CASSIO BERTRAND GASPARIAN COLELLO

INTELIGÊNCIA ARTIFICIAL APLICADA À RECUPERAÇÃO DE INFORMAÇÕES: UM MODELO PROBABILÍSTICO BASEADO EM ONTOLOGIAS

São Paulo
2009

CASSIO BERTRAND GASPARIAN COLELLO

INTELIGÊNCIA ARTIFICIAL APLICADA À RECUPERAÇÃO DE INFORMAÇÕES: UM MODELO PROBABILÍSTICO BASEADO EM ONTOLOGIAS

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Graduação em Engenharia

Área de concentração:
Engenharia Mecatrônica

Orientador: Prof. Dr. Fabio Gagliardi Cozman

São Paulo
2009

FICHA CATALOGRÁFICA

Colello, Cassio Bertrand Gasparian
Inteligência artificial aplicada à recuperação de informações: um modelo probabilístico baseado em ontologias / C.B.G. Colello. -- São Paulo, 2009.
70 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Recuperação da informação 2. Pesquisa 3. Ontologia 4. Inferência bayesiana (Inferência estatística) I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrô- nica e de Sistemas Mecânicos II. t.

RESUMO

Sistemas de recuperação de informação são amplamente utilizados. Tais sistemas são fundamentais para a localização de dados em grandes coleções de documentos (por exemplo, a Web). A proposta do presente trabalho é desenvolver um sistema baseado em um modelo de representação de conhecimentos, que realize uma busca não só com base nos termos inseridos pelo usuário, mas também pelo significado deles. Para tal, foi desenvolvida uma ferramenta que pode ser subdividida em três processos distintos. Primeiramente são extraídos conhecimentos de uma ontologia através dos quais será gerada uma Rede Bayesiana. Em seguida, quando o usuário realiza uma busca, o sistema expande tal pesquisa acrescentado novos termos e pesos, que serão calculados através de inferências realizadas na Rede Bayesiana. Finalmente, são exibidos os resultados ordenados de acordo com a sua relevância e representados em um gráfico no qual eles estarão agrupados de acordo com seus respectivos assuntos.

Palavras chave: Recuperação de informações, Expansão de pesquisa, Ontologia, Rede Bayesiana.

ABSTRACT

Information retrieval systems are widely used nowadays, such systems are essential to search huge data collections (for example, the Web). The goal of this project is to develop a system, based on a knowledge representation model that looks for the information not just by the user's entry but also by the semantic meaning of the terms retrieved. The way that this software works may be sub-divided in three parts. First, it parses the knowledge represented from an ontology and creates a Bayesian Network from the information acquired. Then, when the user conducts a search, the system expands the query by including terms and boosts, which will be calculated through inferences in the Bayesian Network. Finally, it shows the results ordered by relevance and prints a graph that represents how the documents found are related to each other.

Keywords: Information retrieval, Query expansion, Ontology, Bayesian Network.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

1 Introdução

Sistemas de recuperação de informação (RI) são aplicativos cuja função é possibilitar a localização de dados em um conjunto de documentos. Inicialmente eles foram desenvolvidos para organizar acervos de bibliotecas e publicações científicas. Recentemente, eles vêm se tornando cada vez mais necessários em função da complexidade dos bancos de dados e da expansão da Web.

Em geral eles são utilizados nos campos relacionados aos sistemas de informações ou tecnologia da informação. Isto porque nesta área eles são vitais uma vez que auxiliam na tomada de decisões em um negócio, principalmente ao proporcionar rapidez e concisão na coleta dos dados desejados.

Contudo, tais ferramentas também podem ser empregadas em outras áreas como na manufatura mecânica. Pode-se associá-las facilmente a ferramentas de CAPP (Planejamento de processos assistidos por computador) que se utilizam do conceito de tecnologia de grupo [2, 8], visando à redução do tempo e, consequentemente, dos gastos envolvidos no *setup* entre as operações através da classificação dos elementos envolvidos nos processos de manufatura (de acordo com as suas semelhanças).

Apesar das variadas aplicações descritas acima, ferramentas de recuperação de informações enfrentam muitas dificuldades, tendo em vista o grande volume de dados, as incertezas existentes sobre as coleções de documentos e o grande número de respostas que podem ser geradas. Estes problemas são gerados principalmente pelas próprias características da linguagem, ou seja, situações como a de termos sinônimos e todas as demais exceções sintáticas que podem ser encontradas em um texto. Além disso, a solução destes pontos nem sempre é trivial, uma vez que, normalmente, o sistema de busca possui poucas informações a respeito do que está sendo procurado, limitando-se exclusivamente às palavras inseridas pelo usuário. Por exemplo, ao se inserir o termo “aeronave” durante o processo de busca, não se considera o fato de um avião ser uma aeronave, ainda que seja óbvio para o usuário.

Com base nesta realidade, a proposta do presente trabalho é desenvolver uma ferramenta para recuperação de informações que utilize uma base de conhecimentos com as relações semânticas entre os termos presentes nos documentos para aprimorar os modelos clássicos de busca.

2 Objetivos

Os objetivos do trabalho foram:

- Criar um modelo de recuperação de informações que consiga utilizar uma base de conhecimentos para aprimorar o seu processo de busca de dados.
- Desenvolver um *software* que implemente o modelo criado.
- Avaliar o desempenho do programa desenvolvido.

3 Referencial teórico

3.1 Modelos de recuperação de informações

A seguir serão descritos sucintamente o funcionamento de alguns dos modelos clássicos de recuperação de informações:

3.1.1 Booleano

É um modelo que utiliza operadores lógicos de forma que os documentos recuperados são aqueles que possuem termos que satisfazem a expressão lógica de busca. Os operadores empregados são:

- **AND**: O termo que sucede este operador deve obrigatoriamente constar no texto procurado.
- **OR**: Pelo menos um dos termos que sucede ou antecede o operador OR deve constar no documento buscado.
- **NOT**: Os termos que sucedem este operador não podem existir no texto procurado.

Para efetuar pesquisas segundo o modelo booleano, são necessárias duas etapas:

1. **Representação da coleção de documentos**: Para cada um dos termos existentes na base de dados é criada uma lista, na qual cada posição está associada a um dos documentos da coleção de textos. Em cada uma destas posições deve existir um valor correspondente a 0 ou 1, indicando respectivamente a presença ou a ausência do termo no documento. Por exemplo: “001010011” representa um termo que está presente em quatro documentos distintos em uma coleção onde existem apenas nove textos.

   A figura abaixo é um exemplo do processo descrito no qual é gerada uma matriz em que as colunas representam os documentos e as linhas, os termos:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	…
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
…							

Figura 1: Matriz de termos e documentos [1]

2. **Análise de uma pesquisa:** Para se avaliar a pesquisa inserida pelo usuário, é preciso substituir os termos dela pelas suas respectivas listas, geradas no passo anterior. Em seguida, devem ser efetuadas as operações booleanas existentes, ou seja, quando o operador AND é utilizado, executa-se a operação booleana de multiplicação; quando for usado o OR, efetua-se a soma e, no caso do NOT, realiza-se a negação. A seguir, alguns exemplos das operações descritas:

    - “01011” AND “10001” = “00001”
    - “01011” OR “10001” = “11011”
    - NOT “01011” = “10100”

Finalmente, considerando que cada posição das listas corresponde a um documento da coleção, basta identificar, no resultado obtido no segundo passo, quais são as posições dos valores 1 para identificar os textos relevantes para a pesquisa.

Exemplo:

Para uma coleção composta por 6 documentos, $\boldsymbol{d_1}$, $\boldsymbol{d_2}$, ..., $\boldsymbol{d_6}$, é inserida a seguinte pesquisa:

"(avião OR helicóptero) AND (NOT supersônico)".

Os termos "avião", "helicóptero" e "supersônico" são representados pelas listas:

$$\text{avião} = 010001; \qquad \text{helicóptero} = 001000; \qquad \text{supersônico} = 110100;$$

Assim temos que:

$$\text{avião OR helicóptero} = \mathbf{010001} + \mathbf{001000} = \mathbf{011001}$$

$$\text{NOT supersônico} = \overline{\mathbf{110100}} = \mathbf{001011}$$

$\therefore (\boldsymbol{avião\ OR\ helicóptero})\boldsymbol{AND}\ (\boldsymbol{NOT\ supersônico}) = \mathbf{011001} \cdot \mathbf{001011} = \mathbf{001001}$

Logo, os documentos relevantes para esta pesquisa são: $\boldsymbol{d_3}$ **e** $\boldsymbol{d_6}$.

3.1.2 Espaço de Vetores

Neste modelo tantos os documentos quanto às pesquisas são representados como vetores cuja orientação é definida de acordo com os termos existentes em cada um deles. Para determinar o quão relevante um dado texto é para uma busca, é construído um *ranking* dos documentos com base no resultado de uma função de *score.* Tal função é nada mais que uma equação cujo valor corresponde a um índice de semelhança entre o vetor da pesquisa e o vetor do texto analisado.

A dimensão do espaço dos vetores descritos acima é igual ao número de termos existentes na coleção de textos. Assim, o vetor de um documento genérico *d,* representado por $\vec{V}(d)$, será formado por um componente relativo a cada uma das dimensões do espaço, ou seja, para cada um dos termos presentes na coleção de documentos. Existem diversas formas de se calcular tais componentes, sendo que a mais utilizada é *tf-idf*. Já o vetor de uma pesquisa *q,* representado por $\vec{V}(q)$, pode ser obtido de forma análoga.

O *tf-idf* é dado pelo produto da frequência do termo no texto (*tf* ou *term frequency*) e do inverso da frequência do documento (*idf* ou *inverse document frequency*). O primeiro determina a quantidade de vezes que a palavra ou expressão aparecem no texto, o segundo representa o quão comum o termo é na coleção pesquisada.

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_{t,d}. \quad (1)$$

Sendo que:

$tf_{t,d}$ = Número de vezes que o termo *t* aparece no documento *d.*

$idf_{t,d} = \log \frac{N}{df_t}$, onde N é o número de documentos na coleção e df_t é o número de documentos que contém o termo *t.*

Para o cálculo da similaridade entre dois vetores, a função mais utilizada é a *cosine similarity.* Ela pode ser obtida através da seguinte expressão:

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| \cdot |\vec{V}(d_2)|}. \quad (2)$$

O numerador da função é o produto escalar dos vetores, que pode ser calculado por:

$$\vec{V}(d_1) \cdot \vec{V}(d_2) = \sum_{i=1}^{n} d_{1,i} d_{2,i}. \quad (3)$$

Onde:

$d_{1,i}$ = a componente *i* do vetor d_1.

n = número de termos existentes na coleção de documentos.

Já o denominador é determinado pelo produto das distâncias euclidianas dos vetores, a qual pode ser calculada segundo a expressão:

$$\sqrt{\sum_{i=1}^{n} \vec{V}_i^{\,2}(d)}\,. \tag{4}$$

Assim, o *score* de um documento para uma pesquisa pode ser determinado através da função de similaridade descrita em que os vetores utilizados são: o da pesquisa do usuário e o do documento em questão:

$$score(d,q) = \frac{\vec{V}(d)\cdot\vec{V}(q)}{|\vec{V}(d)|\cdot|\vec{V}(q)|}. \tag{5}$$

3.1.3 Modelo de Independência Binária (BIM)

Com o propósito de abordar os sistemas de recuperação de informações probabilísticos, importa explicar alguns dos principais modelos. Para tanto, partimos das seguintes fórmulas básicas de probabilidade:

Bayes' Law: $$P(a|b) = \frac{P(b|a)P(a)}{P(b)}. \tag{6}$$

Product Rule: $$P(a,b|c) = P(a|c)P(b|a,c). \tag{7}$$

O modelo de independência binária (BIM) busca estimar de forma prática a relevância de um documento para uma pesquisa.

Para tal, são feitas diversas hipóteses que, em geral, não estão corretas. Contudo, este modelo normalmente retorna resultados satisfatórios. As hipóteses adotadas são:

- Os termos de um documento são independentes entre si.

- A relevância de um documento é independente da relevância de outro documento.

