhos são criados e treinados, essas instâncias são distribuídas de alguma forma. Portanto existem as seguintes formas de abordar:

- Tudo vai para o nó que já tem o maior número de instâncias (CART);
- Distribuir para todos os filhos, mas com pesos diminuídos, proporcional ao número de instâncias de cada nó filho (C45 e outros);
- Distribuir aleatoriamente para apenas um único nó filho, eventualmente de acordo com uma distribuição categórica;
- Construir, classificar e usar substitutos para distribuir instâncias para um nó filho, onde os substitutos são recursos de entrada que lembram melhor como o recurso de teste envia instâncias de dados para o nó filho esquerdo ou direito (CART, se falhar, a regra do majoritário é usado).

6. Métricas para escolha de melhor variável da árvore

No algoritmo ID3 faz-se o seguinte:

1. Calcular entropia e ganho de informação para cada atributo

$I(P(v_1), \dots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)$ em que v_i são os valores possíveis e $P(v_i)$ é a probabilidade de v_i acontecer.

Ganho de informação é a diferença entre a info original e a info introduzida pelo novo atributo da árvore. Heurística usada escolhe o atributo com maior ganho (menor entropia).

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

2. Classificar os atributos em ordem crescente de entropia (ordem decrescente para ganho de informação)
3. Enquanto a lista de atributos não estiver vazia ou em comprimento limite, selecionar o primeiro atributo para ser a raiz da árvore/subárvore
4. Remover o atributo da lista classificada
5. Voltar para o passo 3.

Exemplo de cálculo de entropia e de importância:

Nº Exemplos: 12

	Yes	No
None	a	b
Some	c	d
Full	e	f

$$-\frac{c}{c+d}\log_2\left(\frac{c}{c+d}\right) - \frac{e}{c+d}\log_2\left(\frac{e}{c+d}\right)$$

$$E(Patrons) = \frac{a+b}{12} * \underbrace{\left(\frac{a}{a+b}, \frac{b}{a+b}\right)}_{\text{Yes}\quad\text{No}} + \frac{c+d}{12} * \underbrace{\left(\frac{c}{c+d}, \frac{d}{c+d}\right)}_{\text{Some}} + \frac{e+f}{12} * \underbrace{\left(\frac{e}{e+f}, \frac{f}{e+f}\right)}_{\text{Full}}$$

7. Complexidade do algoritmo de construção de árvores

   A complexidade temporal para árvores de decisão é O(Nkd), em que N = número de exemplos de treino, k = número de características e d = profundidade da árvore de decisão.

- Redes Bayesianas

1. Como representar uma rede Bayesiana

   Uma rede Bayesiana é um grafo dirigido acíclico na qual cada extremidade corresponde a uma dependência condicional, e cada nó corresponde a um valor random único. Se um extremo (A,B) existe no grafo a conectar as variáveis random A e B, então significa que P(B|A) é um fator na distribuição da probabilidade conjunta. As redes Bayesianas satisfazem a propriedade de Markov, que afirma que um nó é condicionalmente independente dos seus não descendentes, dados os seus pais.

2. Construir uma rede Bayesiana a partir de observações

   A rede é construída de forma a que cada nó é condicionalmente independente dos seus predecessores, dada a probabilidade dos seus pais. A equação usada para guiar a construção da topologia da rede é:

$$P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(x_i))$$

   De forma a que a estrutura da rede esteja correta para o domínio, tem de se escolher pais adequados que garantam que cada nó é condicionalmente independente dos seus pais.

3. Construção de tabelas de probabilidade condicional (TPCs)

   Duas formas de entender:

   - Representação da distribuição de probabilidade conjunta. Útil para construir a rede;
   - Conjunto de variáveis de condicionalmente independentes. Útil para projetar procedimentos de inferência.

