UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ALINE GRACIELA LERMEN DOS SANTOS

JESS – Uma ferramenta para detecção de linguagem em textos

Trabalho de Graduação.

Prof. Dr. Leandro Krug Wives
Orientador

Porto Alegre, dezembro de 2012.

“May the Force be with you.”

AGRADECIMENTOS

Gostaria de agradecer primeiramente à minha família, que me apoiou durante toda a vida e tornou minha graduação possível; aos meus colegas de curso e amigos, sem eles a minha experiência na universidade não seria a mesma; ao Guilherme R., por estar presente durante esses anos, me apoiando e ajudando quando necessário; e ao meu orientador, pelo bom trabalho desenvolvido.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

RI	Recuperação da Informação
IR	Information Retrieval
TF	Term-Frequency (Frequência do Termo)
IDF	Inverse Document Frequency (Frequência Inversa de Documentos)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Com a expansão da Internet nos últimos anos, surgem textos na *web* de diversas fontes, acarretando um aumento na quantidade de idiomas. Ao se deparar com um destes textos, um problema relevante consiste em saber qual é o idioma dele. Este trabalho tem como objetivo abordar o problema da identificação do idioma de textos, como *posts* de blog e jornais *online*, entre inglês e português, utilizando técnicas e abordagens provenientes da área de Recuperação de Informação. A detecção de linguagem é uma variação da classificação de textos, e aqui será utilizado o treinamento com aprendizado supervisionado. A detecção de idioma é feita por três esquemas de ponderação diferentes: *tf-idf*, booleano e *tf*, além de analisar o melhor desempenho quanto ao uso de palavra inteira ou n-gramas. Com essas técnicas, deseja-se saber qual análise terá melhor resultado na classificação, através da quantidade de acertos de cada uma. Os experimentos realizados mostram que a análise com palavras inteiras e com peso booleano possui o melhor desempenho no menor tempo de processamento.

Palavras-Chave: detecção de linguagem, textos online, n-gramas, esquemas de ponderação.

JESS - A tool to detect language in texts

ABSTRACT

With the growth of the Internet in recent years, web texts arise from various sources, resulting in multiple languages as well. When facing one of these texts, one important issue is related to knowing the language used in a specific document. This work aims to approach the problem of identifying the language of online texts, as blog posts and online newspapers, between English and Portuguese, using techniques and approaches from Information Retrieval for this. Language detection is a variation of text classification, and here will be used training with supervised learning. The detection is performed by three different weighting schemes: tf-idf, boolean and tf weights, besides analyzing the best performance over the use of whole word or n-grams. With these techniques, it is desirable to know which analysis will result in better classification by the number of hits each one gets. The experiments show that the analysis with whole words and the boolean weight has the best performance in the shortest processing time.

Keywords: language detection, online text, n-grams, weighting schemes.

1 INTRODUÇÃO

Em 2009, Lewis (2009), catalogou 6909 linguagens naturais humanas sendo usadas. Uma fração desse número, em torno de 145 idiomas (W3TECHS, 2012), faz parte do conteúdo da Internet hoje em dia, gerando uma grande diversidade de linguagens nas quais textos são escritos.

Saber em qual linguagem determinado texto está escrito pode ser importante: determinados assuntos não possuem informações em um dado idioma, por exemplo, e cabe ao usuário traduzir um texto proveniente de outra linguagem para a sua linguagem natural, mas não se pode traduzir esse texto sem saber em que idioma ele foi escrito; ainda, a linguagem de um texto pode ser a chave para categorização de textos de acordo com o idioma. Outro exemplo é a identificação da linguagem de uma página *web* pelo navegador, e caso difira da linguagem padrão escolhida pelo usuário, trabalhando em conjunto com um tradutor automático, o navegador pode oferecer a visualização de uma tradução dessa página. Essa é uma funcionalidade já encontrada no navegador Chrome, da Google.

Outras possibilidades de aplicações para a detecção de linguagem são (i) a detecção do idioma usado em editores de textos e a (ii) detecção do idioma usado em um documento oficial. Para o caso (i), tem-se como exemplo o corretor de texto para verificação ortográfica da ferramenta Word, do pacote Office da Microsoft. No caso, se a linguagem na qual está sendo escrito o texto não for detectada corretamente, o corretor não cumprirá o seu papel, podendo acusar erros de acordo com outros idiomas. Para o caso (ii), pode-se citar o fato de que a União Europeia possui 23 idiomas oficiais no Parlamento Europeu e todos os documentos são publicados em todas as 23 linguagens, gerando um volume mensal de 100.000 páginas para 700 tradutores e 260 assistentes (EUROPARL, 2012). Seria interessante que houvesse uma ferramenta capaz de detectar os idiomas desses documentos que funcionasse em conjunto com um tradutor automático para automatizar o trabalho e torná-lo mais ágil.