Considere $R_{d,q}$ como a variável aleatória que possui o valor de 1 quando certo documento *d* é relevante e 0 caso contrário, dado uma pesquisa *q*. Para abreviar a notação será usado apenas o símbolo R. Então podemos definir:

$$d \text{ é relevante} \Leftrightarrow P(R = 1|d, q) > P(R = 0|d, q). \quad (8)$$

Pode-se representar *d* como um vetor $\vec{x} = (x_1, x_2, \dots, x_m)$ onde $x_t = 1$ se o termo *t* estiver presente em *d* e $x_t = 0$ caso contrário. Também é possível estender a notação vetorial de forma análoga ao que foi feito com os documentos para a pesquisa *q*. Desta forma, $\vec{q} = (q_1, q_2, \dots, q_m)$ sendo q_i associado de forma binária a um determinado termo. Assim segue que:

$$P(R|\vec{x}, \vec{q}) = \frac{P(\vec{x}|R, \vec{q})P(R|\vec{q})}{P(\vec{x}|\vec{q})}. \quad (9)$$

Além disso, considerando que R só pode assumir os valores de 0 e 1,

$$P(R = 1|\vec{x}, \vec{q}) + P(R = 0|\vec{x}, \vec{q}) = 1. \quad (10)$$

3.1.3.1 Construção do ranking

Dada uma pesquisa *q,* deseja-se ordenar os resultados por critério de relevância, ou seja, de acordo com $P(R = 1|\vec{x}, \vec{q})$. Como queremos apenas ranquear os documentos, para simplificar, podemos estabelecer uma razão entre a probabilidade de ser relevante pela de não ser, assim:

$$O(R|\vec{x}, \vec{q}) = \frac{P(R=1|\vec{x}, \vec{q})}{P(R=0|\vec{x}, \vec{q})} = \frac{\frac{P(R=1|\,\vec{q})P(\vec{x}|R=1, \vec{q})}{P(\vec{x}, \vec{q})}}{\frac{P(R=0|\,\vec{q})(\vec{x}|R=0, \vec{q})}{P(\vec{x}, \vec{q})}} = \frac{P(R=1|\,\vec{q})}{P(R=0|\,\vec{q})} \frac{P(\vec{x}|R=1, \vec{q})}{P(\vec{x}|R=0, \vec{q})}. \quad (11)$$

Nesta relação ainda pode-se descartar o termo $\frac{P(R=1|\,\vec{q})}{P(R=0|\,\vec{q})}$, pois o mesmo será uma constante para cada pesquisa. Para determinar o vetor $\vec{x}$ vamos utilizar a independência dos temos. Desta forma segue que:

$$\frac{P(\vec{x}|R=1,\vec{q})}{P(\vec{x}|R=0,\vec{q})} = \prod_{t=1}^{m} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}, \tag{12}$$

$$\Rightarrow O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \prod_{t=1}^{m} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}. \tag{13}$$

Como os valores de x_t são binários, podemos reescrever a equação

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \prod_{t:x_t=1} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})} \prod_{t:x_t=0} \frac{P(x_t|R=1,\vec{q})}{P(x_t|R=0,\vec{q})}. \tag{14}$$

Neste ponto é assumida mais uma simplificação: que os termos não presentes na pesquisa são igualmente prováveis de aparecerem nos documentos relevantes e nos irrelevantes.
Matematicamente:

$$q_t = 0 \Rightarrow P(x_t = 1|R = 1,\vec{q}) = P(x_t = 1|R = 0,\vec{q}). \tag{15}$$

Para simplificar a notação adotamos:

$$p_t = P(x_t = 1|R = 1,\vec{q}). \tag{16}$$

$$u_t = P(x_t = 1|R = 0,\vec{q}). \tag{17}$$

Assim segue que:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \prod_{t:x_t=q_t=1} \frac{p_t}{u_t} \prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t}. \quad (18)$$

Como:

$$\prod_{t:x_t=0,q_t=1} \frac{1-p_t}{1-u_t} = \prod_{t:q_t=1} \frac{1-p_t}{1-u_t} \prod_{t:x_t=1,q_t=1} \frac{1-u_t}{1-p_t}. \quad (19)$$

Substituindo na expressão de $O(R|\vec{x},\vec{q})$ temos:

$$O(R|\vec{x},\vec{q}) = O(R|\vec{q}) \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} \prod_{t:q_t=1} \frac{1-p_t}{1-u_t}. \quad (20)$$

O produto $\prod_{t:q_t=1} \frac{1-p_t}{1-u_t}$ é composto por todos os termos da pesquisa, logo seu valor será uma constante para essa pesquisa assim como o valor de $O(R|\vec{q})$. Portanto, para se criar um ranking dos documentos, o único termo que precisa ser avaliado é o produto $\prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)}$. Uma forma de quantificar esse termo sem que haja perda de informações é através de seu logaritmo, o resultado é denominado "*Retrieval Status Value*" ou simplesmente RSV.

$$RSV_d = \log \prod_{t:x_t=q_t=1} \frac{p_t(1-u_t)}{u_t(1-p_t)} = \sum_{t:x_t=q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}. \quad (21)$$

Definimos:

$$c_t = \log \frac{p_t(1-u_t)}{u_t(1-p_t)} = \log \frac{p_t}{(1-p_t)} + \log \frac{(1-u_t)}{u_t}. \quad (22)$$

Assim o valor de c_t será 0 se a probabilidade de um termo aparecer em um documento relevante for a mesma dele aparecer em um documento não relevante, caso contrário este termo deve assumir um valor positivo. Desta forma é possível montar o ranking através do RSV:

$$\boxed{RSV_d = \sum_{x_t=q_t=1} c_t}. \tag{23}$$

O resultado de uma consulta à base de dados pode ser representado na seguinte tabela:

	Documentos	Relevantes	Não Relevantes	Total
Termo presente	$x_t = 1$	S	$df_t - s$	df_t
Termo ausente	$x_t = 0$	S-s	$(N - df) - (S - s)$	$N - df_t$
Total		S	N-S	N

Tabela 1: Resultados de uma consulta na base de dados

df_t = Número de documentos com o termo procurado.

N = Número de documentos.

S = Número de documentos relevantes.

s = Número de documentos relevantes com o termo procurado.

Logo:

$$p_t = \frac{s}{S} \qquad u_t = \frac{df_t - s}{N - S}. \tag{23}$$

Então,

$$c_t = \log\left(\frac{s/(S-s)}{(df_t-s)/\left((N-df_t)-(S-s)\right)}\right). \quad (24)$$

A técnica utilizada para estimar as probabilidades na construção da expressão acima é baseada na contagem do número de eventos observados sobre o número de eventos existentes. Ela também é conhecida como "*relative frequency*" (freqüência relativa). Assim estimar tais probabilidades através da freqüência relativa dos termos observados pode ser denominado como "*maximum likelihood estimate*" (MLE) uma vez que maximiza os resultados de acordo com os dados analisados.

Um dos problemas de se empregar diretamente o MLE é que os eventos vistos normalmente assumem valores muito altos enquanto que os demais obtêm valores próximos de 0. Para contornar essa dificuldade deve-se "suavizar" a expressão, ou seja, diminuir os valores para eventos com maior chance de aparecer e aumentar os daqueles com uma menor chance de serem vistos. Uma maneira simples de se efetuar tal operação é adicionando uma constante aos termos da fórmula, sendo que o seu valor quantifica a crença do modelo possuir uma distribuição uniforme. Assim, dado que é usual adotar um valor de $\frac{1}{2}$ para a constante [1], segue:

$$\widehat{c_t} = \log\left(\frac{\left(s+\frac{1}{2}\right)/(S-s+\frac{1}{2})}{\left(df_t-s+\frac{1}{2}\right)/\left(N-df_t-S+s+\frac{1}{2}\right)}\right). \quad (25)$$

3.1.3.2 Estimar u_t e p_t

Considerando que a quantidade de documentos relevantes é muito menor que a de documentos existentes em nosso banco, é razoável aproximar o número de documentos não relevantes pelo de documentos existentes em nossa coleção. Ou seja:

$$S \ll N \Rightarrow N - S \approx N. \tag{26}$$

Assim,

$$\boxed{u_t = \frac{df_t}{N}}. \tag{27}$$

$$\log \frac{(1-u_t)}{u_t} = \log \frac{(N-df_t)}{df_t} \approx \log \frac{N}{df_t} = idf_t. \tag{28}$$

Desta relação obtém o df_t que é denominado "*inverse document frequency*" (freqüência invertida do documento) que corresponde justamente a um valor normalmente utilizado em sistemas de recuperação de informações para atribuir pesos a documentos, visando à construção de uma função de "*score*" dos mesmos.

Contudo a técnica utilizada para estimar u_t não pode ser estendida para p_t.

Assim, outra maneira de realizar as estimativas é através de um método iterativo com feedback (*relevance feedback* ou RF) que funciona da seguinte forma:

1. Escolher valores iniciais para u_t e p_t.
2. Utilizando os valores de p_t e u_t estima-se qual é o conjunto de documentos relevantes e os exibe para o usuário. $R = \{d: R_{d,q} = 1\}$.
3. Interagir com o usuário para montar um subconjunto de documentos V que será dividido em:
   a. $VR = \{d \in V, R_{d,q} = 1\} \subset R$
   b. $VNR = \{d \in V, R_{d,q} = 0\}$

   Sendo que VR_t e VNR_t correspondem aos elementos destes subgrupos que possuem o termo x_t.
4. Estimar novamente os valores de u_t e p_t com:

$$p_t = \frac{|VR_t|+\frac{1}{2}}{|VR|+1}. \quad (29)$$

Se V for muito pequeno usar:

$$p_t{}^{(k+1)} = \frac{|VR_t|+k.p_t{}^{(k)}}{|VR|+k}. \quad (30)$$

5. Retornar ao passo 2 até que o usuário esteja satisfeito

Este algoritmo pode ser simplificado assumindo que VR = V:

1. Escolher valores iniciais para u_t e p_t.
2. Achar o conjunto V de documentos.
3. Determinar u_t e p_t:

$$p_t = \frac{|V_t|+\frac{1}{2}}{|V|+1}, \quad (31)$$

$$u_t = \frac{df_t-|V_t|+\frac{1}{2}}{N-|V|+1}. \quad (32)$$

4. Voltar ao passo 2 até que o ranking dos resultados retornados convirja

Substituindo na equação de c_t:

$$c_t = \log\left[\frac{p_t(1-u_t)}{u_t(1-p_t)}\right] \approx \log\left[\frac{|V_t|+\frac{1}{2}}{|V|-|V_t|+1}.\frac{N}{df_t}\right] = \log\left[\frac{|V_t|+\frac{1}{2}}{|V|-|V_t|+1}\right] + \log\left[\frac{N}{df_t}\right]. \quad (33)$$

3.1.4 Okapi BM25

É um modelo não binário que se propõe a realizar pesquisas em coleções de documentos maiores para as quais o BIM não consegue lidar de forma

satisfatória. Para tal é analisada tanto a freqüência dos termos como o tamanho dos documentos.

Como visto no item acima, uma das formas mais simples de se montar um ranking é através do cálculo do idf_t dos termos da pesquisa:

$$RSV_d = \sum_{t \in q} idf_t = \sum_{t \in q} log\left(\frac{N}{df_t}\right). \quad (34)$$

Na ausência de um feedback da relevância dos documentos, costuma-se aproximar $S = s = 0$ (em geral S e $s \ll N$). Assim a fórmula do RSV_d deve ser alterada para:

$$RSV_d = \sum_{t \in q} log\left(\frac{N - df_t + \frac{1}{2}}{df_t + \frac{1}{2}}\right). \quad (35)$$

Um problema encontrado nesta fórmula ocorre se df_t for grande, ou seja, se um dado termo ocorrer em mais da metade dos documentos. Neste caso o valor obtido será negativo, o que não é desejável. Uma alternativa para solucionar essa questão é adotar um mínimo para o resultado, por exemplo, zero.

A expressão descrita pode ser melhorada pela inclusão de fatores relativos à freqüência dos termos e o tamanho dos documentos:

$$RSV_d = \sum_{t \in q} log\left(\frac{N}{df_t}\right) . \frac{(k_1+1)tf_{dt}}{k1\left((1-b)+b.\left(\frac{L_d}{L_{ave}}\right)\right)+tf_{dt}}. \quad (36)$$

tf_{dt} = Freqüência do termo t no documento d.

L_d = Tamanho do documento d.

L_{ave} = Média do tamanho dos documentos da coleção.

k_1 = Variável positiva que ajusta quanto será considerado o fator de freqüência do termo. $k_1 = 0$ corresponde a um modelo binário enquanto que altos valores

corresponderiam aqueles que consideram praticamente só a freqüência do termo para gerar o ranking.
$b =$ Valor entre 0 e 1 que proporcional ao quanto o tamanho dos documentos influenciará o ranking.

Para uma pesquisa com muitos termos a serem procurados pode-se deduzir que alguns destes valores provavelmente vão ter uma maior significância para um resultado ser relevante ou não. Assim é possível modificar novamente o cálculo de RSV_d para considerar esta nova possibilidade pela simples análise da freqüência dos termos na pesquisa:

$$RSV_d = \sum_{t \in q} log\left(\frac{N}{df_t}\right) . \frac{(k_1+1)tf_{dt}}{k_1\left((1-b)+b.\left(\frac{L_d}{L_{ave}}\right)\right)+tf_{dt}} . \frac{(k_3+1)tf_{dq}}{k_3+tf_{dq}}. \quad (37)$$

$tf_{dq} =$ Freqüência do termo t na pesquisa q.

$k_3 =$ Variável positiva proporcional ao quanto será considerado o fator de freqüência do termo na pesquisa.

Segundo [1] os valores que, empiricamente, resultam em um bom desempenho das fórmulas acima são k_1 e k_3 entre 1.2 e 2 e $b = 0.75$.

