Material Complementar - ECA7N - IA - 2019/2

Neste arquivo está disposto um resumo das dos principais tópicos que serão abordados no decorrer da disciplina. Cabe ressaltar que este material não substitui o conteúdo abordado em sala de aula.

Junto com os resumos há links de referência com maiores detalhes sobre os conteúdos abordados.

Link Gerais:

- Slides do Livro Inteligência Artificial feito pelo Russell e Norvig em PDF: http://aima.eecs.berkeley.edu/slides-pdf/
- Instruções para professores de IA feito por Russell e Norvig: http://aima.cs.berkeley.edu/instructors.html
- Slides do Livro Inteligência Artificial feito pelo Russell e Norvig em latex: http://aima.eecs.berkeley.edu/slides-tex/
- Códigos-fonte dos algoritmos e exercícios do livro Inteligência Artificial do Russell e Norvig: https://github.com/aimacode
- Link do site da disciplina: https://sites.google.com/view/guilherme-borges

Sumário

UNIDADE I - Introdução à Inteligência Artificial

1.1 Histórico da inteligência artificial

Link para slides: link.

Vídeos para complementar a disciplina

Vídeo aspectos gerais, filosóficos e reflexões sobre IA:

- Canal Pirula: Inteligência Artificial e Evolução: link

- TEDxMaua: Inteligência artificial e o empoderamento do ser humano: link
- Canal Nerdologia: Computadores têm alma? link
- Canal Nerdologia: O computador ou o cérebro, quem é o mais potente? link
- Canal Nerdologia: Singularidade humana e Ghost in the Shell: link
- Canal Nerdologia: Ultron e a revolução da inteligência artificial: link
- Canal Peixe Babel: Stephen Hawking,Elon Musk e Inteligência Artificial: link
- Canal Peixe Babel: Verdadeiros Perigos da Inteligência Artificial: link
- Canal Peixe Babel: É Preciso Entender a Mente Humana? link
- Canal Peixe Babel: O Futuro das Profissões: link
- Canal Nerdologia: Robôs vão tomar o seu emprego? Automatização do trabalho: link
- Canal Nerdologia: O futuro do seu emprego: link
- Canal Nerdologia: Vivemos em na Matrix: link

Aprendizagem de máquina:

- Canal Nerdologia: Machine Learning: como ensinar uma máquina a aprender: link
- Canal Nerdologia: Redes Neurais e Machine Learning: link
- Canal Peixe Babel: Bolhas Sociais e Algoritmos de Recomendação: link
- Canal Peixe Babel: Trolls da Internet vs Inteligência Artificial: link
- Canal Nerdologia: Ciência de dados: link
- Canal IBM Brasil: Como a Inteligência Artificial realmente funciona? link

UNIDADE II - Representação do Conhecimento

2.1 Formas de representação do conhecimento

Ferramenta de Sistema Baseado em Regra (Rule Engine):

- Lista de algumas com descrição: https://www.baeldung.com/java-rule-engines
- Biblioteca MRules (Java): https://mrules.xyz/en
- Biblioteca Drools (Java): https://www.drools.org/
- Easy Rules (Jeasy) (Java): http://www.jeasy.org/
- Nools (JavaScript): https://www.npmjs.com/package/nools
- Json (JavaScript): https://www.npmjs.com/package/json-rules-engine
- PyKnow (Python): https://pyknow.readthedocs.io/en/stable/
- N-Cube (Groove): https://github.com/jdereg/n-cube
- jruizgit (Ruby, Python, Node.js): https://github.com/jruizgit/rules

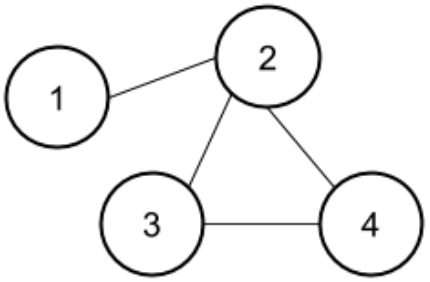
2.2 Resolução de problemas

Links de referência:

- Slides representação do conhecimento 01: link.
- Material grafos:
  - Slides Kleinberg e Tardos: link.
  - Slides Charles Ornelas Almeida e Nivio Ziviani:
    - Link 1
    - Link 2
- Representação do conhecimento Von Zuber: link.

Exercícios de Grafos:

1 - Representar o Grafo abaixo na forma matemática e implementar usando matriz de incidência e usando encadeamento.



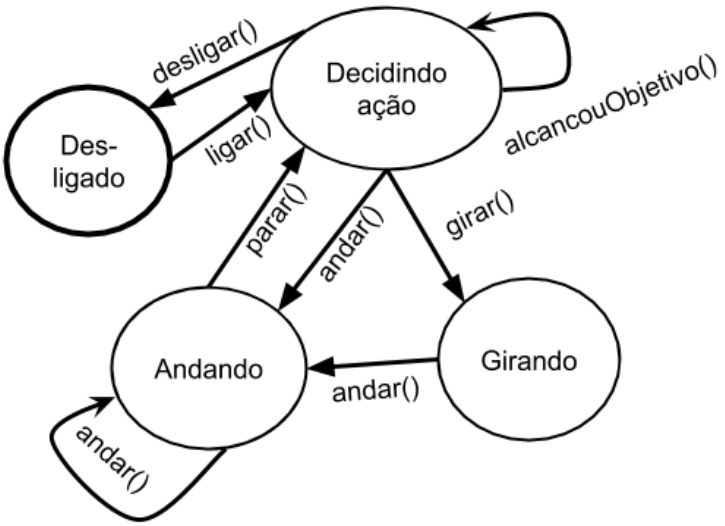
2 - Considere a representação matemática de grafo abaixo, desenhe essa representação e escolha uma forma de implementar ela.

Onde G = (V, E,P):

V = { 1, 2, 3, 4, 5, 6, 7, 8 }

E = { 1–2, 1–3, 2–3, 2–4, 2–5, 3–5, 3–7, 3–8, 4–5, 5–6, 7–8 }

3 - Modelar matematicamente e implementar o grafo abaixo que representa uma máquina de estados. Podemos pensar em um robô que está tentando sair do labirinto, para isso representamos os estados do Robô nos Vértices e as ações de troca de estado nas Arestas.



OBS: Em um caso real, mais funções seriam necessárias e elas não necessariamente seriam adequadamente representadas usando uma máquina de estados. Por exemplo, o Robô continuaria andando até encontrar um obstáculo, nesse momento ele executaria a função **parar()** para decidir a ação e identificar se esse obstáculo é o objetivo. Embora, a função que interrompe o andar não esteja representada, mas ela existe e influencia as ações que afetam a máquina de estados.

2.3 Estratégias de busca (heurística e local)

Links de referência:

- http://aima.eecs.berkeley.edu/slides-pdf/chapter04a.pdf
- Slides Busca Exaustiva/Cega: link.
- Slides Busca Heurística: link.
  - A*
    - https://www.geeksforgeeks.org/a-search-algorithm/
    - https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html
- Slides Busca Local: link.
- Slides Algoritmos Genéticos: link.

```
função AGENTE-RESOLVEDOR-PROBLEMAS(percepção)
    variáveis: estado, a concepção do agente do estado atual do mundo
               seq, uma sequência de ações, inicialmente vazio
               problema, a formulação de um problema
               objetivo, um objetivo, inicialmente nulo
               ação, a ação mais recente, inicialmente nenhuma

    estado = ATUALIZA-ESTADO(estado,percepção)
```

```
      se seq está vazio então
          objetivo = FORMULAR-OBJETIVO(estado)
          problema = FORMULAR-PROBLEMA(estado, objetivo)
          seq = REALIZA-BUSCA(problema)
      ação = AÇÃO-DA-REGRA(regra)
      seq  = REPOUSO(seq)
      retornar ação;
```

Abstrato

```
função BUSCA-EM-ÁRVORE(problema) retorna uma solução, ou falha
     inicializa a fronteira usando o estado inicial do problema
     loop
         se a fronteira está vazia então
             escolha uma folha e remova ela da fronteira
             retornar falha
         se o nó contém um estado buscado como objetivo então
              expande o nó escolhido, adicionando os nós resultantes na
                                      fronteira (sequência de ações)
             retornar solução correspondente
```

Algorítmo menos abstrato

```
função BUSCA-EM-ÁRVORE(problema, fringe) retorna uma solução, ou falha
     fringe = INSERE(CRIAR-NO(ESTADO-INICIAL[problema]),fringe)
     loop
         se fringe está vazio então
             nodo = REMOVE-FOLHA(fringe)
             retornar falha
         se TESTA-OBJETIVO(problema) no ESTADO(nodo)= SUCESSO então
              fringe  = INSERIR-TODOS(EXPANDIR(nodo,problema),fringe)
             retornar nodo
```