   Representar a probabilidade conjunta:

   - Cada entrada da tabela pode ser calculada através da informação disponível na rede;
   - Uma entrada de tabela genérica representa a probabilidade de uma conjunção dos valores da variável aleatória:
     - $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$
     - $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(x_i))$
   - Cada entrada da tabela é representada pelo produto dos elementos apropriados do TPC.
   - Um TPC fornece uma representação decomposta da distribuição condicional

   Ex:

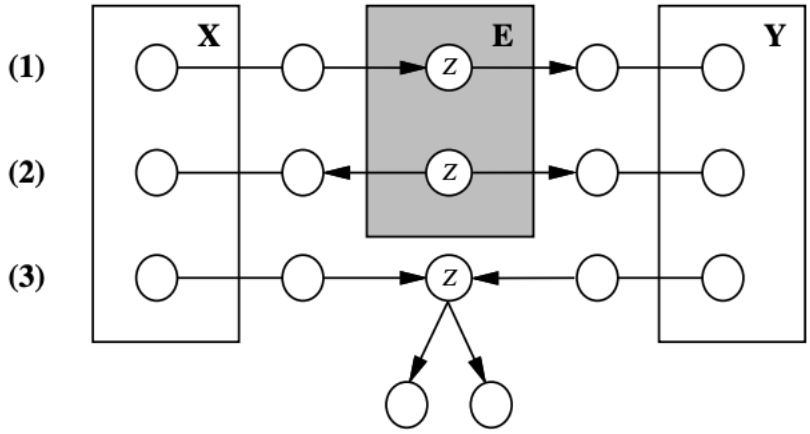
$$P(J \wedge M \wedge A \wedge \neg B \wedge \neg E) = P(J \mid A) * P(M \mid A) * P(A \mid \neg B \wedge \neg E) * P(\neg B) * P(\neg E)$$

4. Condições de independência de variáveis (d-separation e Markov Blanket)

d-separation diz que se todas as arestas entre X e Y estiverem d-separadas por E, então X e Y são condicionalmente independentes, dado E. Nessa condição, dizemos que a borda está bloqueada.

Uma aresta é bloqueada, dado um conjunto de nós E, se houver caminho de modo que uma das seguintes condições seja satisfeita:

1. Variável Z está em E e Z não tem nenhum incidente de arco e outro não incidente;
2. Z está em E e Z tem ambos os arcos não incidentes e
3. Nem Z nem qualquer descendente de Z estão em E, e ambos os arcos são incidentes a Z.



O objetivo da d-separation é calcular a distribuição de probabilidade posterior para um conjunto de variáveis de consulta, dadas as evidências:

P(Consultas | Evidências).

Em princípio, qualquer nó pode ser consulta ou evidência. Quando se aprende classificadores, uma variável é a consulta e algumas outras são provas.

5. Algoritmo para construção de redes Bayesianas

function BELIEF-NET-ASK(X) **returns** a probability distribution over the values of X
  inputs: X, a random variable

  SUPPORT-EXCEPT(X, *null*)

---

function SUPPORT-EXCEPT(X, V) **returns** $\mathbf{P}(X|E_{X\setminus V})$

  if EVIDENCE?(X) **then return** observed point distribution for X
  else
    calculate $\mathbf{P}(E^{-}_{X\setminus V}|X)$ = EVIDENCE-EXCEPT(X, V)
    $\boldsymbol{U} \leftarrow$ PARENTS[X]
    if U is empty
      then return $\alpha\ \mathbf{P}(E^{-}_{X\setminus V}|X)\ \mathbf{P}(X)$
    else
      for each U_i **in** U
        calculate and store $\mathbf{P}(U_i|E_{U_i\setminus X})$ = SUPPORT-EXCEPT(U_i, X)
      return $\alpha\ \mathbf{P}(E^{-}_{X\setminus V}|X) \sum_{\mathbf{u}} \mathbf{P}(X|\mathbf{u}) \prod_i \mathbf{P}(U_i|E_{u_i\setminus X})$