No caso de haver um sistema de feedback é possível usar a fórmula completa de $\widehat{c_t}$ no lugar da aproximação de idf_t discutida no modelo binário. Segue que:

$$RSV_d = \sum_{t \in q} \left[\log \left[\frac{\frac{|VR_t|+\frac{1}{2}}{|VNR_t|+\frac{1}{2}}}{\frac{df_t-|VR_t|+\frac{1}{2}}{N-df_t-|VR|+|VR_t|+\frac{1}{2}}} \right] . \frac{(k_1+1)tf_{dt}}{k1\left((1-b)+b.\left(\frac{L_d}{L_{ave}}\right)\right)+tf_{dt}} . \frac{(k_3+1)tf_{dq}}{k_3+tf_{dq}} \right]. \quad (38)$$

3.1.5 Modelo de dependências limitadas

O modelo de dependências limitadas tenta melhorar a qualidade das respostas ao incluir algumas relações entre os termos existentes nos documentos. Esta é uma possível solução para pesquisas em que há muitos documentos classificados como relevantes.

Dado um documento d pertencente a uma coleção D, R é o conjunto de documentos relevantes de D. Temos então uma possível forma de determinar a função de Score(d) é:

$$Score(d) = \frac{P(R|d)}{P(\neg R|d)} = \frac{\frac{p(d|R)P(R)}{P(d)}}{\frac{P(d|\neg R)P(\neg R)}{P(d)}} \propto \frac{P(d|R)}{P(d|\neg R)}. \quad (39)$$

Considerando uma pesquisa Q, X é o conjunto de termos em Q e Y corresponde ao conjunto dos demais termos existentes na coleção mas não na pesquisa. Assim d(X) é o subconjunto de valores relativos aos atributos de X contidos em d e, analogamente, d(Y) são os termos de Y presentes no documento. Para simplificar a notação vamos representar d(X) por X e d(Y) por Y.

Assumindo que $\neg R \gg R$ é válido aproximar ¬R ≈ D.

$$Score(d) \propto \frac{P(d|R)}{P(d|D)} = \frac{P(X,Y|R)}{P(X,Y|D)} = \frac{P(Y|R)}{P(Y|D)} . \frac{P(X|Y,R)}{P(X|Y,D)}. \quad (40)$$

Como $R \subseteq X \Rightarrow p(X|Y,R) = 1$ obtemos:

$$Score(d) \propto \frac{P(X,Y|R)}{P(X,Y|D)} = \frac{P(Y|R)}{P(Y|D)} . \frac{1}{P(X|Y,D)}. \quad (41)$$

A dependência limitada é dada assumindo que os termos de X e Y são independentes entre si próprios, mas a dependência entre X e Y é permitida. Logo:

$$Score(d) \propto \prod_{y \in Y} \frac{P(y|R)}{P(y|D)} \prod_{x \in X} \prod_{y \in Y} \frac{1}{P(x|y,D)}. \quad (42)$$

Uma forma de estimar $p(y|R)$ é analisando os resultados das pesquisas realizadas no passado. Tal informação estaria disponível em um grupo W de conjunto de informações representadas por vetores para cada uma das pesquisas realizadas.

Portanto, aproximando R pelos vetores de pesquisa em W que contém X, temos:

$$P(y|R) = P(y|X,W). \quad (43)$$

Assim,

$$Score(d) \propto \frac{P(Y|X,W)}{P(Y|D)} \frac{1}{P(X|Y,D)}. \quad (44)$$

Como:

$$p(Y|X,W) = \frac{p(X,W,Y)}{p(X,W)} = \frac{p(W)p(Y|W)p(X|Y,W)}{p(X,W)}. \quad (45)$$

Desconsiderando a constante $\frac{p(W)}{p(X,W)}$:

$$Score(t) \propto \frac{p(Y|W)}{p(Y|D)} \frac{p(X|Y,W)}{p(X|Y,D)} = \prod_{y \in Y} \frac{p(y|W)}{p(y|D)} \prod_{x \in X} \prod_{y \in Y} \frac{p(x|y,W)}{p(x|y,D)}. \quad (46)$$

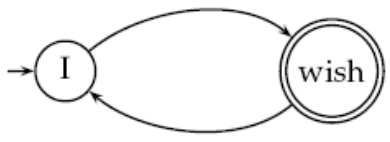
Assim resta calcular as probabilidades “atômicas”:

$$p(y|W), p(y|D), p(x|y,W) \text{ e } p(x|y,D). \quad (47)$$

A forma mais usual de se quantificar tais probabilidades é através do cálculo da idf em W ou D para cada uma das relações descritas.

3.1.6 Modelo de linguagem para RI

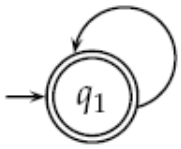
Os modelos probabilísticos de linguagem podem ser definidos como funções que denotam uma medida de probabilidade para cada termo de um vocabulário. Tradicionalmente eles são usados para reconhecer e/ou gerar *strings*, sendo que o conjunto dos *strings* possíveis é denominado a linguagem de um autômato. A seguir temos um exemplo (dado em [1]) de um autômato finito simples (Figura 1) e de um autômato probabilístico (Figura 2) com uma única variável de estado e uma distribuição de probabilidade que permite gerar diferentes termos:



I wish
I wish I wish
I wish I wish I wish
I wish I wish I wish I wish I wish I wish
...

CANNOT GENERATE: wish I wish

Figura 2: Autômato finito simples e alguns strings gerados. A seta indica o estado de início e o círculo duplo um possível estado para o término.Extraído [1].



$P(\text{STOP}|q_1) = 0.2$

the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
...	...

Figura 3: Um autômato finito de um estado que se comporta como um modelo de linguagem.Extraído [1].

Assim ao invés de estimar o quão relevante um documento é para uma pesquisa avaliando a incidência dos termos buscados para gerar um ranking, a idéia básica desta abordagem é construir um modelo de linguagem probabilística M_d para representar cada um destes documentos de forma que seja possível estimar a probabilidade de M_d gerar a pesquisa.

Os principais métodos utilizados para calcular as probabilidades de formação dos strings são descritos a seguir:

Sabendo que:

$$P(t_1 t_2 t_3 \dots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_1, t_2) \dots P(t_n|t_1, \dots, t_{n-1}). \quad (48)$$

É possível verificar que para a estimativa de $P_{bi}(t_1 t_2 t_3 \dots t_n)$ é necessário determinar os valores de todas as probabilidades condicionais entre os seus termos *t*.

Contudo, muitas vezes calcular os fatores descritos acima pode ser uma tarefa muito complicada, por isso costuma-se adotar simplificações, sendo que as mais comuns são o modelo *unigran* e o *bigran*. No primeiro é assumida a independência dos termos, enquanto que no segundo considera-se que os termos dependem apenas do seu antecessor.

A seguir a representação das simplificações descritas para o cálculo de $P(t_1 t_2 t_3 \dots t_n)$.

Modelo *unigran*: $P_{uni}(t_1 t_2 t_3 \dots t_n) = P(t_1)P(t_2)P(t_3) \dots P(t_n).$ (49)

Modelo *bigran*: $P_{bi}(t_1 t_2 t_3 \dots t_n) = P(t_1)P(t_2|t_1)P(t_3|t_2) \dots P(t_n|t_{n-1}).$ (50)

Considerando o modelo de linguagem unigran, tem-se que para o resultado final a ordem dos termos é irrelevante.

Assim, inicialmente temos que:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \quad (51)$$

Como $P(q)$ é igual para todos os documentos e assumindo que $P(d)$ também seja, vamos desconsiderar estas probabilidades. Portanto o único fator a ser avaliado é $P(q|d)$.

Uma forma de montar um ranking seguindo esta relação é com o MULM (*Multinomial Unigran Language Model*), onde esta probabilidade será estimada por:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{tf_{d,t}} \quad (52)$$

$K_q = {L_d!}\Big/{\left(tf_{d1,t}!\, tf_{d2,t}! \dots tf_{dM,t}!\right)}$.

$L_d = \sum_{1 \le i \le M} tf_{di,t}$.

$M =$ Tamanho do vocabulário de termos.

V = Termos que compõem o vocabulário da coleção.

Um sistema de recuperação de informações baseado nesse conceito precisa:

- Gerar um modelo de linguagem para cada documento.
- Estimar $P(q|M_d)$ para cada um dos documentos.
- Montar um ranking de acordo com as probabilidades calculadas.

Para obter bons resultados o usuário deve conhecer a estrutura da coleção de forma a realizar pesquisas somente com as palavras que aparecem nos documentos.

Ainda considerando o unigran a probabilidade $P(q|M_d)$ pode ser estimada segundo o MLE (*Maximun Likelihood Estimation*). Assim:

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{mle}(t|M_d) = \prod_{t \in q} \frac{tf_{d,t}}{L_d} \tag{53}$$

O principal problema desta estimativa é que se um termo inserido na pesquisa não existir no documento então, como os termos da fórmula descrita acima são multiplicados uns pelos outros, temos que $\hat{P}(t|M_d)$ será avaliada como nula. Para contornar essa dificuldade são sugeridas algumas formas de suavização:

1. Misturar os modelos gerados para os documentos e para a coleção como um todo:

$$\hat{P}(t|d) = \lambda \hat{P}(t|M_d) + (1 - \lambda)\hat{P}(t|M_c) \tag{54}$$

   Sendo M_c o modelo de linguagem gerado para a coleção de documentos e o λ um constante com valor entre 0 e 1, que é inversamente proporcional à suavização.

2. Utilizar um modelo de linguagem construído para toda coleção como uma distribuição a priori num processo de atualização bayesiano:

$$\hat{P}(t|d) = \frac{tf_{d,t} + \alpha.\hat{P}(t|M_c)}{L_d + \alpha} \quad (55)$$

Onde α é uma constante cujo valor é proporcional à suavização.

De acordo com a primeira técnica sugerida, pode-se montar um ranking dos documentos fazendo uma comparação das probabilidades $P(d|q)$ segundo a relação:

$$P(d|q) \propto \prod_{t \in q} \left(\lambda P(t|M_d) + (1-\lambda)P(t|M_c)\right) \quad (56)$$

3.1.7 Resumo

Retomando o que foi apresentado, segue-se uma síntese dos modelos.

- **Booleano**:
  - Não monta um ranking dos resultados, apenas determina se um dado documento é relevante ou não.
  - Não considera a freqüência dos termos procurados (apenas se eles existem ou não).
  - Modelo programável, ou seja, utiliza operadores lógicos na sentença de busca.
  - Muito eficiente se o usuário souber exatamente o que procura.

- **Espaço de Vetores**:
  - Modelo matemático simples para o calculo de similaridades.
  - Considera a freqüência dos termos procurados.

- Não considera a ordem que as palavras estão dispostas em um texto.
- Geralmente se comporta bem em coleções genéricas.

- **Modelo binário**:
  - Assume independência entre termos e documentos.
  - Não considera a freqüência dos termos nos documentos nem o tamanho destes.
  - Implementação simples.
  - Apresenta bons resultados de ranking para coleções pequenas.

- **Okapi BM25**:
  - Assume independência entre termos e documentos.
  - Considera a freqüência dos termos e o tamanho dos documentos.
  - Se forem utilizados muitas entradas para pesquisa, o modelo também permite considerar a freqüência dos termos nesta pesquisa.
  - Propõe-se a trabalhar com grandes coleções.

- **Modelo de dependências limitadas**:
  - Independência entre os documentos.
  - Assume que os termos pesquisados são dependentes dos não pesquisados, mas são independentes entre si. E os termos não pesquisados são independentes entre si, mas dependentes dos pesquisados.
  - Visa melhorar o ranking em sistemas onde são geradas muitas respostas para uma pesquisa.

- **Modelo de linguagem**:
  - Independência entre os documentos.
  - Avalia a estrutura dos documentos através da construção de um modelo de linguagem.
  - Normalmente apresenta resultados muito bons quando adotadas formas eficientes de suavização.

- Em geral o seu desempenho é superior aos demais modelos discutidos.

3.2 Avaliação de desempenho em sistemas de recuperação de informações

Os conceitos associados às técnicas de avaliação de desempenho de sistemas de recuperação de informação são:

- **Precisão**: É definida pela razão do número de documentos relevantes retornados para uma pesquisa sobre o número total de documentos retornados, ou seja, o que de fato é relevante na resposta do sistema. Este fator pode ser calculado pela seguinte fórmula:

$$P = \frac{n_{q,r}}{n_q} \tag{57}$$

$P =$ Precisão.

$n_{q,r} =$ Número de documentos relevantes na resposta do sistema.

$n_q =$ Número total de documentos na resposta do sistema.

- **Retorno**: É dado pela razão entre o número de documentos relevantes encontrados para uma pesquisa e o número total de documentos relevantes existentes na coleção. Para se determinar tal fator é necessário um conhecimento prévio da base de dados. Este fator pode ser calculado pela seguinte fórmula:

$$R = \frac{n_{q,r}}{n_{c,r}} \tag{58}$$

$R =$ Retorno.

$n_{q,r}$ = Número de documentos relevantes na resposta do sistema.

$n_{c,r}$ = Número de documentos relevantes existentes na coleção.

- **Tempo de resposta**: Corresponde ao intervalo de tempo entre a entrada da pesquisa e o retorno do sistema.

A precisão e o retorno são fatores que avaliam o sistema de forma quantitativa enquanto o tempo de resposta avalia a sua velocidade.