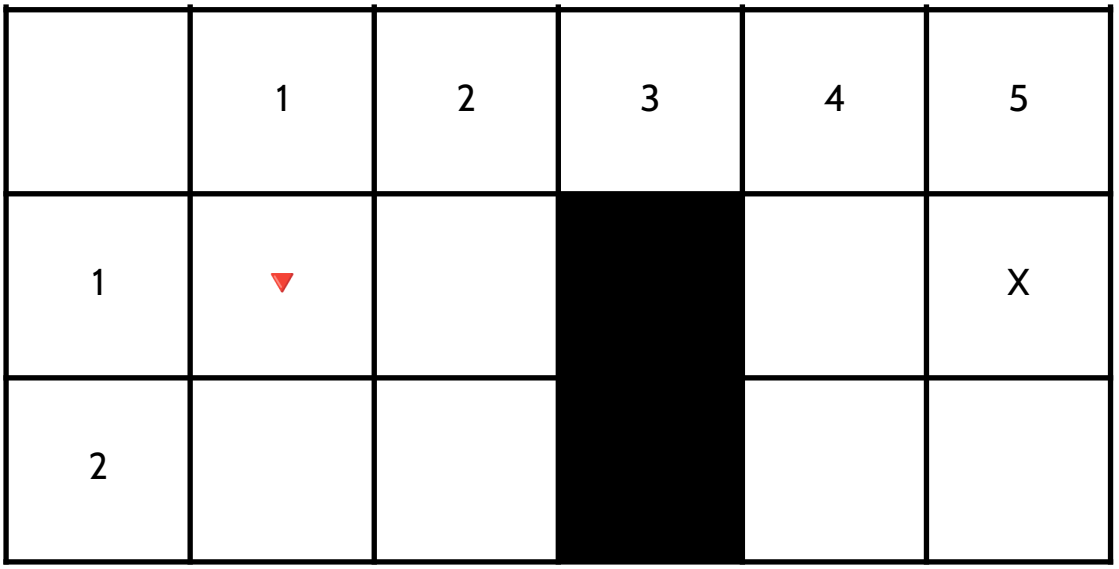
Abstrato

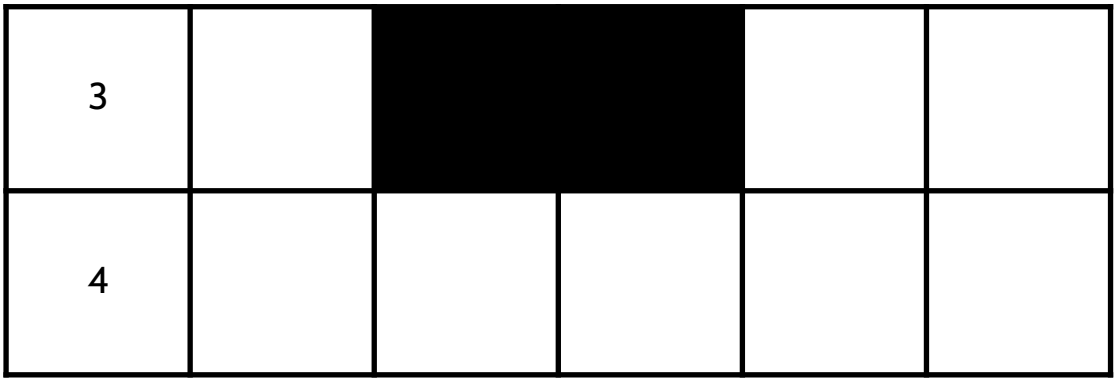
```
função BUSCA-EM-GRAFO(problema) retorna uma solução, ou falha
     inicializa a fronteira usando o estado inicial do problema
     inicializa o conjunto de nós de exploração como vazio
     loop
         se a fronteira está vazia então
             escolha uma folha e remova ela da fronteira
             retornar falha
         se o nó contém um estado buscado como objetivo então
              adiciona o nó no conjunto de nós explorados
             expande o nó escolhido, adicionando os nós resultantes na
                                      fronteira (sequência de ações)
                 somente se o nó já não está está na fronteira ou no
                 conjunto de nós explorados
             retornar solução correspondente
```

```
função GENETIC-ALGORITHM(populacao,FITNESS-FN) retorna um individuo
        entrada: populacao, um conjunto de indivíduos
                 FITNESS-FN, uma função que mede o fitness de um
                             indivíduo

        repita
            nova_populacao = conjunto vazio
              para i de 1 até SIZE(populacao) faça
                x = RANDOM-SELECAO(populacao, FITNESS-FN)
                y = RANDOM-SELECAO(populacao, FITNESS-FN)
                crianca = REPRODUZIR(x,y)
                se (pequena probabilidade aleatória) então
                    crianca = MUTAR(crianca)
                adicionar crianca na nova_populacao
            populacao = nova_populacao
        ate (algum indivíduo estar apto o suficiente ou o tempo decorrido)
        retornar o melhor indivíduo em populacao, segundo o FITNESS-FN

função REPRODUZIR(x,y) retorna um individuo
        entrada: x,y, indivíduos progenitores

      n = LENGTH(x)
      c = um número aleatório de 1 a n
      retornar APPEND(SUBSTRING(x, 1, c),SUBSTRING(y, c + 1, n))
```

Exercícios em sala de aula:

1. Considere o problema usado no slide de busca heurística abaixo e os códigos fonte de algoritmos de busca disponibilizados. Modele o problema usando grafos ou matriz e aplique um dos algoritmos disponibilizados para fazer o agente chegar no destino.
    a. link dos slides: link;
    b. link do código-fonte: link.





2. Se faz necessário resolver o problema de ordenamento de dados em um vetor. Resolva esse problema usando algoritmos genéticos. Para tanto considere:
    a. Código-fonte do exemplo de algoritmos genéticos: link.
    b. Slides algoritmos genéticos: link.
    c. Passos a serem pensados:
        i. Como seria a função objetivo (fitness)?
            1. 95450*x - x*x (função de avaliação)
        ii. Como representar os dados em um cromossomo?
            1. Representar de forma binária de 32bits
        iii. Como gerar a população inicial?
        iv. Como selecionar os indivíduos para reprodução?
        v. Como fazer o corte para gerar os novos indivíduos?
        vi. Como fazer a mutação?
        vii. Qual a condição de parada?

2.4 Teoria dos jogos e busca competitiva

Links de Referência:

- Slides de busca competitiva da disciplina: link.
- Slides de teoria dos jogos: link.
- Slides Busca Competitiva - Paulo Engel (UFRGS): link.
- Slides de Teoria dos Jogos de Ana Bazzan (UFRGS): link.
- Slides de Teoria dos Jogos de Fernando J. Von Zuben (Unicamp): link.

UNIDADE III - Agentes inteligentes

Links de Referência:

- Slides Agentes Russel: http://aima.eecs.berkeley.edu/slides-pdf/chapter02.pdf
- Slides Mono-Agentes, Disciplina: link.
- Slides Multiagente: link.

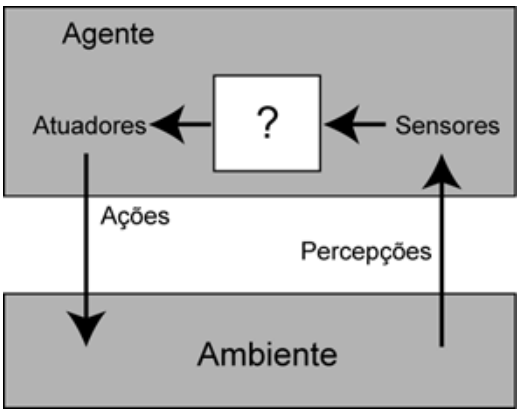
3.1 Conceito de agentes, ambiente e racionalidade

A técnica de agentes inteligentes é um elemento central dentro da inteligência artificial. De forma informal, podemos dizer que eles são as entidades de software que reagem ao ambiente e executam ações. Apesar de muitas vezes não ser óbvio reconhecer o que são agentes, eles já estão bastante inseridos na ferramentas da computação atuais.

Na internet, por exemplo, os bots, os navegadores, ferramentas de monitoramento da internet e os webservices são agentes. Em jogos de computador, todos os personagens não jogadores que interagem com o usuário de forma autônoma são agentes. O software que reconhece a voz do usuário ao dizer "Ok Google" também é um agente.

A maioria dos serviços em background tanto nos desktops quanto nos smartphones são agentes. Até mesmo pensando em cenários mais próximos da indústria 4.0, o software que controla um robô, um carro autônomo, ou mesmo uma montadora autônoma em uma fábrica são agentes.

Uma definição mais precisa seria: *"um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores (seja em software como em hardware) e de agir por intermédio de atuadores"* [RUSSELL e NORVIG, 2012]. A generalidade de tal definição permite que existam os mais diversos modelos de agentes fazendo usos das diferentes técnicas, das quais detalhamos a seguir. Apesar disso, podemos representar graficamente essa a estrutura comum entre eles seguindo a imagem representada



É importante ressaltar também que ao se modelar um software baseado em agentes é possível desenvolvê-lo pensando em usar um único agente ou com múltiplos agentes. Um programa mono agente, como por exemplo, um jogo de xadrez ou um serviço de monitoramento de memória, é desenvolvido pensando somente na interação dele com o mundo, incluindo o usuário. Por outro lado, um sistema multiagente deve incluir em suas interações a possibilidade de cooperar, competir ou comunicar com outros agentes em suas avaliações. Por exemplo, múltiplos programas que fazem roteamento ad hoc estarão competindo pelos pontos de acesso sem fio, que possuem vazão limitada pensando em maximizar suas próprias métricas.

Existem diversos modelos conhecidos de agentes, vamos tratar alguns deles de maneira superficial. Para maiores detalhes sugiro a leitura do capítulo 2 do livro do Russell e Norvig (2012) e a análise dos códigos fontes em java do livro (link).