---

function EVIDENCE-EXCEPT(X, V) **returns** $\mathbf{P}(E^{-}_{X\setminus V}|X)$

  $\mathbf{Y} \leftarrow$ CHILDREN[X] $-$ V
  if Y is empty
    then return a uniform distribution
  else
    for each Y_i **in** Y **do**
      calculate $\mathbf{P}(E^{-}_{Y_i}|y_i)$ = EVIDENCE-EXCEPT(Y_i, null)
      $\mathbf{Z}_i \leftarrow$ PARENTS[Y_i] $-$ X
      for each Z_{ij} **in** $\mathbf{Z}_i$
        calculate $\mathbf{P}(Z_{ij}|E_{Z_{ij}\setminus Y_i})$ = SUPPORT-EXCEPT(Z_{ij}, Y_i)
    return $\beta \prod_i \sum_{y_i} P(E^{-}_{Y_i}|y_i) \sum_{\mathbf{z}_i} \mathbf{P}(y_i|X, \mathbf{z}_i) \prod_j P(z_{ij}|E_{Z_{ij}\setminus Y_i})$

6. Manipulação de variáveis numéricas

   A força de uma rede Bayesiana é altamente escalável e pode aprender de forma incremental, porque tudo o que fazemos é contar as variáveis observadas e atualizar a tabela de distribuição de probabilidade. Semelhante à rede neural, a rede Bayesiana espera que todos os dados sejam binários, a variável categórica precisará de ser transformada em múltiplas variáveis binárias, conforme descrito acima. Variável numérica geralmente não é um bom ajuste para a rede Bayesiana.

7. Manipulação de dados em falta

   Quando somos deparados com dados em falta, o que se faz em redes Bayesianas é supor que sabemos a estrutura causal do problema inicial. Criámos suposições face aos dados que estão em falta.

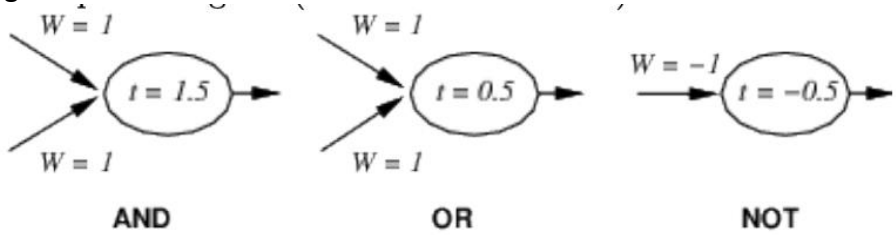
8. Cálculo de probabilidades usando a rede

   Explicado no ponto 3 deste capítulo

9. Expressão probabilística da rede (descrição)
10. Complexidade do mecanismo de inferência (cálculo das probabilidades)
11. Complexidade do algoritmo de construção da rede e das TPCs

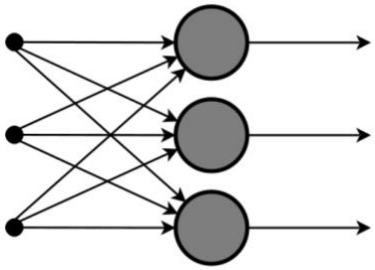
- Redes Neurais
    1. Como representar uma rede neural

       Para representar uma rede devemos de:
       - Definir a topologia: número de unidades, tipos de unidades e formato de conexões;
       - Inicializar os pesos da rede e treinar estes pesos usando um algoritmo de aprendizagem aplicado a um conjunto de treino para a determinada tarefa implementada pela rede e
       - As operações de unidades individuais podem ser comparadas com portas lógicas.



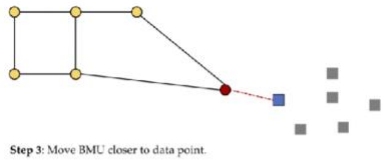
    2. Tipos de rede neural

       Feedforward Neural Network: este tipo de rede neural é uma das formas mais simples de ANN (Artificial Neural Network), onde os dados ou a entrada viajam numa direção. Os dados passam pelos nós de entrada e saem nos nós de saída. Essa rede neural pode ou não ter camadas ocultas. Em palavras simples, ele tem uma onda propagada frontal e nenhuma propagação reversa usando uma função de ativação de classificação.