Assim, uma forma de analisar o desempenho de um sistema em relação à qualidade de suas respostas é calculando a curva de *precisão x retorno*. Tal gráfico permite visualizar o quão boa é a precisão para cada nível de retorno obtido.

A figura a seguir consiste em um exemplo, extraído de [1], de uma curva de *precisão x retorno*:

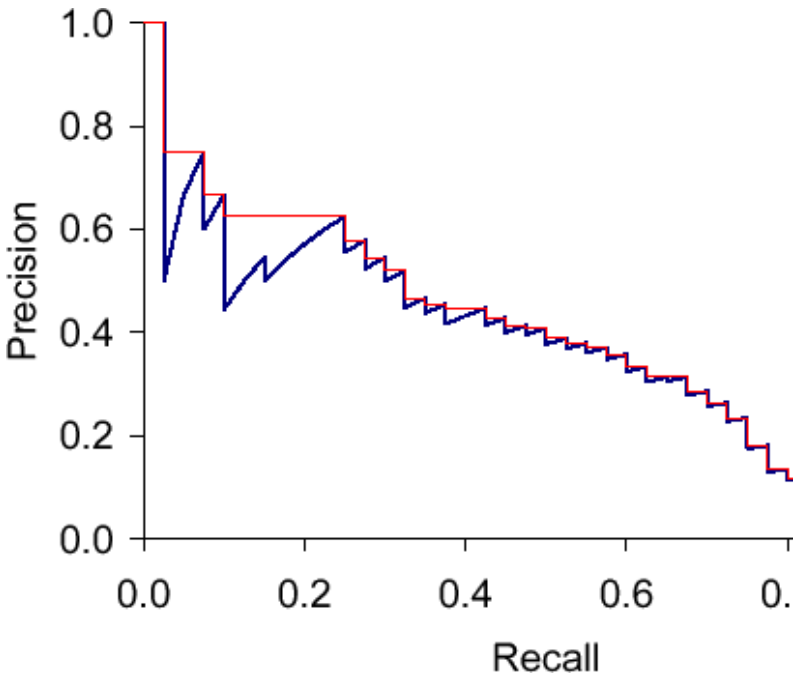


Figura 4: Precisão (precision) x Retorno (recall)

Frequentemente a curva obtida apresenta variações locais (pequenos picos). Isto ocorre porque, toda vez que um documento relevante sucede um não

relevante no ranking gerado para uma pesquisa, tem-se um aumento da precisão e do retorno. Uma técnica utilizada para remover estas variações e melhorar a visualização da curva consiste em efetuar uma interpolação denominada *11-point interpolated average precision.* Tal método consiste em calcular a média aritmética do fator de precisão para 11 níveis de retorno previamente definidos (0.0, 0.1, 0.2, ... , 0.9, 1.0). Para este procedimento é empregada a seguinte função no cálculo da precisão em um nível de retorno:

$$p_{interp}\,(r) = \max_{r' \geq r} p(r'). \qquad (59)$$

Onde:

r = nível de retorno.

r' = nível de retorno acima de r.

p(r) = precisão para o nível r.

A figura abaixo (extraída de [1]) ilustra o resultado obtido através da aplicação da *11-point interpolated average precision:*

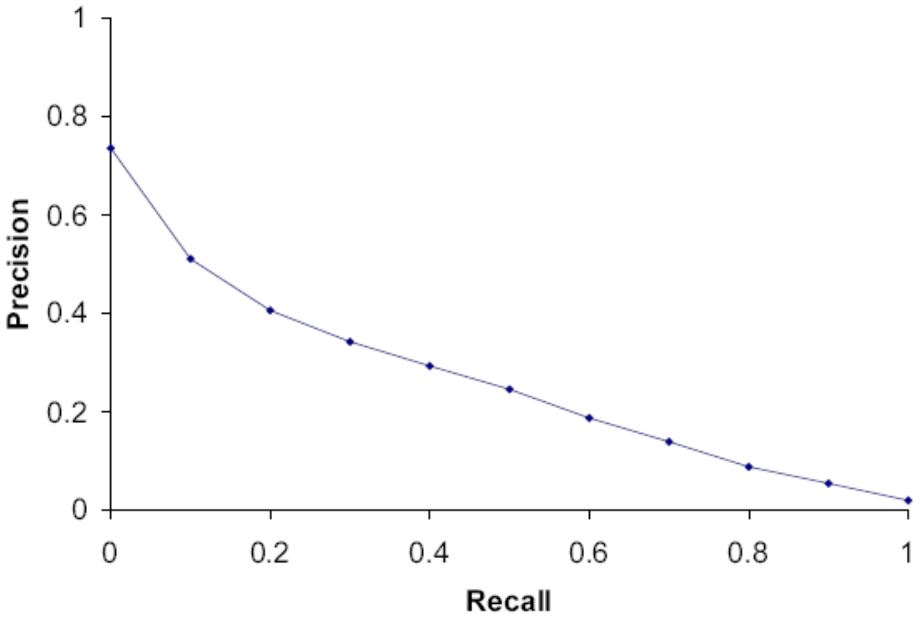


Figura 5: Precisão x Retorno - 11-point interpolated average precision

Também é possível estimar a performance do sistema através do cálculo de um fator único de medida denominado *F-measure* [5], que consiste na média harmônica dos valores obtidos para a precisão e para o retorno. Assim, maximizar essa medida é o mesmo que maximizar a combinação entre a precisão e o retorno.

$$F\text{-}measure = 2\frac{R \times P}{R + P}. \tag{60}$$

3.3 Representação de conhecimentos

O conhecimento pode ser representado por um conjunto de informações armazenadas, nas quais pode-se interpretar, identificar, predizer e responder questões [16]. Por isso, modelos de representações são importantes não só para se recuperar dados, mas também para raciocinar através deles de forma que seja possível efetuar deduções com base nas informações obtidas.

Uma ontologia é um modelo de representação de conceitos e os seus respectivos relacionamentos definidos para um domínio. Por exemplo, para o domínio de meios de transporte, pode ser definida uma classe de “veículos automotores” que possui subclasses tais como “automóveis”, “motocicletas” etc. Além disso, cada membro desta subclasse pode possuir atributos, ou seja, um “automóvel” tem um “motor”, “rodas”, “portas” etc.

Uma forma de construir uma ontologia é através de uma representação lógica dos conceitos. A seguir uma descrição dos principais tipos:

3.3.1 Lógica proposicional

É o tipo mais simples de lógica. Nela cada símbolo representa uma proposição ou um fato. Uma proposição pode ser definida como uma sentença afirmativa que exprime um pensamento ou sentido completo (não necessita de nenhum

complemento para ser entendida). Além disso, neste modelo, cada uma destas proposições pode assumir somente dois valores: verdadeiro ou falso.

Exemplos:

P = “Está chovendo” (pode ser falso ou verdadeiro conforme a situação).

Q = “Cachorros são mamíferos” (sempre verdadeiro).

R = “Elefantes sabem voar” (sempre falso).

A partir de uma proposição simples (que não possui nenhuma outra proposição integrada a ela) é possível criar proposições compostas (formada por duas ou mais proposições) utilizando-se de conectores lógicos. A tabela seguinte representa os diferentes tipos de conectivos:

Símbolo	Descrição
⌐	Negação. Inverte o valor da proposição.
∧	Conjunção (“e”). Todas as proposições devem ser verdadeiras para que a expressão seja verdade.
∨	Disjunção (“ou”). Ao menos uma das proposições deve ser verdadeira para que a expressão também seja verdadeira
→	Implicação (“se ... implica ...”). Se a primeira proposição for verdadeira a segunda também será.
⇔	Bicondicional. A expressão será verdadeira somente quando ambas as proposições possuírem o mesmo valor,

Tabela 2: Conectivos lógicos

A tabela da verdade abaixo exemplifica a aplicação dos conectivos descritos:

A	B	⌐A	A ∧ B	A ∨ B	A → B	A ⇔ B
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Tabela 3: Funções lógicas. Verdadeiro = 1; Falso = 0

3.3.2 Lógica de primeira ordem

Também conhecida como lógica de predicados, a lógica de primeira ordem possui três tipos de elementos sintáticos básicos representados por símbolos: objetos, predicados (relações) e funções.

Na lógica de primeira ordem, além dos conectivos lógicos, existem dois tipos de quantificadores que permitem determinar propriedades de um conjunto/grupo de objetos. Tais quantificadores são:

- **Universal**: Representado pelo símbolo "∀" significa "para todo" e referencia todo um conjunto de objetos. Por exemplo, "$\forall x\ \text{elefante}(x) \rightarrow \text{mamífero}(x)$" significa que todos elefantes são mamíferos.

- **Existencial**: Representado pelo símbolo "∃" significa "existe" pelo menos um dos objetos de um grupo. Por exemplo, "$\exists x\ \text{ave}(x) \longrightarrow \text{voa}(x)$" signifca que pelo menos uma ave voa.

3.3.3 Lógica descritiva

A lógica descritiva é um fragmento de lógica de primeira ordem na qual se pretende facilitar a notação e a descrição das definições e das propriedades das categorias. Para tal, são definidos dois grupos distintos de sentenças:

- **TBox**: Especifica informações sobre os conceitos do domínio. Exemplos:
  "$Mulher \equiv Pessoa \wedge Feminino$"
  "$Homem \equiv Pessoa \wedge \neg Feminino$".

- **ABox**: Especifica as propriedades dos indivíduos do domínio. Exemplo: "$Mulher(Maria)$", "$Homem(João)$", "$temFilho(Maria, João)$"

Exemplo Ontologia

$$P(Animal(x)) = 0.9$$
$$P(Rational(x)) = 0.6$$
$$P(hasChild(x, y)) = 0.3$$
$$Human(x) \equiv Animal(x) \wedge Rational(x)$$
$$Beast(x) \equiv Animal(x) \wedge \neg Rational(x)$$
$$Parent(x) \equiv Human(x) \wedge \exists hasChild(x, y).Human(y)$$
$$Kangaroo(x) \sqsubseteq Beast(x)$$
$$P(Kangaroo|Beast) = 0.4$$
$$MaternityKangaroo(x) \equiv Kangaroo(x) \wedge \exists hasChild(x, y).Kangaroo(y)$$

3.3.4 KRSS [9]

O KRSS (*Description-Logic Knowledge Representation System Specification*) consiste em um conjunto de especificações para sistemas de representação de conhecimentos baseados em lógica descritiva. A tabela a seguir descreve os principais termos da sintaxe utilizada no KRSS:

Sintaxe	Significado
(and $C_1 \ldots C_n$)	$C_1 \sqcap \ldots \sqcap C_n$
(or $C_1 \ldots C_n$)	$C_1 \sqcup \ldots \sqcup C_n$
(not C)	$\neg C$
(all R C)	$\forall R: C$
(some R)	$\exists R$
(some R C)	$\exists R. C$
(define-concept C D)	$C \equiv D$
(define-primitive-concept C D)	$C \sqsubseteq D$
(probabity A α)	$P(A) = \alpha$
(conditional-probability B A α)	$P(A\|B) = \alpha$

Exemplo KRSS:

```
(probability A(x) 0.7)
(probability B(x) 0.4)
(define-concept C(x)
    (and
         A(x)
         (not B(x) )
    (
)
```

3.4 Redes Bayesianas

Uma Rede Bayesiana é um grafo direcional acíclico composto por um conjunto de variáveis e de suas relações de dependências, representados por nós e arcos, respectivamente. Tais relações existentes entre as variáveis são definidas sob o formato de probabilidades condicionais de forma que é possível utilizar a rede para modelar conhecimentos incertos.

Com estas ferramentas é possível calcular a distribuição de um subconjunto de variáveis através da observação de alguns nós do grafo. Tal processo é definido como inferência e permite uma grande flexibilidade e adaptabilidade do modelo, já que os valores obtidos são atualizados no momento em que novas evidências são observadas. A figura abaixo ilustra um exemplo de Rede Bayesiana:

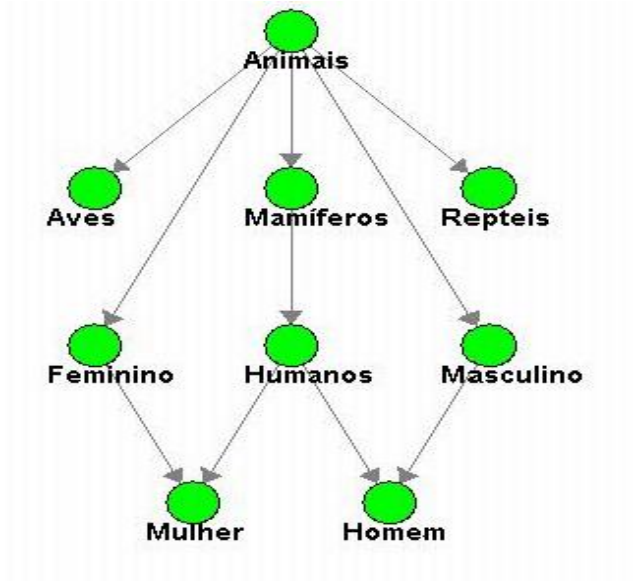


Figura 6: Exemplo de Rede Bayesiana [2]

4 Proposta

A proposta do presente trabalho é desenvolver uma ferramenta de recuperação de informações capaz de utilizar um conjunto de conhecimentos sobre as relações semânticas existentes no seu banco de dados para melhorar o seu processo de pesquisa. Ou seja, um sistema de busca que melhora a qualidade dos resultados através da análise do significado das palavras procuradas.