Para exemplificação, uma função representa um agente genérico pode ser resumida através do pseudocódigo abaixo.

```
função AGENTE-GENÉRICO(percepção)
            ação = DECIDE-AÇÃO(percepção)
            retornar ação;
```

Em suma um agente recebe a percepção decide de alguma forma que ação tomar e retorna uma ação. Esta função pode ser executada dentro de um ciclo infinito em um fluxo de execução independente ou mesmo pode ser executado ou mesmo no fluxo principal e ainda assim será um agente. A diferença entre os diversos modelos de agente está no processo de decidir a ação. Vamos aos demais modelos.

Agente Dirigido por Tabela: um agente baseado em tabela é um agente que mapeia as sequências de percepções através do uso de uma tabela como forma para indexar as ações a serem executadas. O pseudo código abaixo demonstra e seu funcionamento. Apesar desse modelo ser intuitivo esse modelo não escala apropriadamente conforme o tamanho do problema cresce, pois seria necessário um grande tempo para construir a tabela. Além disso, esse modelo retira a autonomia do agente reduzindo a capacidade dele em se adaptar ao ambiente. Mesmo usando aprendizagem, seria necessário um longo tempo para aprender a tabela de entrada.

```
função AGENTE-DIRIGIDO-POR-TABELA(percepção)
     variáveis: percepções, uma sequência, inicialmente vazia
                tabela, uma tabela de ações, indexada por sequências
                       de percepções, inicialmente completamente
                       especificada

     anexar percepção ao fim de percepções
     ação = ACESSAR(percepções,tabela)
     retornar ação;
```

Agente Reativo Simples: realiza as ações considerando somente o estado atual de percepção ignorando as percepções históricas. Desta forma, as percepções são recebidas pelo agente, o agente então interpreta essas ações e obtém um estado. Este estado é avaliado através de uma lista de regras condição-ação (“se isto então ação”) resultando em uma ação. Este tipo de agente é mais apropriado para ambientes completamente observáveis onde não há incerteza. Um exemplo em pseudocódigo seria:

```
função AGENTE-REATIVO-SIMPLES(percepção)
```

```
        variáveis estáticas: regras, um conjunto de regras condição-ação

        estado = INTERPRETAR-ENTRADA(percepção)
        regra = REGRA-CORRESPONDENTE(estado, regras)
        ação = AÇÃO-DA-REGRA(regra)
        retornar ação;
```

Agente Reativo Baseado em Modelo: é um agente que mantém algum tipo de estado interno que dependa do histórico de percepções e assim reflita pelo menos alguns dos aspectos não observados do estado atual. Para tanto o modelo descreve como o mundo evolui independentemente do agente e como as ações do próprio agente afetam o mundo. Cabe ressaltar que este tipo de agente ainda tem problemas para decidir corretamente em casos de incerteza, desta forma o estado obtido/inferido pelo agente seria melhor interpretado como algo semelhante a "como o mundo me parece agora" e não "como o mundo está agora". Um exemplo em pseudocódigo deste agente seria:

```
função AGENTE-REATIVO-BASEADO-EM-MODELOS(percepção)
     variáveis: estado, a concepção do agente do estado atual do mundo
                modelo, uma descrição de como o próximo estado depende
                        do estado atual e da ação
                regras, um conjunto de regras condição-ação
                ação, a ação mais recente, inicialmente nenhuma

     estado = ATUALIZA-ESTADO(estado, ação, percepção, modelo)
     regra = REGRA-CORRESPONDENTE(estado, regras)
     ação = AÇÃO-DA-REGRA(regra)
     retornar ação;
```

Em relação especial a técnica de regras condição-ação, é bem conhecido na literatura que conforme o programa cresce em complexidade a quantidade de regras que tem de ser implementadas aumenta consideravelmente, tornando-se uma tarefa mais trabalhosa e suscetível a erros de programação. Para casos mais complexos, é indicado algumas outras técnicas no lugar da função REGRA-CORRESPONDENTE para encontrar a ação. Sendo elas:

- **Agente Baseado em Objetivo**: as vezes além da informação do estado atual, um agente precisa da informação se ele vai conseguir atingir seu objetivo através da ação, ou de um conjunto de ações. Para tanto Agentes Baseados em Objetivo utilizam técnicas de busca e planejamento, os quais são subáreas de IA. Embora o agente pareça menos eficiente, ele é mais flexível porque o conhecimento que apoia suas decisões é representado de maneira explícita e pode ser modificado.
- **Agente Baseado em Utilidade**: em algumas situações somente a informação se um conjunto de ações leva ao objetivo não é o suficiente para expressar o quão bom para o agente é um plano. Ou seja, agentes baseados em utilidade conseguem estimar o quão boa é uma ou mais soluções, selecionando sempre a com o melhor valor (ou utilidade). Desta forma é possível avaliar diferentes métricas, como uso de bateria,

tempo de resposta, tempo para chegar ao objetivo, uso de memória, etc, ao fim obtendo um valor de utilidade. Este tipo de função é bastante adequada para ser usada em casos do mundo real, onde há incerteza, estocasticidade e uma observalidade parcial.

- **Agente com Aprendizagem**: é um tipo de agente utilizado em ambiente que necessitam de exploração e adaptação. O grande benefício é que no decorrer do tempo, novos caminhos e configurações podem ser encontrados durante a exploração. Para tanto, é necessário utilizar um componente para gerenciar o aperfeiçoamento do aprendizado (o que o agente sabe), outro componente para calcular o desempenho das ações aprendidas no ambiente externo. Além disso, há dois mais componentes o crítico, que determina que algo deve ser mudado para funcionar melhor no futuro (comportamento proativo) e o gerador de problema, responsável por sugerir ações que levam a resultados não conhecidos para explorar novas possibilidades para aprendizado.

Além da questão de tomada de decisão do agente cabe ressaltar que existem outras diversas propriedades que podem ser implementadas em agentes, entre elas a mobilidade. Os agentes que atendam a essa propriedade são chamados de **Agentes Móveis**.

Agentes móveis são componentes de software autônomos com capacidade de migrar de um nó a outro em um sistema distribuído, mantendo o seu estado interno. Eles constituem um modelo de programação distribuída que oferece uma maneira alternativa para o projeto de programação de aplicações distribuídas. Este modelo se fundamenta na ideia de que, em vez de transmitir somente dados entre computadores de uma rede, como em sistemas distribuídos tradicionais, o código executável também possa ser transmitido.

3.2 Agentes Cognitivos e Reativos

Para definir arquiteturas internas dos agente é necessário saber qual o tipo de agente do qual se está tratando. Neste caso, um agente pode ser classificado como **cognitivo** ou **reativo**.

Oliveira (1996) define que, no caso de um agente cognitivo, temos uma íntima relação à idéia de agente racional, que tem como característica a capacidade de escolher as ações a executar, dentre as existentes em seu repertório, coerentemente com seus objetivos.

Quando um agente sempre escolhe a ação mais coerente com seus objetivos, podemos dizer que possui "racionalidade ideal", o que é computacionalmente inviável, pois requer que o agente tenha um conhecimento total e correto sobre o problema e o ambiente, sendo que a escolha poderia requerer um infinito número de passos de inferência

Um agente cognitivo é um agente racional que possui alguma representação explícita de seu conhecimento e objetivos.

Um agente **reativo** não necessariamente é um agente racional: seu comportamento pode ser definido através de um padrão estímulo-resposta (...) Um agente pode ser ’mais

cognitivo' do que outro, conforme o grau de racionalidade explícita de seu comportamento (Oliveira, 1996).

Nota Professor → melhorar essa seção próximo semestre. Utilizar umas fontes melhores e deixar mais didático o conteúdo. (Pegar parte que o Russell explica).

OLIVEIRA, Flávio Moreira de. Inteligência artificial distribuída. **Anais da IV Escola Regional de Informática**, p. 54-71, 1996.

3.3 Sistemas Multiagentes

Links de referência:

- Slides de multi-agentes da disciplina: link.
- Curso de multi-agentes: link.
- Seção sobre busca competitiva e teoria dos jogos: link.

Tópico extra sobre agentes: Ferramentas

Apesar de agentes serem facilmente implementáveis diretamente em uma linguagem de programação, algumas ferramentas podem ser usadas para dar suporte a programação de agentes:

- **Jade** (http://jade.tilab.com/): é um framework open source em Java usado para programação de aplicações baseadas em agentes distribuídos na rede em um ambiente peer-to-peer.
- **Tropos** (http://www.troposproject.eu/node/93): é uma metodologia de desenvolvimento de agentes open source com o objetivo de facilitar a modelagem e implementação de agentes inteligentes.
- **JACK** (http://aosgrp.com/products/jack/): é uma plataforma comercial para desenvolvimento de sistemas multiagentes. Ele se destaca por possibilitar uma coleção de agentes pré-prontos para agilizar o processo de modelagem e implementação. Além disso, permite programar direto em um framework BDI.
- **NetLogo** (https://ccl.northwestern.edu/netlogo/): é um framework e linguagem de programação open source utilizada para realizar modelagem e simulação de ambientes multiagentes. Além de ser usado na computação ele também é utilizado para simular ambientes econômicos, biológicos e sociais.
- **Aglets** (http://aglets.sourceforge.net/): é uma plataforma e biblioteca open source de programação em Java para facilitar o desenvolvimento de aplicações baseadas em agentes móveis.