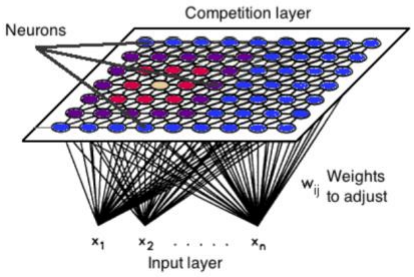


       Radial basis function Neural Network: As funções básicas radiais consideram a distância dum ponto em relação ao centro. As funções têm duas camadas, primeiro onde as características são combinadas com a função base radial na camada interna e, em seguida, a saída desses recursos é levada em consideração ao computar a mesma saída no próximo passo que é basicamente uma memória.

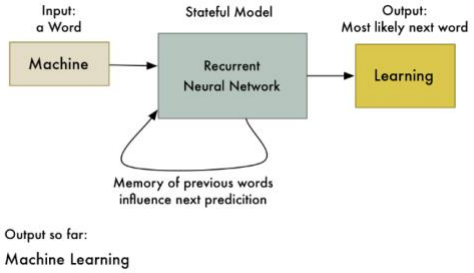
Kohonen Self Organizing Neural Network: O objetivo de um mapa Kohonen é inserir vetores de dimensão arbitrária em mapas discretos compostos por neurônios. O mapa precisa de ser treinado para criar a sua própria organização dos dados de treino. Compreende uma ou duas dimensões. Ao treinar o mapa, a localização do neurônio permanece constante, mas os pesos diferem dependendo do valor.



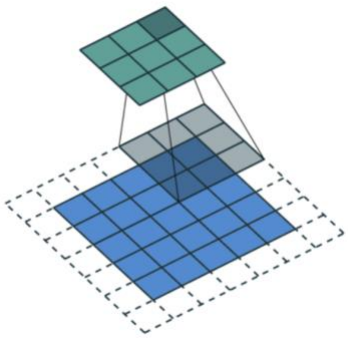
Step 3: Move BMU closer to data point.



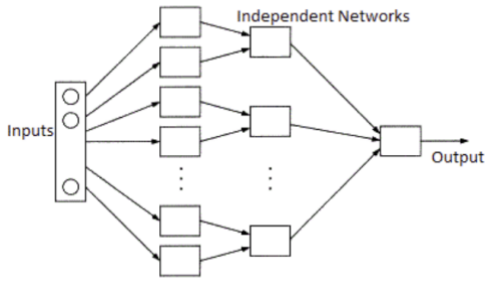
Recurrent Neural Network (RNN) – Long Short Term Memory: A rede neural recorrente trabalha com o princípio de salvar a saída de uma camada e alimentar isso de volta para a entrada para ajudar na previsão do resultado da camada. A primeira camada é formada semelhante a redeu neural de Feedforward com o produto da soma dos pesos e das características. O processo da rede neural recorrente começa assim que isso é calculado, isso significa que, de um passo de tempo para o próximo, cada neurônio lembra-se de algumas informações contidas no passo anterior.



Convolutional Neural Network: Estes tipos de redes são semelhantes às redes neurais avançadas, onde os neurônios têm pesos e biases capazes de aprender.



Modular neural network: Possuem uma coleção de diferentes redes trabalhando independentemente e contribuindo para a saída. Cada rede neural tem um conjunto de entradas que são únicas em comparação com outras redes que constroem e executam subaéreas. Essas redes não interagem ou sinalizam umas às outras na realização das tarefas. A vantagem de uma rede neural modular é que ela divide um grande processo computacional em componentes menores, diminuindo a complexidade. Esta divisão ajudará a diminuir o número de conexões e negará a interação dessas redes entre si, o que, por sua vez, aumentará a velocidade de computação.