Assim foi elaborado um sistema composto por três módulos:

1. Aquisição de conhecimentos
2. Pesquisa no banco de dados e criação de um *ranking* com os resultados
3. Exibição dos resultados encontrados agrupados de acordo com a sua relevância e assunto

A seguir, uma representação esquemática dos módulos:

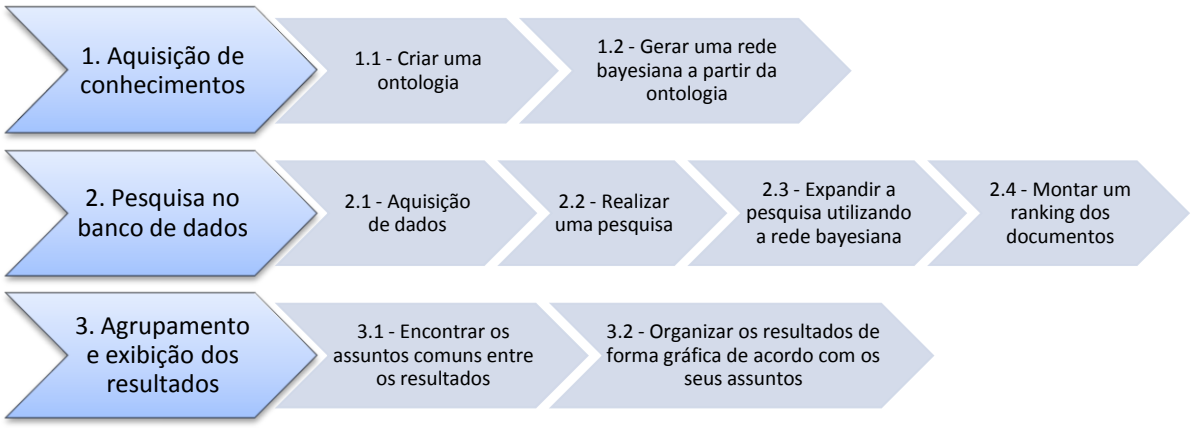


Figura 7: Representação esquemática dos módulos do sistema proposto

4.1 Aquisição de conhecimentos

Neste módulo, é possível fornecer ao sistema conhecimentos relativos às relações semânticas existentes no conteúdo dos documentos da base de dados a ser pesquisada. Em seguida, o sistema estrutura as informações inseridas de forma que seja possível utilizá-las na fase de busca [3].

As operações descritas são executadas através de dois processos consecutivos descritos a seguir:

4.1.1 Criar uma ontologia

O primeiro passo é escrever uma ontologia para representar os conhecimentos sobre os documentos do banco de dados. O formato desta ontologia deve ser o KRSS [9], que foi escolhido pelos seguintes motivos:

- Permitir o trabalho com relações probabilísticas (inserir probabilidades na ontologia)
- Possuir uma sintaxe simples e clara que facilita a construção das ontologias

4.1.2 Gerar rede bayesiana

A partir da ontologia será gerada uma rede bayesiana. Isto deve ser feito para que seja possível lidar com as incertezas presentes nos dados e extrair informações relevantes sob a forma de probabilidades. Ou seja, com a rede bayesiana cria-se um modelo probabilístico dos conhecimentos descritos nas ontologias.

Uma forma de realizar tal operação de transformação é através do processo ilustrado na figura abaixo:

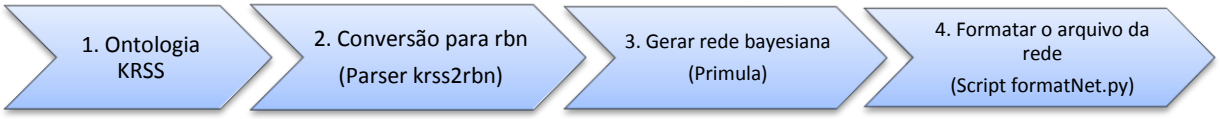


Figura 8: Processo para gerar uma rede bayesiana a partir de uma ontologia KRSS

1. Primeiramente é necessário salvar a ontologia escrita no formato KRSS em um arquivo de texto.
2. Em seguida, é preciso converter o seu formato para “RBN” [10], que também é uma forma de representar ontologias probabilísticas, mas que

possui uma sintaxe/formatação mais clara. Para realizar essa formatação pode ser utilizado o *parser* “krss2rbn” escrito em Java.

3. A próxima etapa é executada através do programa Primula [10], que constrói uma rede bayesiana no formato BIF [15] a partir do arquivo em RBN.
4. Finalmente, será necessário garantir que no arquivo BIF da rede gerada no passo anterior não existam caracteres especiais tais como acentos, letras maiúsculas e palavras com formatações irregulares. Por exemplo, expressões do tipo “possuiMotor” deve ser transformada em “possui motor”. Além disso, os nós auxiliares da rede recebem um prefixo igual a “aux.” para que eles possam ser identificados durante as inferências. Para facilitar esta operação, foi desenvolvido um *script* em python, denominado “formatNet.py”, que realiza todas as transformações descritas.

4.2 Pesquisas no banco de dados

No segundo módulo é executada a busca pelos documentos arquivados, utilizando informações extraídas da rede bayesiana criada no módulo anterior. Tal processo pode ser subdividido em quatro etapas:

4.2.1 Aquisição de documentos

Inicialmente é necessário um conjunto de documentos de texto que irão compor o banco de dados.

No programa desenvolvido o banco de dados é constituído por um diretório com uma série de arquivos de texto, onde cada um destes arquivos corresponde a um documento. Estes documentos foram retirados de artigos da Wikipidia e formatados com um *script* denominado “formatFiles.py”, escrito em python, garantindo-se que a formatação dos textos seja a mesma da rede bayesiana.

4.2.2 Realizar uma pesquisa

Para efetuar uma pesquisa devem ser inseridos termos chave sobre o assunto/documento que se pretende encontrar. Por exemplo, caso se deseje procurar textos sobre acidentes com aviões, pode-se formar um comando de busca “acidente avião”.

Além de palavras, é possível inserir também operadores lógicos que possibilitam a execução de uma busca avançada. Estes operadores estão descritos na tabela abaixo:

Operador	Descrição	Exemplo
AND	Utilizado para localizar documentos que possuam ambos os temos inseridos	jato ***AND*** supersônico
OR	Conjunção. Encontra arquivos que possuem um termo e/ou o outro termo	aeronave ***OR*** avião
NOT	Exclusão. Localiza textos que não possuem a palavra	avião ***NOT*** balão
“”	Permite pesquisas por expressões	“avião supersônico”
^	Atribui pesos aos temos pesquisados	aeronave***^0.5*** jato***^2***
+	Indica que é obrigatória a existência de um termo	**+**supersônico
~	Busca por palavras próximas	“jato supersônico”***~2*** (procura pelas palavras “jato” e “supersônico” que estejam distantes em até duas palavras)
*****	Operador “coringa” (substitui uma ou mais letras)	par***** (procura por palavras do tipo: "para", "pare", "paraquedas", etc.)

Tabela 4: Operadores lógicos

4.2.3 Expansão da pesquisa

Uma vez criada uma pesquisa, o sistema irá expandi-la com base nos conhecimentos previamente inseridos [5]. Isto significa acrescentar mais termos e/ou operadores de forma a melhorar os resultados.

Para realizar tal processo, será utilizada a rede bayesiana gerada no primeiro módulo. Nesta rede serão observados todos os nós que possuam o mesmo nome das palavras presentes na *query* do usuário. Em seguida, serão executadas múltiplas inferências nos demais nós da rede (com exceção dos nós auxiliares, que possuem o prefixo “aux.”). Assim serão obtidas duas listas, uma composta de palavras (correspondente aos nomes dos nós da rede bayesiana) e outra com os seus respectivos valores de probabilidades calculadas através das inferências. A figura abaixo ilustra o procedimento descrito:

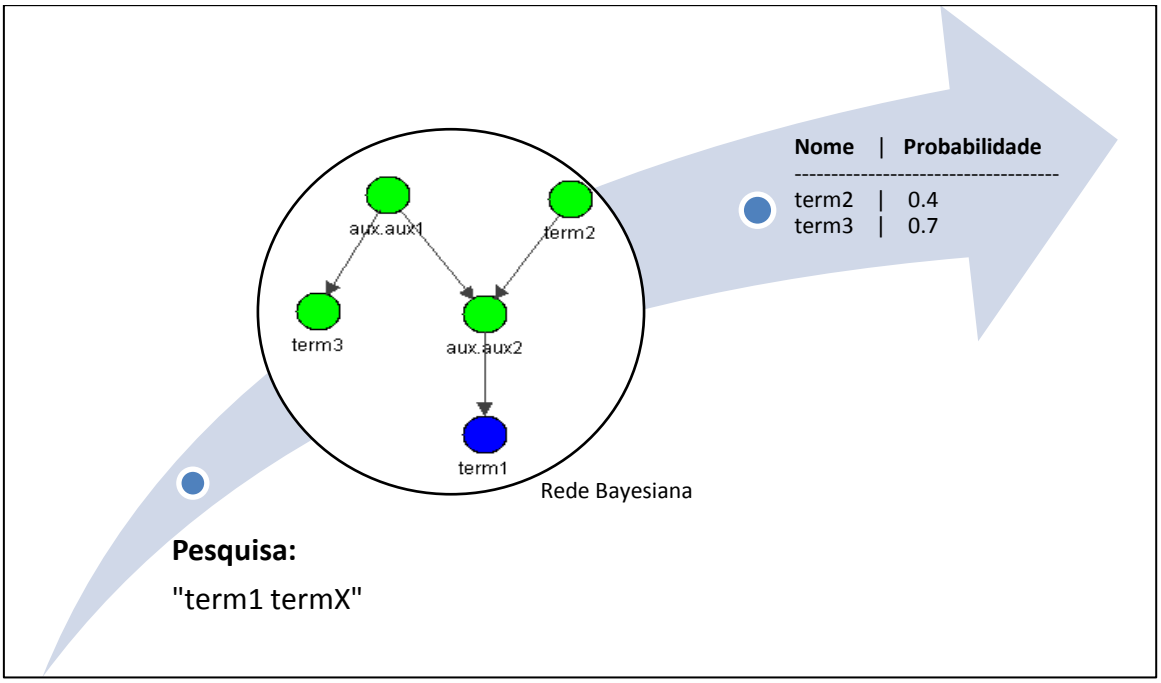


Figura 9: Inferências na rede bayesiana a partir de uma pesquisa. No exemplo para uma pesquisa “term1 termX” foi observado na rede o nó de nome “term1” (em azul) e em seguida foram extraído os valores das probabilidades dos demais nós não auxiliares (os nós auxiliares são indicados pelo prefixo “aux.”)

Com base nos resultados obtidos, os termos são adicionados de acordo com as seguintes regras:

- Se a probabilidade do termo for igual a 0 ou muito próxima de 0 (menor que 0.0001), o termo será adicionado à pesquisa junto com um operador de exclusão (NOT).

- Se a probabilidade for maior ou igual a 0.5, o termo será adicionado com um operador de conjunção (OR), sendo atribuído a ele um peso igual ao valor de sua probabilidade que atua como um fator de correção. Assim, se o termo for um sinônimo de alguma palavra procurada pelo usuário, o seu peso será igual a 1 (ou seja, será simplesmente adicionado um termo a mais na pesquisa como alternativa para a palavra inserida); se ele for apenas uma expressão relacionada ao que está sendo buscado, o termo também será procurado mas a sua a sua presença deverá ser menos relevante do que a dos termos buscados originalmente.

A seguir uma tabela que resume as regras definidas acima:

Probabilidade	Operador	Exemplo
≈ 0	NOT “*termo”*	“*termoOriginal* NOT *termoAdicionado*”
≥ 0.5	OR “*termo”^probabilidade*	“termoOriginal OR temoAdicionado^0.6”

Tabela 5: Regras para a expansão da pesquisa

4.2.4 Montar um ranking dos documentos

A próxima etapa consiste em selecionar e ordenar, de acordo com a relevância, os documentos da base que atendem a pesquisa expandida. Para tanto, os documentos devem ser analisados e, em seguida, deve ser gerado um valor (ou *score*) que permita avaliar o quão relevante o documento é para a pesquisa.

Para o sistema proposto o *score* será determinado segundo o modelo do Espaço de Vetores. Tal modelo foi escolhido pelas seguintes razões:

- O modelo do Espaço de Vetores comporta a inserção de pesos relacionados aos termos no cálculo do *score* de cada um dos textos.
- É um modelo tradicional bem conhecido e testado. Além disso, existem diversas ferramentas e bibliotecas que o implementam.