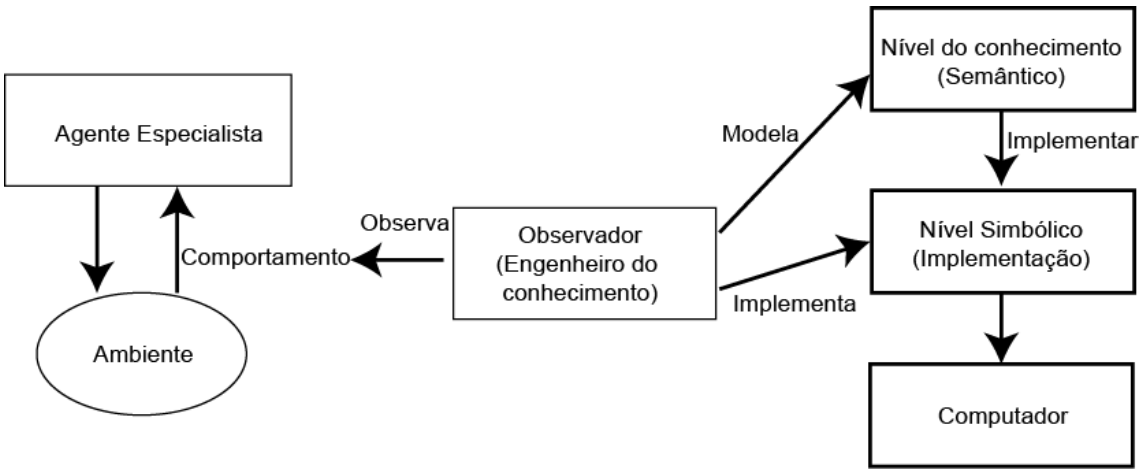
UNIDADE IV - Sistemas Especialistas

4.1 Histórico e conceitos de sistemas especialistas
4.2 Bases de conhecimento
4.3 Raciocínio encadeado para frente e para trás

Um sistema especialista é uma forma de sistema baseado no conhecimento especialmente projetado para emular a especialização humana de algum domínio específico e armazená-lo em uma base de conhecimento. O objetivo é extrair o conhecimento de especialistas humanos na resolução de problemas específicos para que seja usado por programas. A grande vantagem é que uma vez que este conhecimento seja interpretado por uma máquina, é possível facilitar ou mesmo substituir o trabalho de um especialista em dada tarefa, o que é uma vantagem também para a organização/empresa.

Além disso, o conhecimento formalizado usando sistemas especialistas ajudam a especificar padrões para computação. Alguns exemplos de casos seriam o HTML e em aprendizagem de máquina. Em HTML todos os atributos e elementos são formalizados por linguagens que permitem verificar se um esquema está corretamente formado, ou mesmo se um atributo é válido ou não. Em aprendizagem de máquina esta técnica pode ser usada para definir/identificar atributos dependentes que sejam relevantes para o processo de inferência antes de iniciar a coleta de dados. Por exemplo, através de tal análise seria possível identificar se uma nova variável possui correlação com a previsão de tempo, o que determinaria a necessidade ou não de monitorar ela e adicioná-la na base de treino do algoritmo de aprendizagem.

O processo para extração do conhecimento segue em geral o modelo de Newell (1982), representado pela imagem abaixo. A pessoa responsável para extrair o conhecimento e modelá-lo é conhecido como engenheiro do conhecimento. Para tanto existem diversas técnicas que podem ser usadas: desde entrevistas a observação do comportamento do especialista no trabalho. Contudo, isto não é uma tarefa fácil devido a fatores como: o especialista se sente desconfortável e evita tocar em certos detalhes; o conhecimento do especialista se tornou tácito (automático) e ele não consegue expressar todos os detalhes pertinentes de forma escrita, ou mesmo em entrevista; ou mesmo o conhecimento sobre os termos e conceitos do assunto que o Engenheiro do Conhecimento está modelando não são suficientes para ele conseguir abstrair o problema corretamente, resultando em retrabalho.



Através das observações encontradas com o especialista, o próximo passo é realizar a modelagem do conhecimento no **nível de conhecimento**. O nível de conhecimento utiliza-se ferramentas que expressam a semântica das observações encontradas através de modelagens conceituais ou mesmo ontológicas. Como tais modelagens não executáveis para em um computador é necessário que o engenheiro de conhecimento converte tal modelagem em um **nível simbólico**, formalizando ele ao nível que um computador possa interpretar.

Um dos grandes desafios da área é que em cada novo nível de representação de conhecimento parte da semântica do conhecimento se perde, principalmente pela limitação da expressividade das linguagens. Por exemplo, a fala de um especialista não é capaz de englobar todo o conhecimento e semântica que ele possui sobre o assunto; de forma semelhante, a modelagem possivelmente não conseguiria expressar todo o conhecimento passado de forma falada em um modelo conceitual. Um modelo conceitual, por sua vez, não consegue ser expressado completamente com a mesma semântica ao ser formalizado matematicamente para torná-lo executável por um computador.

Referências locais:

- NEWELL, Allen. The knowledge level. **Artificial intelligence**, v. 18, n. 1, p. 87-127, 1982.
- WATERMAN, D.A., **A Guide to Expert Systems** , Addison-Wesley Publishing Company, 1986.

(Tópico Extra) Ferramentas e linguagens utilizadas para modelagem de bases de Conhecimento:

- CMAP tools (https://cmapcloud.ihmc.us/): ferramenta de modelagem de diagramas conceituais. É uma ferramenta bem geral e permite iniciar a modelagem de conhecimento de maneira sem formalismo.
- Protegé (https://protege.stanford.edu/): é uma ferramenta open-source para edição e modelagem de ontologias. Utiliza o padrão OWL (Web Ontology Language) de linguagem de representação.

- RDF (Resource Description Framework): Mantido pela W3C (https://www.w3.org/RDF/) é a linguagem para representação de recursos na web.
- LD (Lógica de Descrição - http://dl.kr.org/): uma linguagem formal baseada em lógica. Descendente de redes semânticas e do sistema KL-ONE.
- Mais materiais: http://www.cs.man.ac.uk/~horrocks/Slides/index.html

UNIDADE V - Aprendizagem de Máquina

Links de referência:

- Slides da disciplina: link.
- Canal Serafim Nascimento: link.
- Canal Marciel Dantas: link.
- Videos de algoritmos em C: link.
- Canal Zurubabel: link.
- Canal muito bom aprendizagem em jogos de PC: link.

5.1 Aprendizado de máquina

A aprendizagem de máquina pode ser descrita como a capacidade de um agente de software melhorar seu comportamento através do estudo diligente de experiências passadas [RUSSELL e NORVIG, 2012]. Um dos principais usos é o reconhecimento de padrões para inferência, o que envolve reconhecimento de imagens, linguagem natural e fala, diagnóstico médico, classificação de spam, locomoção de robôs, entre outras formas de inferência indutiva.

Os algoritmos de aprendizagem possuem duas fases, uma de treino, ou exploração, utilizado para gerar ou ajustar o modelo de inferência e a execução, usada para aplicar o modelo obtido. Uma das características mais marcantes é que esses algoritmos são muito dependente dos dados que ele utiliza para treinar seu modelo de inferência, onde poucos dados são mais suscetíveis a erros. Cabe ressaltar que mesmo possuindo um grande número de dados, cabe ressaltar que não é garantido que todas as alternativas estejam contidas neles, por isso em geral há sempre uma taxa de erro associada a cada modelo gerado com treinamento. De acordo com Gollapudi et al (2016), o uso de tais conjuntos de dados de experiências passadas podem ser agrupados nos paradigmas de aprendizagem abaixo:

- *Aprendizado supervisionado*: São apresentadas ao computador exemplos de entradas e saídas desejadas a partir de um dataset relacionado com o problema alvo. No dataset a saída do mapeamento é chamada de “classe”, sendo obrigatoriamente um dado discreto, ou seja, com saídas conhecidas finitas. O objetivo é aprender uma regra geral que mapeia as entradas para as saídas.
- *Aprendizado não supervisionado*: Nenhum tipo de etiqueta é dado ao algoritmo de aprendizado, deixando-o sozinho para encontrar estrutura nas entradas fornecidas. O aprendizado não supervisionado pode ser um objetivo em si mesmo (descobrir novos

padrões nos dados) ou um meio para atingir um fim. Esse tipo de técnica é comumente utilizado em áreas como a mineração de dados, por exemplo.