3. Funcionamento de uma rede neural

   A ideia básica por detrás de uma rede neural é simular muitas células cerebrais densamente interconectadas dentro dum computador, para que possa aprender coisas, reconhecer padrões e tomar decisões de maneira humana. A rede aprende sozinha, assim como um cérebro. As redes neurais são simulações de software: são feitas através de programação de computadores muito comuns, trabalhando de uma maneira muito tradicional com os transístores comuns e portas lógicas conectadas serialmente, para se comportarem como se fossem construídos biliões de células cerebrais altamente interconectadas a trabalhar em paralelo.
4. Resolver problemas usando redes neurais

   Retratar exemplos reais de como usar redes neurais (simulação de jogos, simulação de eventos, etc)
5. Algoritmo de aprendizagem de pesos num Multi Layer Perceptron (MLP)

   Redes de hopfield: algoritmo

$$
\begin{aligned}
&\text{for i} = 1 \text{ to n do} \\
&\quad v_i = v_i^0; \\
&\text{endfor} \\
&\text{repeat} \\
&\quad \text{for i} = 1 \text{ to n do} \\
&\quad\quad v_i^- = v_i; \\
&\quad\quad v_i = step(\textstyle\sum_{j=1}^{n} w_{ij} v_j + e_i - \Theta_i) \\
&\text{until } v_i = v_i^- \text{ for all } n_i \in N
\end{aligned}
$$

Perceptrons: regra de aprendizagem:

- Inicializar pesos $w_0, w_1, w_2, \ldots, w_d$
- Repeat
  - for each training example (x_i, y_i)
    - compute $f(w, x_i)$
    - update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda \lfloor y_i - f(w^{(k)}, x_i) \rfloor x_i$$

- Until stopping criteria condition is met

6. Características de boas funções de ativação

   A característica mais importante de uma boa função de ativação é ser diferenciável para que desse modo seja possível minimizar o erro de classificação durante o treino da rede. Algumas funções de ativação são: step0, step, sigmóide, reLU, tanh.

7. Complexidade do algoritmo de aprendizagem de pesos

- Geração de Planos
  1. Definições

     Geração de planos são estratégias de busca que conseguem produzir sequências de ações que resultam num estado final, porém as representações de estados são atômicas (simples e primitivas). Requerem heurísticas especificas ao domínio para encontrar a solução de forma eficiente e é necessário ter uma solução que tire partido da estrutura do problema (onde se usam linguagens baseadas em lógica de primeira ordem para representar a estrutura do problema).

  2. Partial Order Planner (POP)

     O planeamento de ordem parcial é uma abordagem para o planeio automatizado que mantém uma ordenação parcial entre ações e apenas comete a ordenação entre ações quando forçado, ou seja, a ordenação de ações é parcial. Talvez, esse planeamento não especifique qual ação sairá primeiro quando duas ações são processadas. Por outro lado, o planeamento de ordem total mantém uma ordenação total entre todas as ações em todos os estágios do planeamento. Dado um problema no qual a sequência de ações é necessária para atingir a meta, um plano de ordem parcial especifica todas as ações que precisam de ser tomadas, mas especifica uma ordenação entre as ações somente quando necessário.

3. Algoritmo POP

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, c ← SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

---

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

4. Complexidade do algoritmo POP
5. Construção de planos

   Metodologia para resolver problemas com a abordagem de geração de planos:
   - Decidir sobre o que falar;
   - Decidir o vocabulário de condições (literais), operadores e objetos;
   - Codificar operadores;
   - Codificar uma descrição de uma instância do problema (estado inicial, por exemplo);
   - Apresentar problemas ao gerador de planos e obter planos (problema = objetivo).
6. Planos de ordem parcial e de ordem total

   Explicada a diferença no ponto 2 deste capítulo.

7. Planos totalmente instanciados
   Maioria dos “planners” utilizados descreve estados e operadores em STRIPS ou extensões. Os estados são representados por conjunções de literais “ground” (completamente instanciados), sem funções. Somente os operadores completamente instanciados podem ser executados.
8. Planos parcialmente instanciados
9. Plano consistente
   Um plano é consistente se não tiver contradições na ordem ou valores das restrições. Contradição ocorre quando $S_i \prec S_j$ aparece ao mesmo tempo que $S_j \prec S_i$ no plano gerado ou quando = A e v = B.
10. Definição de ações pré-condições e pós-condições
11. Programação em PDDL

- WEKA
  1. Interpretação de resultados
     Explicado no capítulo machine Learning e decision trees
  2. Matriz de confusão
     Explicado no capítulo machine Learning e decision trees
  3. Interpretação de valores na matriz de confusão
     Explicado no capítulo machine Learning e decision trees