Para o desenvolvimento do programa foi escolhido o Lucene [12], que é uma biblioteca em Java com ampla variedade de métodos para se analisar os textos (extrair as palavras/termos dos documentos e construir um índice a partir deles) e oferece uma função para a construção do *ranking* baseada no modelo de Espaço de Vetores [7].

A função de score é dada pela seguinte equação:

$$Score(d) = coord_{d,q} \times queryNorm_q \times \sum_{t \in q} \left(tf_{d,t} \times idf_t \times boost_t \times norm_{t,d} \right) \quad (61)$$

Os seus termos correspondem a:

$$tf_{d,t} = \sqrt{f_{t,d}}. \quad (62)$$

$f_{t,d} =$ Número de vezes que o termo *t* aparece no documento *d.*

Este fator garante o *score* dos documentos que possuem mais ocorrências dos termos pesquisados seja maior.

$$idf_t = 1 + \log \frac{N}{1 + df_t}. \quad (63)$$

N = Número total de documentos.

$df_t =$ Número de documentos em que o termo *t* está presente.

Assim, quanto menor o número de textos com os termos da pesquisa, maior será a relevância daqueles que os possuírem.

$$coord_{d,q} = \frac{overlap}{maxOverlap}. \quad (64)$$

$overlap =$ Número dos termos pesquisados que estão contidos no documento.

$maxOverlap =$ Número total de termos pesquisados.

Em geral a presença de uma grande quantidade dos termos pesquisados em um texto indica que ele deve possuir um grau de relevância maior.

$$queryNorm_q = \frac{1}{\sqrt{ssw}}. \quad (65)$$

$$ssw = {boost_q}^2 \times \sum_{t \in q} (idf_t \times boost_t)^2. \quad (66)$$

$boost_q =$ Peso da pesquisa.

O valor do $boost_q$ é 1 na maior parte das vezes. Ele só será alterado no caso o usuário insira o valor desejado através da inclusão de parâmetros especiais na própria sentença da pesquisa.

$$norm_{t,d} = boost_d \times Ln_{field} \times \prod_{\substack{campo\ \ f\ do \\ termo\ \ t\ de\ d}} boost_f. \quad (67)$$

$boost_d$ = Peso do documento *d*. Para a presente aplicação este fator terá sempre o valor de 1.

$boost_f$ = Peso do campo. O campo corresponde a uma área específica do texto, como por exemplo, o título.

Ln_{field} = Tamanho do campo contabilizado de acordo com o número de termos.

$$boost_t = P(t_{ont}). \tag{68}$$

$boost_t$ = Peso do termo *t.*

$P(t_{ont})$ = Se o termo estiver contido na pesquisa original ele assumirá o peso máximo de 1. Caso contrário, o seu valor será igual ao da probabilidade obtida como resultado da inferência na rede bayesiana.

4.3 Agrupamento e exibição dos resultados

Uma vez localizados e ordenados os documentos relevantes para uma pesquisa, o sistema proposto deve classificá-los de acordo com seus respectivos assuntos e montar um gráfico de relações a ser exibido para o usuário. Tal procedimento pode ser subdividido em duas fases, descritas a seguir.

4.3.1 Detecção de características

Nesta etapa serão detectados os diferentes assuntos existentes nos textos buscados, tendo como base as respostas obtidas para uma dada pesquisa, a partir das quais será gerado um novo índice. Isto é equivalente a montar uma matriz contendo a relação de quantas vezes cada termo aparece em cada documento (Matriz de documentos). Assim, utilizando o algoritmo *Non-matrix Factorization* [14, 18] é possível decompor a matriz em outras duas:

- **Matriz de características**: As linhas correspondem às características e as colunas aos termos. Os valores indicam o quanto um termo está associado a uma característica.

- **Matriz de pesos**: Cada linha corresponde a um documento e cada coluna a uma característica. Os valores indicam o quanto um documento está relacionado a uma característica.

Desta forma, através da verificação dos valores encontrados nestas matrizes, pode-se agrupar tanto os documentos quanto os próprios termos em grupos de características (ou assuntos). Isto pode ser realizado identificando os maiores valores das matrizes.

$$[Matriz\ de\ características] \times [Matriz\ de\ pesos] = [Matriz\ de\ documentos]. \quad (69)$$

Vale comentar que, para a utilização do algoritmo, é necessário estipular o número de características que devem ser encontradas. Tal número varia de acordo com a base de dados e com a pesquisa realizada. Portanto, no programa desenvolvido, este valor deverá ser inserido pelo usuário. Além disso, no *software* desenvolvido foi utilizada a biblioteca Jama [14] para otimizar os cálculos realizados com matrizes de grande porte.

4.3.2 Exibição dos resultados

Finalmente, a última etapa que o sistema proposto deve executar é a de organizar as respostas obtidas, auxiliando o usuário a encontrar os documentos desejados. Assim, além da tradicional lista de documentos ordenados de acordo com a sua relevância, foi desenvolvida uma representação gráfica que exibe os resultados agrupando-os conforme seus respectivos assuntos (características).

A representação gráfica é gerada a partir das características encontradas na primeira etapa deste módulo. Para cada uma destas características (ou assuntos) existirá uma série de termos e de textos relacionados de forma que o gráfico a ser construído consistirá em ligar os nós destes documentos aos nós dos termos.

As figuras a seguir ilustram os dois modos de exibição das respostas: uma lista em que os textos estão ordenados de acordo com o seu *score* (relevância) e um gráfico em que se tenta identificar assuntos/características comuns nos resultados:

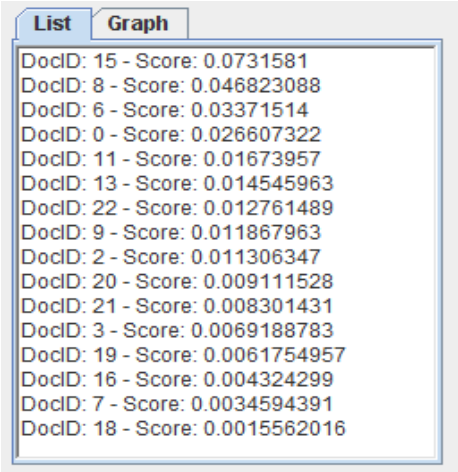


Figura 10: Lista de resultados ordenados de acordo com sua relevância para a pesquisa

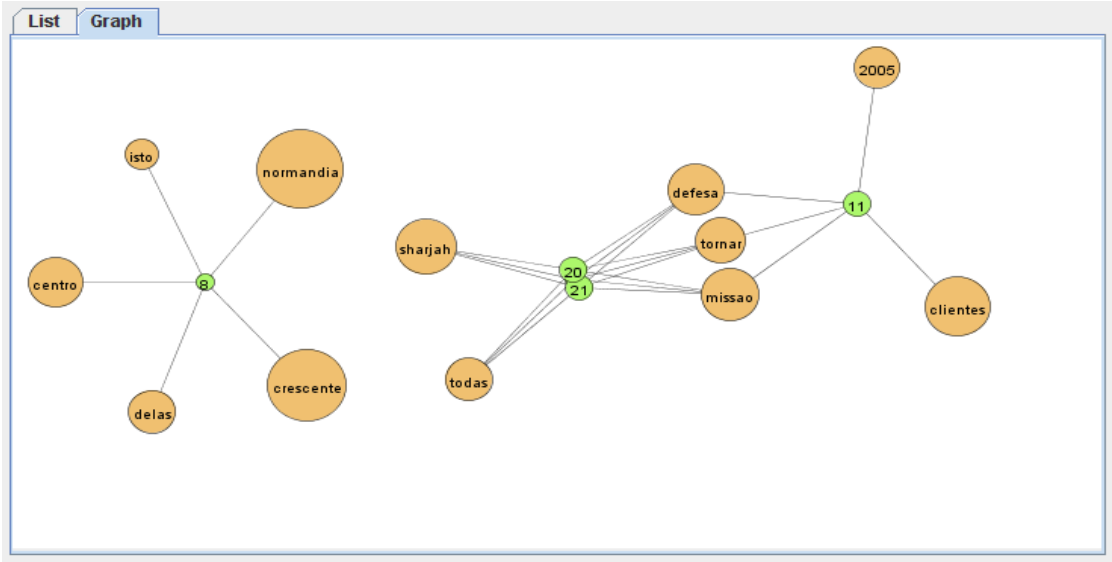


Figura 11: Gráfico de agrupamento das respostas por assuntos

A disposição dos nós do gráfico das características, representado acima, são determinadas primeiramente pelo um algoritmo denominado *relaxation* [18, 19] e em seguida refinada pelo método de otimização *simulated annealing* (ou recozimento simulado) [22].

5 Testes e análise dos resultados

Para a avaliação do sistema proposto foram realizados três experimentos com o *software* desenvolvido. Em cada um destes testes foram consideradas duas situações distintas: uma em que ocorre a expansão da pesquisa e outra na qual a pesquisa do usuário não sofre nenhuma alteração.

A partir dos resultados obtidos, tanto para o sistema sem a expansão como para o com expansão, foram construídas curvas de precisão x retorno (*precision x recall*).

Para a construção do gráfico de avaliação foi desenvolvido um programa em python denominado “11Eval.py”. Este programa recebe arquivos de texto contendo as respostas obtidas pelos sistemas e um arquivo com as respostas esperadas (o que os sistemas deveriam responder para cada uma das pesquisas). Com base nestes dados ele constrói a curva de precisão x retorno e, em seguida, aplica a técnica de *11-point averaged precision* de interpolação (descrita detalhadamente no item 3).

A seguir a descrição dos experimentos:

1. No primeiro teste foi construído um banco composto por 25 artigos sobre aeronaves extraídos da Wikipedia. Depois, com base nos artigos coletados, foi desenvolvida uma ontologia para representar as classes das aeronaves, ou seja, uma ontologia criada manualmente para representar conhecimentos específicos sobre o assunto dos textos da base (anexo 1).

   A seguir o gráfico obtido para um teste composto por três pesquisas distintas. O “IR 0” representa a curva do sistema sem expansão da pesquisa e o “IR 1” representa aquele que empregou a expansão:

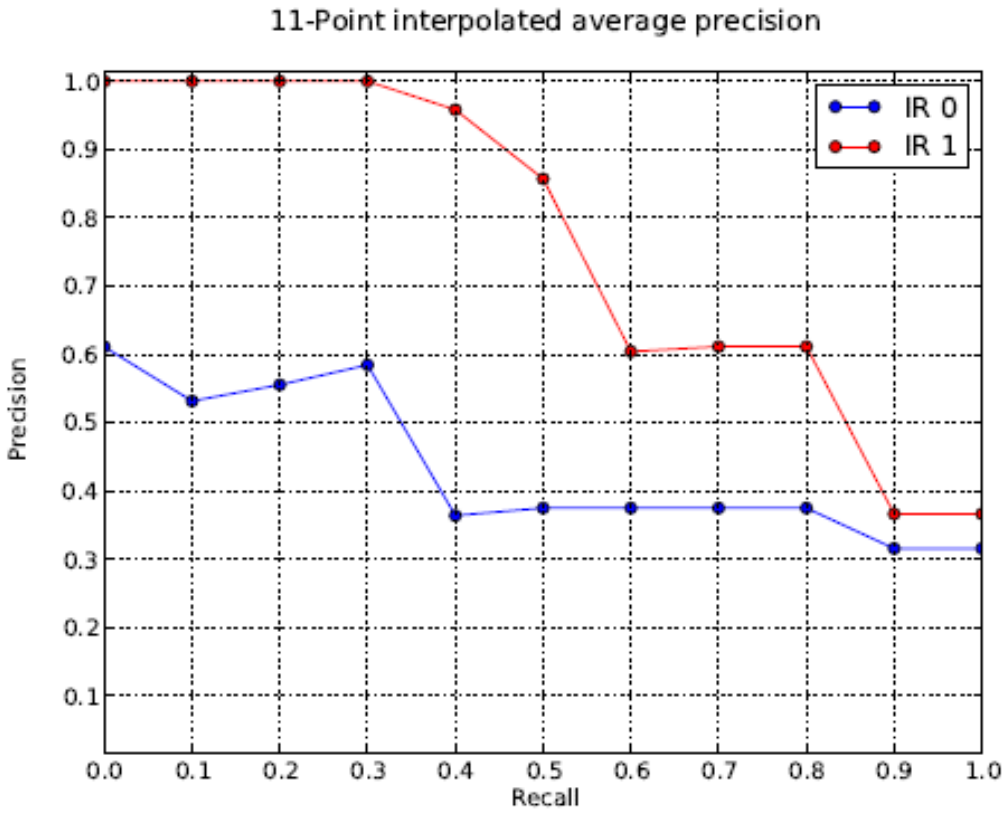


Figura 12: Precisão x Retorno – Teste para a coleção de 25 documentos. IR 0 é o modelo sem a expansão da pesquisa, o IR 1 é o modelo com a expansão

Com base no gráfico, é possível observar que, para as pesquisas realizadas, os resultados obtidos com a expansão da pesquisa atingiram índices de precisão superiores para todos os níveis de retorno.