- *Aprendizado semi-supervisionado*: utiliza datasets que possuem ambos dados sem rótulo e dados com rótulo para gerar seus modelos. Contudo, é importante ressaltar para que este tipo de problema seja resolvido com sucesso é necessário atender algumas premissas para dos dados sem rótulo de acordo. Estas premissas são definidas pelo algoritmo de aprendizagem utilizado. Esta forma de aprendizagem tem motivação na forma humana de pensar.
- *Aprendizado por reforço*: É utilizada em ambientes dinâmicos onde o software deve ajustar seu comportamento em tempo de execução para alcançar um determinado objetivo. Para tanto é atribuído ao agente de software feedbacks de suas ações premiando ou punindo, na medida em que é navegado o espaço do problema.
- *Deep Learning (Aprendizagem profunda)*: É utilizada para construir modelos de rede neural mais complexos para problemas de aprendizagem semi-supervisionada, operando com poucos dados rotulados. Em comparação as redes neurais artificiais de uso geral, as redes neurais usadas nesse tipo de problema são mais avançadas sendo capazes de trabalhar em grandes bases de dados. Alguns exemplos desses algoritmos são: Convolutional Networks; Restricted Boltzmann Machine (RBM); Deep Belief Networks (DBN); Stacked Autoencoders. Mais informações acessar o link: http://deeplearningbook.com.br/capitulos/.

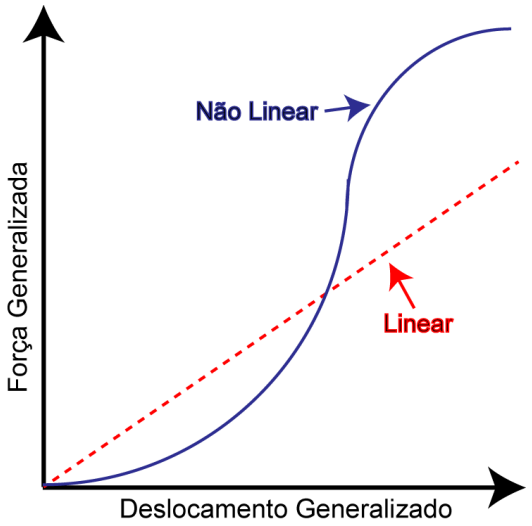
Cada um dos paradigmas acima estão relacionados a tarefas de aprendizagem. Entre as tarefas de aprendizagem, isto é, a tarefa que pretendemos submeter um dado algoritmo de aprendizagem, podemos comumente categorizar nas tarefas abaixo citadas. Contudo, cabe ressaltar que a maioria dos algoritmos não é adequado para todos esses tipos de tarefas.

- *Classificação*: entradas são divididas em duas ou mais classes, e o aprendiz deve produzir um modelo que vincula entradas não vistas a uma ou mais dessas classes (classificação multi-etiquetada). Isso é tipicamente abordado de forma supervisionada. A filtragem de spam é um exemplo de classificação, em que as entradas são as mensagens de emails (ou outros) e as classes são "spam" ou "não spam".
- *Regressão (conhecido também por: forecasting ou prediction)*, também um problema supervisionado, as saídas são contínuas, em vez de discretas. Um exemplo disso é a predição de temperatura ou mesmo do clima no dia seguinte.
- *Clustering*: divide um conjunto de entradas é em grupos. De maneira diferente da classificação, os grupos não são conhecidos previamente, tornando o clustering uma tarefa tipicamente não supervisionada.
- *Simulação*: neste tipo de problema, os cientistas de dados combinam diversas técnicas de aprendizagem de máquina (supervisionada, não-supervisionada, ...) para realizar a análise de casos que possuam muitas incertezas com o objetivo de suporte a tomada de decisão estratégica (humana). Alguns exemplos de casos de simulação: predição de ações no mercado; entender o comportamento de consumidores; busca na web e extração de informação.

- *Otimização*: é um caso que algoritmos de aprendizagem de máquina são usados para encontrar ou definir a melhor solução para um problema. Isto pode ser feito através da análise dos dados, resultando tanto ajustes de parâmetros dos algoritmos, como o uso de outras técnicas para suporte.

Além disso, é importante notar que os algoritmos de aprendizagem podem ter diferentes tipos de comportamento em seu modelo: comportamento linear e não linear. Os algoritmos lineares geram um modelo linear, resultando em uma reta de inferência. Ele pode ser muito útil em casos específicos com poucas variáveis, contudo tende a muitos erros caso tenha de considerar muitas variáveis. Algoritmos de Regressão Linear e Regressão logística são exemplos de algoritmos que geram modelos lineares. Os algoritmos não-lineares permitem resolver problemas mais próximo do mundo real onde apresenta múltiplos caminhos e destinos, desencadeando em múltiplos finais. Algoritmos de redes neurais e redes bayesiana são exemplos de algoritmos que podem resolver problemas não-lineares.

Para melhor entendimento das diferenças, clique aqui para visualizar um exemplo visual.



Outro tópico que tenho de colocar são os modelos algoritmos , o tipo (grandes grupos) deles e para que cada um serve melhor ser utilizado. (pegar do livro do GOLLAPUDI e do WITTEN). Colocar dentro de apredizagem supervisionada, não supervisionada..

5.1.1 Modelos de Representação de Conhecimento

Existem diversos modelos de representação do conhecimento para aprendizagem, cada um possui características favoráveis em determinadas condições. Sendo as principais representações e alguns de seus algorítmos:

Modelos Matemáticos:

- Regressão linear / logística
- Redes Neurais
- Máquinas de Vetores de Suporte, …

Modelos Simbólicos
- Árvores de Decisão
- Regras em lógica proposicional ou de 1ª ordem
- Redes Semânticas, …

Modelos “Lazy”
- K-NN
- Raciocínio Baseado em Casos (CBR), …

Modelos Probabilísticos
- Naïve Bayes
- Redes Bayesianas
- Misturas de Gaussianas
- Modelos Ocultos de Markov (HMMs), …

Repositório de Dados na Web
- UCI data repository
    - http://archive.ics.uci.edu/ml/
- Kaggle
    - https://www.kaggle.com/

Referências Locais

- GOLLAPUDI, Sunila. **Practical Machine Learning**. Packt Publishing Ltd, 2016.
- WITTEN, Ian H. et al. **Data Mining: Practical machine learning tools and techniques**. Morgan Kaufmann, 4ª edição, 2016.
- BRAGA, Antonio de Padua; LUDERMIR, Teresa Bernarda: **Redes Neurais Artificiais: Teoria e aplicações**. Rio de Janeiro: LTC, 2017.

5.2 Métricas para Avaliação de Algoritmos de Aprendizagem

Em aprendizagem de máquina existem algumas métricas importantes que são comuns a maioria dos algoritmos para que seja possível saber se o algoritmo está próximo do resultado desejado. Essas métricas são divididas pelo tipo de problema sendo tratado. Nesta

seção iremos aprender sobre os dois grupos de problemas mais frequentes: classificação e regressão.

5.2.1 Métricas para Problemas de Classificação

Antes entrar nas métricas é necessário abordar o conceito de matriz confusão.

A matriz confusão é usada em casos de classificação para calcular os resultados corretos e os errados de classificação, quando um modelo de inferência é gerado. Esses dados utilizados para testar o classificador são em geral parte do dataset que não foi utilizado no treino. Em alguns casos é possível dividir os dados de treino para a parte de teste. Para explicar melhor como ela funciona considere a tabela abaixo com duas variáveis: Positivo e Negativo. Os valores verdadeiros são expressos pelas linhas em amarelo. Os valores classificados na etapa de teste são expressos nas colunas em azul.

A classificação está correta quando os valores são classificados devidamente tal qual estão rotulados, estes representados em verde na tabela, correspondente à diagonal principal. Assim, se um valor rotulado como *Positivo* for classificado como *Positivo*, ele é um Verdadeiro positivo (VP), de forma análoga, se o valor rotulado como *Negativo* for classificado como *Negativo* podemos dizer que ele é um Verdadeiro negativo (VN). Cabe ressaltar que caso houverem mais que duas colunas, o VN é calculado somando todas as classificações corretas que não estão rotulados com o rótulo sendo avaliado como VP.

Nos casos em vermelho, temos as classificações incorretas. Se um valor Rotulado como *Positivo* for classificado como *Negativo*, ele é um Falso negativo (FN). De forma semelhante, se um valor rotulado como *Negativo* for classificado como *Positivo* chamamos ele de Falso positivo (FP). Caso exista mais que dois campos rotulados, podemos dizer informalmente que todos os valores acima da diagonal principal são FN e todos os valores abaixo da diagonal principal são FP.

Matriz confusão		**Valores Classificados/Preditos**	
		Positivo (P)	*Negativo (N)*
Valores Verdadeiros	*Positivo (P)*	Verdadeiro positivo (VP)	Falso negativo (FN)
	Negativo (N)	Falso positivo (FP)	Verdadeiro negativo (VN)

Mais informações:

- Descrição WikiPedia para matriz confusão: link.
- Descrição do blog de mineração de dados: link.A partir da tabela confusão podemos calcular diversas métricas para ajudar na escolha de um algoritmo de aprendizagem de máquina.