2. Para o segundo teste foi utilizada a coleção denominada *Reuters Corpus* [21] composta por aproximadamente 10.000 notícias sobre diversos temas. Para a construção da ontologia foi desenvolvido um programa em python (wnTest.py) que gera um arquivo em RBN a partir das relações existentes na WordNet [22]. Vale comentar que este procedimento garantia ontologias limitadas e com relacionamentos fracos (apenas definições de classes e sub-classes). Além disso, por se tratar de um procedimento automático, há pouco controle do conhecimento que é inserido em cada uma das ontologias, de forma que não é possível representar exatamente as informações existentes na coleção de textos.

O objetivo do teste é apenas comparar os resultados do modelo que utiliza os conhecimentos representados nas ontologias e da que não os utiliza. Assim, não foi necessário avaliar a relevância de todos os documentos existentes na coleção, apenas daqueles que os sistemas retornaram. Desta forma o conjunto de textos relevantes foi aproximado pela união das respostas válidas dos dois sistemas.

Por exemplo, se para uma dada pesquisa o primeiro sistema retornou os documentos “a”, “b” e “c”, sendo “a” e “b” relevantes para a pesquisa, e o segundo sistema obteve como resposta “b”, “d”, “e” e “f”, com “b” e “f” avaliados relevantes. Então, para a situação descrita, considerou-se que todos os documentos relevantes existentes na coleção são: “a”, “b” e “f”.

Portanto, os resultados apresentados a seguir servem unicamente para a comparação dos modelos, e não devem ser utilizados para avaliar o desempenho individual dos sistemas.

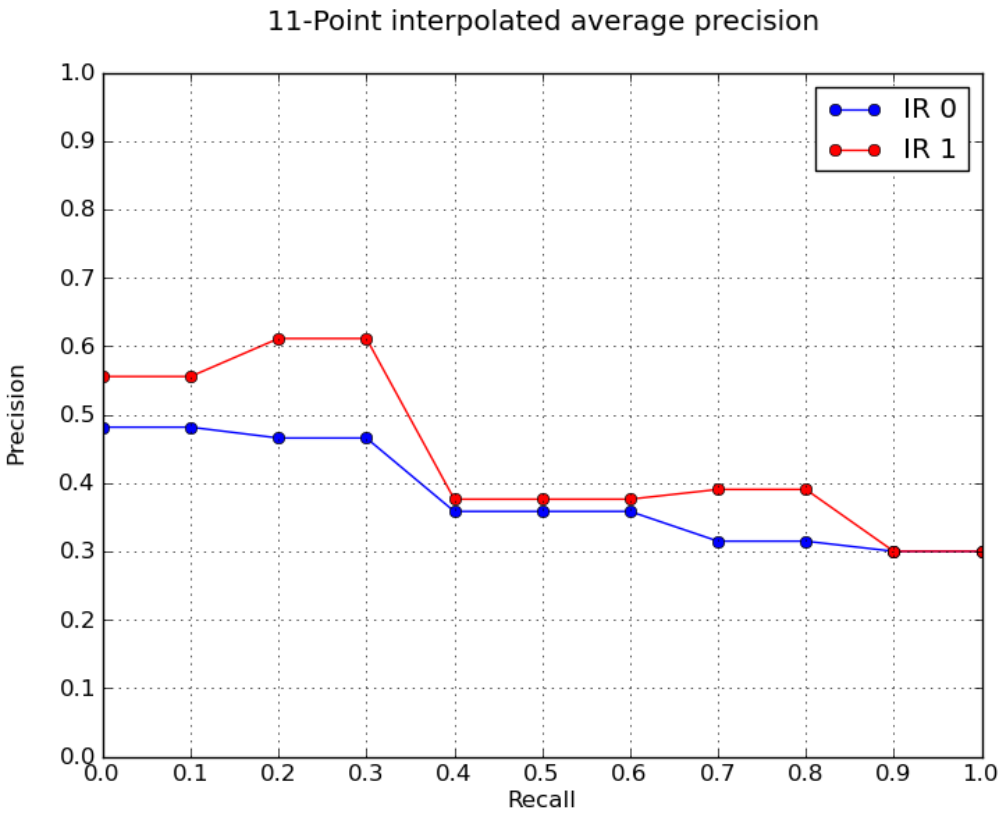


Figura 13: Precisão x Retorno - Teste com a coleção de 10.000 documentos. IR 0 representa o modelo sem a expansão da pesquisa e o IR 1 o modelo com a expansão

De acordo com os resultados obtidos, pode-se concluir que, para este teste, o modelo com a expansão da pesquisa obteve resultados ligeiramente melhores, sendo que os seus índices de precisão se mantiveram um pouco acima para cada um dos níveis de retorno.

3. O terceiro experimento realizado seguiu o mesmo procedimento do segundo teste. A única diferença foi dada na escolha da ontologia, uma vez que, propositalmente, o seu assunto (o conhecimento representado) não tinha uma relação com a pesquisa inserida. Através deste teste buscou-se verificar qual seria o comportamento do sistema para uma ontologia inadequada e/ou que não possuísse informações sobre o tema pesquisado.

   O gráfico a seguir ilustra os resultados obtidos:

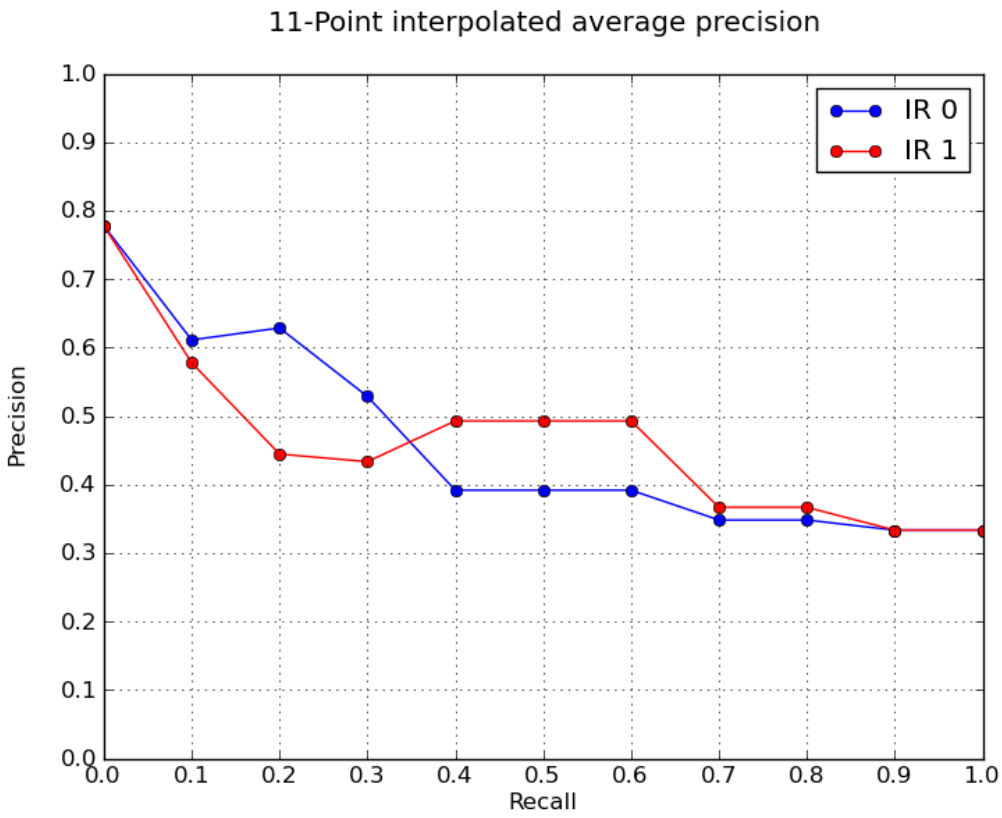


Figura 14: Precisão x Retorno - Teste com a coleção de 10.000 documentos e com uma ontologia mal formulada. IR 0 representa o modelo sem a expansão da pesquisa e IR 1 o modelo com a expansão

Neste teste verificou-se que, para os primeiros níveis de retorno, o sistema com a pesquisa expandida apresentou índices de precisão inferiores. Isto ocorreu porque os termos inseridos pela ontologia mal formulada acabaram elevando o *score* de documentos não relevantes. Contudo, também é possível observar que, principalmente para os últimos níveis de retorno, não há uma grande diferença entre os modelos uma vez que os pesos dos termos adicionados foram muito baixos de forma que eles influenciaram pouco a construção do *ranking*. De forma geral, a maior diferença observada deu-se na ordenação dos textos considerados relevantes.

Contudo, vale ressaltar que testes com ontologias inadequadas ainda devem ser objetos de estudos futuros, dado a grande diversidade de situações que os envolvem.

6 Conclusão

Neste projeto foi possível desenvolver um modelo que permite ampliar o modelo de recuperação de informações do Espaço de Vetores. Tal modelo consiste na implementação de uma expansão de pesquisa baseada em informações previamente inseridas sob o formato de ontologias.

Após uma série de testes, verificou-se que o modelo proposto pode melhorar o desempenho dos sistemas de busca que utilizam o Espaço de Vetores. Contudo observou-se também que tal melhora esta vinculada a uma série de fatores e, dependendo das condições, os resultados obtidos podem ser até mesmo piores.

Dentre os fatores mencionados destaca-se a forma com que a ontologia é construída e selecionada para uma pesquisa. A qualidade dos resultados de uma busca depende diretamente de quão adequada é a ontologia, ou seja, se ela consegue representar bem as informações relacionadas ao assunto que se está procurando.

Além da construção e da escolha da ontologia, a forma como usuário insere a pesquisa também interfere no desempenho do sistema. Se ele tiver um conhecimento prévio de como está estruturado a base de conhecimentos e dos conceitos contidos nela, será possível efetuar uma busca que tire o máximo de proveito do modelo proposto. Isto acontece porque, quando os termos da pesquisa são iguais aqueles dos nós da rede bayesiana utilizada, o programa conseguirá estimar melhor as probabilidades e, consequentemente, efetuar uma expansão da pesquisa de modo mais completo.

O instrumento construído tem o mérito de apresentar uma alternativa de sistemas de busca capaz de considerar as relações semânticas, o que amplia a qualidade das respostas e, ainda, com a vantagem de aproveitar os modelos clássicos de recuperação de informações. Isso significa um avanço sobre as tentativas de sistemas semânticos que pressupõem uma organização textual prévia em conformidade com o sistema (por exemplo, com palavras chave).

7 Referências Bibliográficas

[1] D. MANNING, Christopher, RAGHAVAN, Prachakar, SCHÜTZE, Hinruch. Introduction to Information Retrieval. Cambrige University Press. New York, 2008.

[2] GOES, Monica, E. Especificação de Produtos e Partes em Manufatura Utilizando Ontologias Baseadas em Lógica e Probabilidades. São Paulo, 2005

[3] CHAUDHURI, Surajit, DAS, Gautam, HRISTIDIS, Vangelis, WEIKUM, Gerhard. Probabilistic Information Retrieval for Ranking of Database Query Results. http://ranger.uta.edu/~gdas/websitepages/preprints-papers/PIR-vldb.pdf consultado em 16/06/2009.

[4] SINGHAL, Amit. Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 2001.

[5] PAZ-TRILLO, Christian, WASSERMANN, Renata, BRAGA, Ana P. An Information Retrieval using Ontologies. São Paulo, 2005.

[6] D. Lin. Information-theoretic definition of similarity. Em *Proceedings of the 15th International Conference on Machine Learning,* páginas 296-304. Morgan Kaufmann Publishers Inc. San Francisco, 1998.

[7] VALLET, David, FERNÁNDEZ, Mirian, CASTELLS, Pablo. Ontology-Based Retrieval Model. http://www.acemedia.org/aceMedia/files/document/wp7/2005/eswc05-uam.pdf consultado em 16/06/2009. Madri.

[8] ROZENFELD, Henrique. Sistemas de Classificação e Tecnologia de grupo. http://www.numa.org.br/conhecimentos/conhecimentos_port/pag_conhec/TG_Class_Produtos.htm cosultado em 16/06/2009.

[9] PATEEL-SCHNEIDER, Peter F., SWARTOUT, Bill.Description-Logic Knowlwdge Representation System Specification from the KRSS Group of the ARPA Knowledge Sharing Effort. http://dl.kr.org/krss-spec.ps consultado em 24/09/2009.

[10] JAEGER, Manfred. The Primula System: User's Guide. http://www.cs.aau.dk/~jaeger/Primula/primula-manual2.2.pdf consultado em 24/09/2009.

[11] COZMAN, Fabio Gagliardi, POLANTRO, Rodrigo Bellizia. Loopy propagation in a probabilistic description logic. In SUM, páginas 120-133, 2008.

[12] Lucene 2.4.1 API. http://lucene.apache.org/java/2_4_1/api/index.html consultado em 24/09/2009.

[13] Package Jama. http://math.nist.gov/javanumerics/jama/doc/ consultado em 24/09/2009.

[14] LEE, Daniel D., SEUNG, H. Sebastian. Algorithms for Non-negative Matrix actorization http://hebb.mit.edu/people/seung/papers/nmfconverge.pdf consultado em 24/09/2009.

[15] COZMAN, Fabio Gagliardi, JavaBayes User Manual. http://www.cs.cmu.edu/~javabayes/Home/index.html consultado em 14/09/2009.

[16] ARTERO, Almir Olivette. Inteligência Artificial: Teórica e Prática. Editora Livraria da Física. São Paulo, 2009.

[17] FAZZINGA, Bettina, GIONFORME, Giorgio, GOTTLO, Georg, LUKASIEWICZ, Thomas. From Web Search to Semantic Web Search. INFSYS Research Report 1843-08-11. Viena, Áustria, 2009.

[18] SEGARAN, Toby. Programando a Inteligência Coletiva. Editora Ata Books. Rio de Janeiro, 2008.

[19] FRY, Ben. Visualizando Dados. Editora Alta Books.Rio de Janeiro, 2008.

[20] KLEIN, Steven Bird, LOPER, Edward. Natural Language Processing in Python. O'Reilly Media. Junho, 2009.

[21] WordNet 3.0 Reference Manual. http://wordnet.princeton.edu/wordnet/documentation/ consultado em 20/11/2009.

[22] JÚNIOR, Cairo Lúcio Nascimento, YONEYAMA, Takashi. Inteligência Artificial em Controle e Automação. Editora Blucher. São Paulo, 2004.

Anexos

8.1 Ontologia criada manualmente

A seguir uma ontologia sobre aeronaves no formato RBN utilizada para a execução do Teste 1.

```
Aeronave(x) = 0.9;

Aerostato(x) = ( Aeronave(x):0.1,0 );
Aerodino(x) = ( Aeronave(x): (Aerostato(x):0,1), 0);

Militar(x) = ( Aeronave(x):0.4,0 );
Civil(x) = ( Aeronave(x): (Militar(x):0,1), 0);

temPropulsaoPropria(x) = n-or{ (Aerostato(x):0.4,0), (Aerodino(x):0.85,0) | :
x=x};

PropulsorMotor(x) = n-or{ (temPropulsaoPropria(x):(Aerostato(x):1,0),0),
(temPropulsaoPropria(x):(Aerodino(x):0.5,0),0) | : x=x};

PropulsorTurbina(x) = (temPropulsaoPropria(x):
((Aerostato(x):0,1):(PropulsorMotor(x):0,1),0) ,0.4);

PropulsorAsaRotativa(x) = (temPropulsaoPropria(x): ((Aerostato(x):0,1):(
(PropulsorTurbina(x):0,1):(PropulsorMotor(x):0,1),0 ),0) ,0.1);

Dirigivel(x) = (Aerostato(x):PropulsorMotor(x),0);
Balao(x) = (Aerostato(x):( temPropulsaoPropria(x):0,1 ),0);
Planador(x) = (Aerodino(x):( temPropulsaoPropria(x):0,1 ),0);
Helicoptero(x) = (Aerostato(x):PropulsorAsaRotativa(x),0);
Aviao(x) = (Aerodino(x):(
temPropulsaoPropria(x):(PropulsorAsaRotativa(x):0,1),0 ),0);


Jato(x) = ( PropulsorTurbina(x):1,0 );
Zepelim(x) = ( Dirigivel(x):0.99,0 );

% Helicopteros militares
Apache(x) = ( Helicoptero(x): (Militar(x):0.8,0), 0);
AH-64(x) = ( Apache(x):0.9,0 );

% Avioes militares
SuperHornet(x) = ( Jato(x): (Militar(x):0.7,0), 0);

% Dirigiveis
ADB-1(x) = ( Dirigivel(x):0.2,0 );
ADB-2(x) = ( Dirigivel(x):0.2,0 );
ADB-3(x) = ( Dirigivel(x):0.2,0 );
```

8.2 Ontologias criadas automaticamente

A seguir as ontologias (em RBN), criadas de forma automática, que forram utilizadas no Teste 2:

8.2.1 Ontologia de combustíveis:

Fuel(x) = 0.9;

Coke(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Combustible(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Propane(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Combustible(x):0,1):(Coke(x):0,1),0),0);

Fire(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

RedFire(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Firewood(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Illuminant(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Biomass(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

CoalGas(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

DieselOil(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

FossilFuel(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):(Coke(x):0,1),0),0),0),0),0),0),0),0),0),0),0);

WaterGas(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((Coke(x):0,1):(FossilFuel(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0);

NuclearFuel(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

Igniter(x) = n-or{ (Fuel(x):0.5,0) | y : x=x};

PineKnot(x) = n-or{ (Firewood(x):0.5,0) | y : x=x};

Coal(x) = n-or{ (FossilFuel(x):0.5,0) | y : x=x};

NaturalGas(x) = n-or{ (FossilFuel(x):0.5,0) | y : x=x};

Brand(x) = n-or{ (Firewood(x):0.5,0) | y : x=x};

TownGas(x) = n-or{ (CoalGas(x):0.5,0) | y : x=x};

Kindling(x) = n-or{ (Igniter(x):0.5,0) | y : x=x};

Cordwood(x) = n-or{ (Firewood(x):0.5,0) | y : x=x};

Derv(x) = n-or{ (DieselOil(x):0.5,0) | y : x=x};

Firelighter(x) = n-or{ (Igniter(x):0.5,0) | y : x=x};

Punk(x) = n-or{ (Igniter(x):0.5,0) | y : x=x};

Backlog(x) = n-or{ (Firewood(x):0.5,0) | y : x=x};

Lignite(x) = n-or{ (Coal(x):0.5,0) | y : x=x};

SteamCoal(x) = (n-or{ (Coal(x):0.5,0) | y : x=x} : (Lignite(x):0,1),0);

Anthracite(x) = (n-or{ (Coal(x):0.5,0) | y : x=x} : ((Lignite(x):0,1):(SteamCoal(x):0,1),0),0);

YuleLog(x) = n-or{ (Backlog(x):0.5,0) | y : x=x};

Charcoal(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):(WaterGas(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

BituminousCoal(x) = (n-or{ (Coal(x):0.5,0) | y : x=x} : ((Lignite(x):0,1):((SteamCoal(x):0,1):(Anthracite(x):0,1),0),0),0);

CannelCoal(x) = n-or{ (BituminousCoal(x):0.5,0) | y : x=x};

SeaCoal(x) = (n-or{ (BituminousCoal(x):0.5,0) | y : x=x} : (CannelCoal(x):0,1),0);

Methanol(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):(WaterGas(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

Jet(x) = n-or{ (Lignite(x):0.5,0) | y : x=x};

Kerosene(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((Methanol(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):(WaterGas(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

Gasohol(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((Methanol(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):((WaterGas(x):0,1):(Kerosene(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

Gasoline(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((Methanol(x):0,1):((Gasohol(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):((WaterGas(x):0,1):(Kerosene(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

UnleadedGasoline(x) = n-or{ (Gasoline(x):0.5,0) | y : x=x};

Butane(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Gasoline(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((Methanol(x):0,1):((Gasohol(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):((WaterGas(x):0,1):(Kerosene(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

Paraffin(x) = n-or{ (Kerosene(x):0.5,0) | y : x=x};

Napalm(x) = (n-or{ (Gasoline(x):0.5,0) | y : x=x} : (UnleadedGasoline(x):0,1),0);

LeadedGasoline(x) = (n-or{ (Gasoline(x):0.5,0) | y : x=x} : ((UnleadedGasoline(x):0,1):(Napalm(x):0,1),0),0);

Petroleum(x) = (n-or{ (FossilFuel(x):0.5,0) | y : x=x} : ((Coal(x):0,1):(NaturalGas(x):0,1),0),0);

FuelOil(x) = (n-or{ (Fuel(x):0.5,0) | y : x=x} : ((Biomass(x):0,1):((Firewood(x):0,1):((Combustible(x):0,1):((DieselOil(x):0,1):((CoalGas(x):0,1):((Butane(x):0,1):((RedFire(x):0,1):((Fire(x):0,1):((Gasoline(x):0,1):((Illuminant(x):0,1):((Propane(x):0,1):((NuclearFuel(x):0,1):((Coke(x):0,1):((Methanol(x):0,1):((Gasohol(x):0,1):((FossilFuel(x):0,1):((Igniter(x):0,1):((Charcoal(x):0,1):((WaterGas(x):0,1):(Kerosene(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0),0);

GasOil(x) = n-or{ (FuelOil(x):0.5,0) | y : x=x};

ResidualOil(x) = n-or{ (Petroleum(x):0.5,0) | y : x=x};

8.2.2 **Ontologia de navios**:

Ship(x) = 0.9;

Wreck(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Troopship(x) = (n-or{ (Ship(x):0.5,0) | y : x=x} : (Wreck(x):0,1),0);

AbandonedShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Minelayer(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Steamer(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Nuclear-PoweredShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Minesweeper(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Hulk(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Shipwreck(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Lightship(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

SchoolShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Warship(x) = (n-or{ (Ship(x):0.5,0) | y : x=x} : ((Minesweeper(x):0,1):((Shipwreck(x):0,1):((Nuclear-PoweredShip(x):0,1):((Hulk(x):0,1):((Wreck(x):0,1):((SchoolShip(x):0,1):((Troopship(x):0,1):((Minelayer(x):0,1):((Lightship(x):0,1):((Steamer(x):0,1):(AbandonedShip(x):0,1),0),0),0),0),0),0),0),0),0),0),0),0);

Flagship(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Pirate(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Icebreaker(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Blockade-Runner(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Gas-TurbineShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

SisterShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

SlaveShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Three-Decker(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

TransportShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Whaler(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

CargoShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

HospitalShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

SmallShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

Tender(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

TreasureShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

PassengerShip(x) = n-or{ (Ship(x):0.5,0) | y : x=x};

PaddleSteamer(x) = n-or{ (Steamer(x):0.5,0) | y : x=x};

Man-Of-War(x) = n-or{ (Warship(x):0.5,0) | y : x=x};

LibertyShip(x) = n-or{ (CargoShip(x):0.5,0) | y : x=x};

DestroyerEscort(x) = (n-or{ (Warship(x):0.5,0) | y : x=x} : (Man-Of-War(x):0,1),0);

AircraftCarrier(x) = (n-or{ (Warship(x):0.5,0) | y : x=x} : ((DestroyerEscort(x):0,1):(Man-Of-War(x):0,1),0),0);

Three-Decker(x) = (n-or{ (Warship(x):0.5,0) | y : x=x} : ((DestroyerEscort(x):0,1):((Man-Of-War(x):0,1):(AircraftCarrier(x):0,1),0),0),0);

Battleship(x) = (n-or{ (Warship(x):0.5,0) | y : x=x} : ((DestroyerEscort(x):0,1):((Man-Of-War(x):0,1):((AircraftCarrier(x):0,1):(Three-Decker(x):0,1),0),0),0),0);

Corvette(x) = (n-or{ (Warship(x):0.5,0) | y : x=x} : ((Battleship(x):0,1):((DestroyerEscort(x):0,1):((Man-Of-War(x):0,1):((AircraftCarrier(x):0,1):(Three-Decker(x):0,1),0),0),0),0),0);

ContainerShip(x) = n-or{ (CargoShip(x):0.5,0) | y : x=x};

8.2.3 Ontologia de países em guerra:

State(x) = 0.9;

WorldPower(x) = n-or{ (State(x):0.5,0) | y : x=x};

CityState(x) = n-or{ (State(x):0.5,0) | y : x=x};

Reich(x) = n-or{ (State(x):0.5,0) | y : x=x};

Dominion(x) = n-or{ (State(x):0.5,0) | y : x=x};

Suzerain(x) = n-or{ (State(x):0.5,0) | y : x=x};

ForeignCountry(x) = n-or{ (State(x):0.5,0) | y : x=x};

RogueState(x) = n-or{ (State(x):0.5,0) | y : x=x};

Ally(x) = n-or{ (State(x):0.5,0) | y : x=x};

CommonwealthCountry(x) = n-or{ (State(x):0.5,0) | y : x=x};

DevelopingCountry(x) = n-or{ (State(x):0.5,0) | y : x=x};

SeaPower(x) = n-or{ (State(x):0.5,0) | y : x=x};

HohenzollernEmpire(x) = n-or{ (Reich(x):0.5,0) | y : x=x};

ThirdReich(x) = n-or{ (Reich(x):0.5,0) | y : x=x};

Hegemon(x) = n-or{ (WorldPower(x):0.5,0) | y : x=x};

War(x) = 0.9;

WorldWar(x) = n-or{ (War(x):0.5,0) | y : x=x};

Jihad(x) = n-or{ (War(x):0.5,0) | y : x=x};

BiologicalWarfare(x) = n-or{ (War(x):0.5,0) | y : x=x};

HotWar(x) = n-or{ (War(x):0.5,0) | y : x=x};

LimitedWar(x) = n-or{ (War(x):0.5,0) | y : x=x};

CivilWar(x) = n-or{ (War(x):0.5,0) | y : x=x};

InformationWarfare(x) = n-or{ (War(x):0.5,0) | y : x=x};

ChemicalWarfare(x) = n-or{ (War(x):0.5,0) | y : x=x};

PsychologicalWarfare(x) = n-or{ (War(x):0.5,0) | y : x=x};

GermWarfare(x) = n-or{ (BiologicalWarfare(x):0.5,0) | y : x=x};