Acurácia total: Indica a proporção de predições corretas, sem levar em consideração o que é positivo e o que é negativo. Esta medida é altamente suscetível a desbalanceamentos

do conjunto de dados e pode facilmente induzir a uma conclusão errada sobre o desempenho do sistema.

```
Acurácia = TOTAL DE ACERTOS / TOTAL DE DADOS NO CONJUNTO
Acurácia = (VP + VN) / (P + N)
```

Precisão: proporção de exemplos positivos classificados corretamente entre todos aqueles preditos como positivos. Esta é uma medida que pode ser vista como uma medida de exatidão do modelo

```
precisão = ACERTOS POSITIVOS / TOTAL DE POSITIVOS
precisão = VP / (VP + FP)
```

Sensibilidade ou Recall: A proporção de verdadeiros positivos: a capacidade do sistema em predizer corretamente a condição para casos que realmente a têm. Número de exemplos classificados como pertencentes a uma classe, que realmente são daquela classe, dividido pela quantidade total de exemplos que pertencem a esta classe, mesmo que sejam classificados em outra. No caso binário, positivos verdadeiros divididos por total de positivos.

```
Sensibilidade = ACERTOS POSITIVOS / TOTAL DE POSITIVOS
Sensibilidade = VP / (VP + FN)
```

Especificidade: A proporção de verdadeiros negativos: a capacidade do sistema em predizer corretamente a ausência da condição para casos que realmente não a têm.

```
Especificidade = ACERTOS NEGATIVOS / TOTAL DE NEGATIVOS
Especificidade = VN / (VN + FP)
```

Eficiência: A média aritmética da Sensibilidade e Especificidade. Na prática, a sensibilidade e a especificidade variam em direções opostas. Isto é, geralmente, quando um método é muito sensível à positivos, tende a gerar muitos falso-positivos, e vice-versa. Assim, um método de decisão perfeito (100 % de sensibilidade e 100% especificidade) raramente é alcançado, e um balanço entre ambos deve ser atingido.

```
Eficiência = (SENS + ESPEC) / 2
```

Valor Preditivo Positivo (não tão essencial): A proporção de verdadeiros positivos em relação a todas as predições positivas. Esta medida é altamente suscetível a desbalanceamentos do conjunto de dados e pode facilmente induzir a uma conclusão errada sobre o desempenho do sistema.

```
Valor Preditivo Positivo = ACERTOS POSITIVOS / TOTAL DE PREDIÇÕES
POSITIVAS
Valor Preditivo Positivo = VP / (VP + FP)
```

Valor Preditivo Negativo (não tão essencial): A proporção de verdadeiros negativos em relação a todas as predições negativas. Esta medida é altamente suscetível a desbalanceamentos do conjunto de dados e pode facilmente induzir a uma conclusão errada sobre o desempenho do sistema.

```
Valor Preditivo Negativo = ACERTOS NEGATIVOS / TOTAL DE PREDIÇÕES
NEGATIVAS
Valor Preditivo Negativo = VN / (VN + FN)
```

5.2.2 Métricas para Problemas de Regressão

No caso da regressão temos ainda algumas métricas diferentes que devem ser consideradas. Para essas métricas considere as variáveis abaixo. Lembrando que ***n*** é a quantidade de instâncias em uma base de dados e caractere ***i*** o índice atual da instância avaliada.

$$y_i = valor\ real$$
$$\hat{y}_i = valor\ previsto$$

Mean Squared Error – MSE

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Talvez seja a mais utilizada, esta função calcula a média dos erros do modelo ao quadrado. Ou seja, diferenças menores têm menos importância, enquanto diferenças maiores recebem mais peso.

Existe uma variação, que facilita a interpretação: o Root Mean Squared Error. Ele é simplesmente a raiz quadrada do primeiro. Neste caso, o erro volta a ter as unidades de medida originais da variável dependente.

Mean Absolute Error – MAE

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Bastante parecido com MSE, em vez de elevar a diferença entre a previsão do modelo, e o valor real, ao quadrado, ele toma o valor absoluto. Neste caso, em vez de atribuir um peso de acordo com a magnitude da diferença, ele atribui o mesmo peso a todas as diferenças, de maneira linear.

Se imaginarmos um exemplo simples, onde temos apenas a variável que estamos tentando prever, podemos ver um fato interessante que difere o MSE do MAE, e que devemos levar em conta ao decidir entre os dois: o valor que minimizaria o primeiro erro seria a média, já no segundo caso, a mediana.

Mean Absolute Percentage Error – MAPE

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Este erro calcula a média percentual do desvio absoluto entre as previsões e a realidade. É utilizado para avaliar sistemas de previsões de vendas e outros sistemas nos quais a diferença percentual seja mais interpretável, ou mais importante, do que os valores absolutos.

Maiores informações:

- http://mariofilho.com/as-metricas-mais-populares-para-avaliar-modelos-de-machine-learning/

5.3 Problemas de inconsistência de dados

Existem algumas inconsistências de dados que devem ser observadas em projetos de aprendizagem de máquinas. Felizmente entender e identificar elas ajuda a evitá-las. São elas:

5.3.1 Under-fitting

Um modelo possui under-fitting quando ele não possui informação suficiente para gerar um modelo preciso. Isto é, a base de dadas utilizada para treinar o modelo de aprendizagem é muito pequena para ter uma acurácia viável. Além disso, caso o classificador seja muito rígido ou simples demais, o problema de under-fitting ocorrerá, não por causa da falta de dados, mas pelo algoritmo gerar um modelo inadequado para o conjunto de dados passado. Por exemplo, se um problema é não linear e tentarmos utilizar um algoritmo linear a taxa de erros será maior.

Para evitar esse problema é necessário avaliar se: (1) há dados o suficiente para se gerar um modelo consistente; (2) se o algoritmo de aprendizagem de máquina é adequado para os dados e o problema pretendido.

5.3.2 Over-fitting

Over-fitting é o oposto de under-fitting. Enquanto uma amostra não é apropriada para definir um modelo ótimo, uma grande base de dados também tem o risco de ter um modelo com dados com over-fit. Em termos mais simples, o modelo memoriza os dados de treinos ao invés de aprender a situação e gerar um modelo compatível com situações futuras com incerteza. Um modelo que possui overfit demonstra baixa performance com as menores flutuações de novos dados de forma exagerada. A principal razão é que os campos fora da curva não foram contemplados pelo modelo, ou mesmo devido ao tamanho da base alguns algoritmos eliminam os pontos fora da curva ao normalizar os dados. Este tipo de problema é comum, mas é corrigível.

Um erro comum em alguns casos iniciantes ocorre também devido ao mesmo dataset de treino usado nos testes para gerar as métricas.

Para evitar esse problema considere: (1) usar bases de dados diferentes da utilizada no treino, ou mesmo separe esse dataset em amostras menores para treino e teste, algumas ferramentas como o Weka, inclusive fazem isso automaticamente; (2) teste vários algoritmos e analise as diferentes métricas para verificar o mais adequado para o caso; (3) avaliar a base de dados com outras ferramentas para visualizar onde os pontos estão se concentrando.

5.3.3 Instabilidade de dados

Um problema que pode ocorrer devido ao erro manual ou a falta de interpretação de dados relevantes. Isso resultará em uma distorção dos dados, o que acabará resultando em um modelo incorreto, com muitos erros quando implantado. A forma de corrigir esse problema, é a necessidade de realizar um processo para corrigir os dados do problema, o que pode ser necessário recorrer a trabalhos humanos manuais, sendo, mesmo assim, suscetível a resultar na construção de um modelo incorreto.

5.3.4 Formatos de dados imprevistos

Aprendizagem de máquina é destinada a transação com novos danos de forma constante. Conforme o tempo passa novos formatos de dados e opções podem ser introduzidos entre os dados, os quais podem não ser suportados pelo sistema de aprendizagem, ou mesmo resultarem em uma imprecisão muito acentuada. Para tentar corrigir esse problema é possível: (1) construir um sistema para tratar esses novos formatos e converter em formatos compatíveis; (2) revisar o modelo de aprendizagem e gerar um novo modelo.

Mesmo assim, é uma prática de tempos em tempos de revisar o modelo de aprendizado para ter certeza se ele está tendo ou não esse problema.

5.4 Ferramentas

Existem diversas ferramentas no mercado para aprendizagem de máquina. Seguem algumas para se ter uma noção:

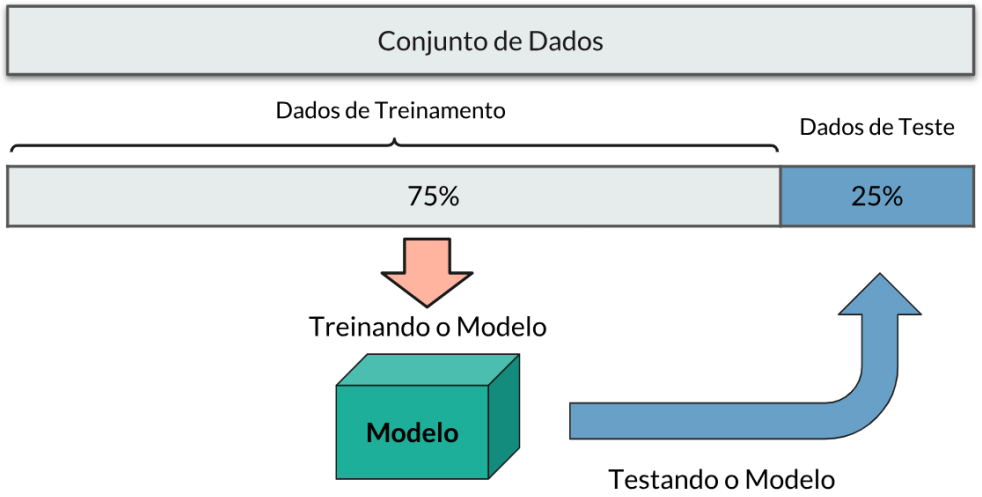
- *Apache Mahout* (https://mahout.apache.org/): é um projeto da Apache Software Foundation para distribuir implementação grátis de algoritmos de aprendizagem de máquina distribuídos e escaláveis. O foco inicial é tratar problemas de filtro colaborativo, clustering e classificação. Permite programar em java, contudo cabe ressaltar que é um projeto recente e diversos algoritmos ainda estão em processo de implementação.
- R (https://www.r-project.org/): é uma linguagem gratuita conhecida para execução de funções matemáticas e gerar gráficos. Entre as bibliotecas que podem tem usadas há diversas funções para aprendizagem de máquina (https://cognitiveclass.ai/courses/machine-learning-r/).
- Phyton (scikit-learn: http://scikit-learn.org/): é uma linguagem open-source com suporte a uma biblioteca para aprendizagem de máquina.
- TensorFlow (https://www.tensorflow.org/): é uma bilioteca open-source feita em Python especializada em aprendizagem de máquina
- Julia (https://julialang.org/): é uma linguagem focada em executar funções matemáticas em ambientes paralelos e distribuídos. Entre as bibliotecas há diversos algoritmos para aprendizagem de máquina.
- Weka (https://www.cs.waikato.ac.nz/ml/weka/): é uma ferramenta para aprendizagem de máquina e mineração de dados feita em Java. Ele pode ser usada como uma biblioteca de aprendizagem para ser executada dentro de outras aplicações Java.
- Brain.js (https://github.com/BrainJS/brain.js): é uma biblioteca javascript que provém suporte a algoritmos de rede neural e rede neural recorrente.
- Keras (https://keras.io/): O Keras possui uma sintaxe bastante clara, a documentação é muito boa (apesar de ser relativamente novo) e funciona com a principal linguagem para desenvolvimento de modelos de Deep Learning, Python. Keras é uma biblioteca de alto nível que funciona sobre o Theano ou Tensorflow (é configurável). Além disso, Keras reforça o minimalismo, você pode construir uma rede neural em apenas algumas linhas de código.
- Shogun (http://www.shogun-toolbox.org/): é uma das bibliotecas de aprendizagem de máquina mais antiga. Possui suporte a várias linguagens inclusive C++. O código pode ser acessado também via GitHub: link.

5.5 Aprendizado supervisionado

No aprendizado supervisionado são apresentadas ao computador exemplos de entradas e saídas desejadas a partir de um dataset relacionado com o problema alvo. O objetivo é aprender uma regra geral que mapeia as entradas para as saídas. Existem diversos algoritmos de aprendizagem supervisionada. Nas próximas seções exploraremos dois deles.

Porém, todos os tipos de algoritmos supervisionados, eles podem ser utilizados visando dois tipos principais de modelos de treinamento para o modelo de conhecimento: Treino-teste/ e Validação Cruzada. Mais informações: link.

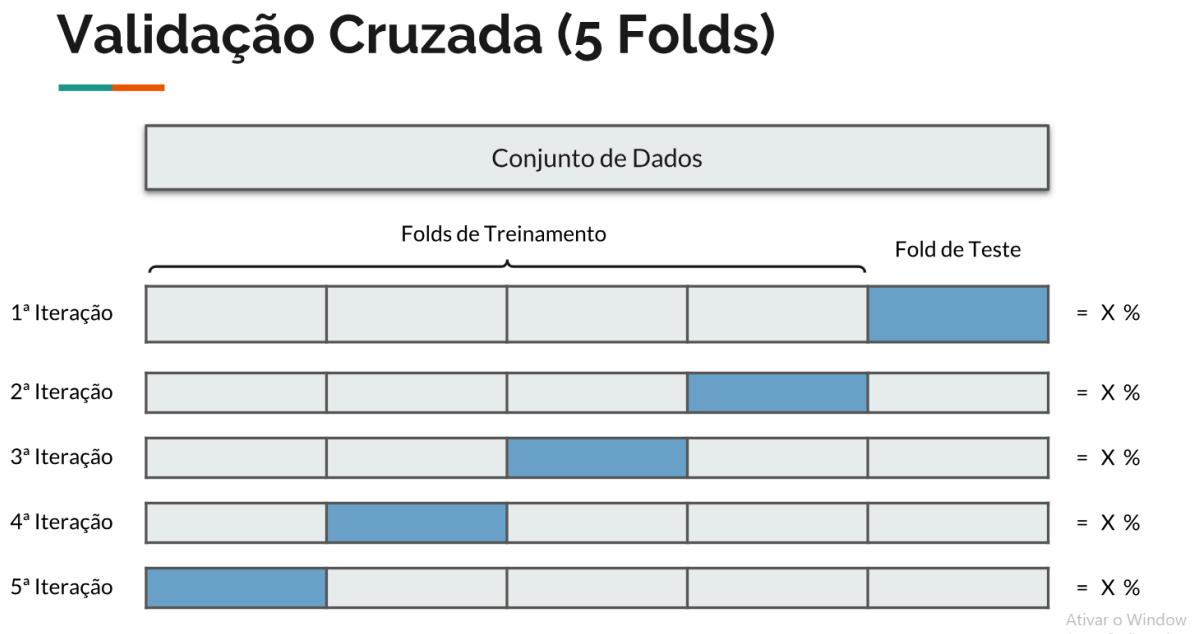
Train-test ou hold-out validation (Treino-Teste): O método de holdout reserva um conjunto para teste e o demais para treinamento. Geralmente se usa uma quantidade de ⅓ dos dados para teste, mas isso não é regra. O problema deste método consiste onde os exemplos podem não ser representativos. Em outras palavras, a classe pode não existir nos exemplos de teste. A figura abaixo demonstra um exemplo.



Cross Validation (Validação Cruzada): Existem vários tipos de cross validation, porém a mais comum é conhecida como k-fold cross validation. Neste método criamos um número k de amostras, sendo que cada amostra é deixada de lado enquanto o modelo treina com o restante delas. O processo de repete até que seja possível determinar a "qualidade" (depende da métrica em uso) de cada observação (chegando também em uma média geral). Os valores comuns para o número k de amostras são entre 5 e 10.

k-fold cross validation geralmente é usado quando não se tem dados suficientes para seguir com a estratégia mais simples de hold-out validation. Um outro tipo de cross validation é conhecido como repeated k-fold cross validation. Neste método o processo realizado na abordagem k-fold cross validation é repetido n vezes. No final, a qualidade do modelo é definida pelo desempenho médio de cada repetição. Este método é mais utilizado quando temos mais capacidade de processamento disponível.

A figura abaixo ilustra de forma mais didática esse método considerando o exemplo de 5-folds.

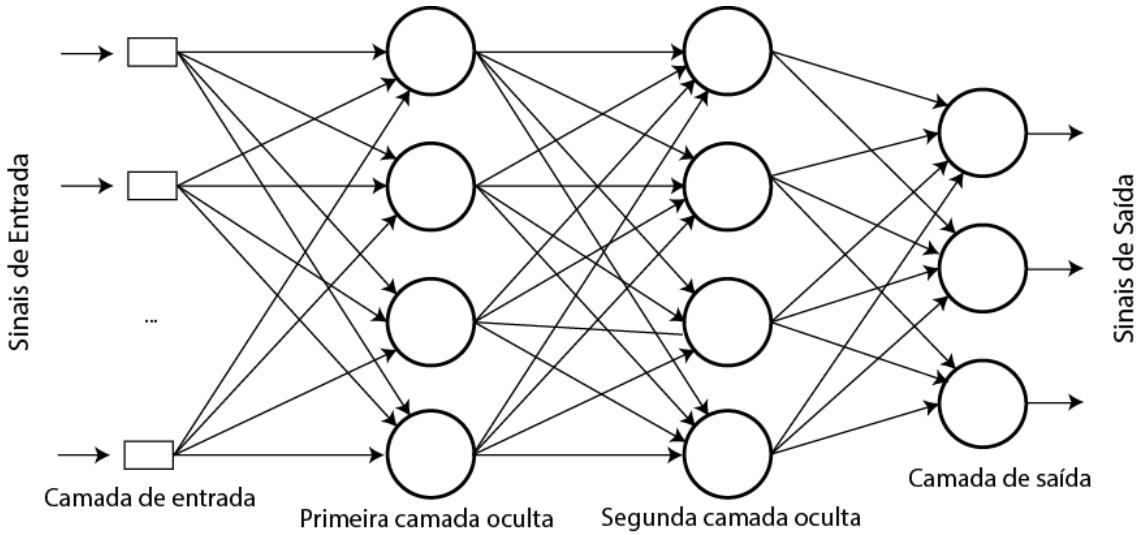


5.5.1 Rede neural artificial

Redes neurais artificiais (ANN) são uma classe de técnicas de reconhecimento de padrões usando funções matemáticas inspirado na estrutura biológica das redes de neurônios [GOLLAPUDI et al., 2016; BRAGA et al., 2007]. Uma ANN caracteriza-se por ser composta por elementos processadores, chamados neurônios, densamente interconectados que são capazes de adquirir conhecimento com o passar do tempo. O neurônio artificial adquire conhecimento através de seu relacionamento com os demais neurônios, baseado na repetição de um conjunto de soluções onde a saída de um neurônio da rede compõem a entrada de outro.

Esses métodos podem ser usados para resolver tanto problemas de regressão como problemas de classificação. Eles se relacionam com modelagem Deep Learning e tem muitos subcampos de algoritmos que ajudam a resolver problemas específicos. Uma das redes neurais mais utilizadas e populares é a multi-layer Perceptron. Vamos analisar este caso em maiores detalhes.

A figura abaixo ilustra uma rede neural de múltiplas camadas. O perceptron multicamadas (MLP) é uma rede neural semelhante ao perceptron simples, porém possui mais de uma camada de neurônios intermediárias (as camadas ocultas), estas representadas pela primeira e pela segunda camada oculta. Em casos em que não há a possibilidade de uma única reta separar os elementos, há o uso da MLP que, gera mais de uma reta classificadora. O aprendizado nesse tipo de rede é geralmente feito através do algoritmo de retropropagação do erro (backpropagation).



5.5.2 Exemplo 1 usando o Weka: Rede Neural

Para usar uma MLP para classificação em um programa em Java é necessário importar a biblioteca do weka.jar e especificar uma base de dados para treino. Vamos considerar a base de dados de exemplo disponível na instalação do Weka: C:\Program Files\Weka-3-8\data\iris.2D.arff

Esta base possui 3 atributos:

1. petallength: tamanho da pétala (varia de 1 a 6.9);
2. petalwidth: largura da pétala (varia de 0.1 a 2.5);
3. class: Classe do problema representando a espécie da flor com 3 possíveis valores (Iris-setosa,Iris-versicolor,Iris-virginica).

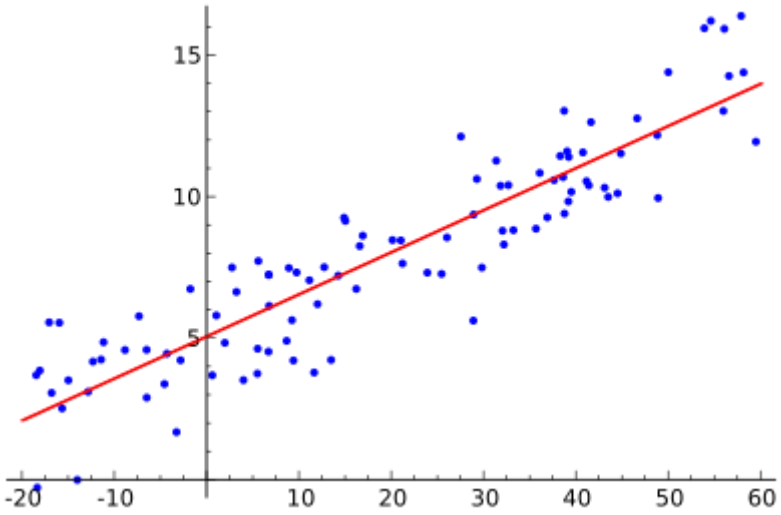
Para maiores detalhes sobre as métricas desta base acesse o menu exploração do weka e execute diferentes classificadores.

Código-fonte de exemplo: link.

5.5.3 Regressão linear simples

Regressão linear simples é um algoritmo que compara dois modelos; onde não há variáveis independentes; e forma uma linha usando a variável dependente. Para se entender melhor, observe o gráfico abaixo. As variáveis/modelos com dependência são representados pelos eixos X e Y. Ao ao aplicar o algoritmo de regressão linear simples obtém-se uma função matemática linear representada pela linha vermelha.

A partir da função, dado um valor qualquer de Y, é possível estimar qual é o valor de X, e vice versa, mesmo que este não estiver entre as variáveis os dados previamente avaliados. Este método é muito útil para modelos mais simples de predição (com poucas variáveis dependentes) e é eficiente computacionalmente. Contudo, como o próprio nome já indica, este algoritmo somente pode ser usado para regressão e em casos lineares.



5.5.4 Exemplo 2 usando o Weka: Regressão Linear

Para realizar uma regressão/predição usando Regressão Linear em um programa escrito em Java também é necessário importar a biblioteca do weka.jar e especificar uma base de dados para treino. De forma adicional, devido a instabilidades em algumas versões também será necessário importar as bibliotecas: core.jar; mtj.jar; e arpack_combined.jar. Clique no link oficial para download.

Vamos considerar a base de dados de exemplo de passageiros por mês disponível na instalação do Weka: C:\Program Files\Weka-3-8\data\airline.arff. Mensalmente uma empresa aérea registra o número de passageiros que usaram seus serviços na casa dos milhares usando dois atributos:

1. passenger_numbers: valor numérico, variando;
2. date: data do registro. Formato: 'yyyy-MM-dd' - datas entre 1949-01-01 a 1960-12-01.

Aparentemente a base tem comportamento linear. No código de exemplo vamos predizer: (1) o número de passageiros de acordo com a data passada e (2) a data de acordo com o número de passageiros. Link para o código de exemplo: link.

5.6 Aprendizado não-supervisionado

Na aprendizagem não-supervisionada nenhum tipo de etiqueta é dado ao algoritmo de aprendizado, deixando-o sozinho para encontrar estrutura nas entradas fornecidas. O aprendizado não supervisionado pode ser um objetivo em si mesmo (descobrir novos padrões nos dados) ou um meio para atingir um fim. Esse tipo de técnica é comumente utilizado em áreas como a mineração de dados, por exemplo (link para mais infos: link).

As principais atividades usando aprendizado não-supervisionado são:

- **Agrupamentos (Clustering)**: agrupar instâncias similares em aglomerados. As técnicas de agrupamento procuram semelhanças e diferenças num conjunto de dados e agrupam os registos semelhantes em segmentos ou clusters, de uma forma automática,

de acordo com algum critério ou métrica. Não é necessário definir os grupos nem os atributos que devem ser utilizados para segmentar o conjunto de dados. Por exemplo, um conjunto de dados pode conter vários clientes ricos sem filhos, e clientes com baixos rendimentos com um filho. Durante o processo de descoberta, esta diferença poderá ser utilizada para separar os dados em dois segmentos. No caso de existirem mais semelhanças e diferenças do género, o conjunto de dados pode ser ainda mais subdividido. Alguns algoritmos:

  - K-Means Clustering: link.
  - Cobweb: link.
  - Hierarchical Clustering: link.
  - Density Based Clustering: link e link

- **Associação**: Detecção de associações entre atributos. Como exemplo, eles podem solucionar problemas de análise de cesto de produtos, gerando modelos descritivos que permitem descobrir regras do género: "Os consumidores que adquirem pizzas têm uma probabilidade 3 vezes superior de adquirirem também queijo, do que aqueles que não compram". Este tipo de abordagem manipula a compra de um conjunto de produtos, como uma única transacção. O objectivo é o de encontrar tendências nas várias transações analisadas que possam ser utilizadas para entender e explorar padrões de compra natural. Esta informação pode ser utilizada para ajustar o estoque, modificar a disposição física dos produtos, ou lançar campanhas promocionais direcionadas. Exemplos de algoritmos:
  - Apriori: link, link.
  - FP-Growth: link;

O indutor analisa os exemplos fornecidos e tenta determinar se alguns deles podem ser agrupados de alguma maneira, formando Após a determinação dos agrupamentos, em geral, é necessário uma análise para determinar o que cada agrupamento significa no contexto problema sendo analisado.

5.7 Aprendizado por reforço

A aprendizagem por reforço é utilizada em ambientes dinâmicos onde o software deve ajustar seu comportamento em tempo de execução para alcançar um determinado objetivo. Para tanto é atribuído ao agente de software com feedbacks de suas ações premiando ou punindo, na medida em que é navegado o espaço do problema.

Falar um pouco sobre em geral e colocar Q-learning e Markov.

5.8 Exercícios Práticos

- Exercícios Supervisionados: link.
- Exercícios Por Reforço: link (ainda não feito).

UNIDADE VI - Redes Neurais (incluído em aprendizagem de máquina)
6.1 Conceitos iniciais
6.2 Topologia de redes neurais
6.3 Aplicações de redes neurais
6.4 Construção de redes a partir dos dados
6.6 Aplicações

UNIDADE VII - Algoritmos Genéticos, Conjuntos e Lógica Fuzzy

7.1 Computação evolutiva
7.2 Conjuntos Fuzzy (falta só isso)
7.3 Lógica Fuzzy
7.4 Aplicações

Referencial Bibliográfico

Bibliografia básica

- RUSSELL, Stuart; NORVIG, Peter. Inteligência artificial. Rio de Janeiro, RJ: Elsevier, 2004.
- FERNANDES, Anita Maria da Rocha. Inteligência artificial: noções gerais. Florianópolis, SC: Visual Books, 2005.
- SCHILDT, Herbert. Inteligência artificial utilizando a linguagem C. São Paulo: McGraw-Hill, 1989.

Bibliografia complementar

- BISHOP, Christopher M. Pattern recognition and machine learning. New York: Springer, 2006.
- HAYKIN, Simon S. Redes neurais: princípios e prática. 2. ed. Porto Alegre: Bookman, 2001.
- NASCIMENTO JÚNIOR, Cairo Lúcio; YONEYAMA, Takashi. Inteligência artificial em controle e automação. São Paulo, SP: Edgard Blücher, 2004.
- LUGER, GEORGE F. Inteligência Artificial 6ª edição. São Paulo: Pearson, 2013.
- REZENDE, S. O. Sistemas Inteligentes fundamentos e aplicações. Barueri SP: Manole, 2005.