O objetivo deste trabalho é detectar se o idioma usado em textos eletrônicos (no caso foram usados textos de notícias eletrônicas) – é o inglês ou o português. Para tanto, foi desenvolvida uma ferramenta denominada de Jess.

O foco deste trabalho se encontra nos problemas de classificação de textos, que consistem em definir uma ou mais classes para um texto. O objetivo é detectar o idioma de textos com três métodos diferentes – analisando palavras inteiras com peso *tf-idf*, palavras inteiras com peso booleano e n-gramas com peso *tf* – e comparar qual dos métodos é o mais eficiente. A abordagem tomada para atingir o objetivo é: dado um texto de entrada, cria-se um vetor de termos desse texto e então se comparam as frequências dos termos deste vetor com as frequências dos termos dos vetores de cada

idioma, classificando o texto como pertencente ao idioma que teve maior similaridade, além de comparar os resultados obtidos a partir de diversas técnicas empregadas.

No capítulo seguinte são apresentados trabalhos relacionados com detecção de linguagem e classificação de textos, enquanto no Capítulo 3 são inseridos os modelos teóricos com os quais se desenvolveu este trabalho e no Capítulo 4 uma descrição de como funciona e como foi implementada a ferramenta. Os resultados obtidos são descritos no Capítulo 5. Finalmente, são apresentadas as conclusões do trabalho, assim como suas limitações, restrições e trabalhos futuros.

2 TRABALHOS CORRELATOS

Neste capítulo serão mostrados alguns trabalhos que foram base de pesquisa para o desenvolvimento da ferramenta para detecção de linguagem Jess.

A base inicial para a pesquisa foi a relação entre detecção de linguagem e o uso de n-gramas. N-gramas são segmentos de palavras de tamanhos variados, e foram escolhidos para o estudo por ser de simples compreensão e implementação, além de bem conhecidos na área de processamento de linguagens naturais. Outras abordagens incluem a verificação de palavras frequentes e letras ou caracteres especiais de um idioma, como uma abordagem não-computacional, e a medida de distância baseada em informação mútua, que consiste em comparar a compressão de um texto (i.e., uma espécie de assinatura que representa o texto) com a compressão (assinatura) de textos cujas linguagens sejam conhecidas (BENEDETTO, 2002). Cada um dos trabalhos possui sua peculiaridade, mas no geral concordam que a abordagem n-grama é a mais simples que possui melhor resultado.

TextCat é uma implementação do trabalho de Cavnar (1994), que visa categorizar textos em perfis de acordo com a linguagem ou assunto, suportando 69 linguagens diferentes, textos com alguns erros e situações onde o texto não se encaixa em nenhuma categoria ou mais de uma. Sua implementação é simples: as palavras do texto a ser categorizado são divididas em gramas, cujos tamanhos variam de 1 a 5, então é verificada a frequência de cada grama e criado um *ranking* em ordem decrescente de valor, comparadas as posições com as frequências de gramas dos perfis, que passaram pelo mesmo processo que o texto passou, e escolhido o perfil que possui a menor diferença de posições entre ele e o texto. Porém, durante os testes, a quantidade de gramas foi limitada para selecionar uma quantidade de gramas de maiores frequências em cada teste (variando entre 100, 200, 300 e 400), o que não acontece na ferramenta para detecção de linguagem Jess. Ainda, o TextCat não utiliza as palavras inteiras, visto que pode tornar o sistema mais sensível a erros de digitação, mesmo que ele possa suportar alguns.

TypeAny é uma ferramenta desenvolvida por Ehara (2008), que permite o usuário entrar com textos em diversas linguagens, sem precisar alterar manualmente a linguagem em um método de entrada (*Input Method Engine*), detectando automaticamente quando há alteração de idioma, inclusive alterando os caracteres para o alfabeto do respectivo idioma (por exemplo, do alfabeto romano para *kanjis* japoneses). Quando o usuário entra com uma palavra, o servidor identifica qual o idioma a que pertence e o retorna ao usuário, inclusive alterando o alfabeto, se for o caso. Nessa implementação, são utilizados *tokens*, e cada *token* é delimitado por espaços em branco. Assim, o processo reinicia para cada palavra. A escolha do idioma é uma estimativa que se dá através da maximização de uma equação envolvendo o conjunto de linguagens e o *token*, aplicadas ao Modelo Oculto de Markov. A detecção acontece

durante a digitação, o que faz com que o usuário precise verificar na janela do programa se o resultado está correto ou não.

O trabalho desenvolvido por Mariño (2006) é um sistema de tradução Inglês-Espanhol e Espanhol-Inglês. Trabalha com tuplas de unidades bilíngues, sendo uma tupla a frase mais curta que permite única segmentação do *corpus* bilíngue. A extração da tupla se dá por uma aproximação de acordo com uma equação de estatísticas entre fonte e objetivo, tem alinhamento muitos-para-muitos, de modo que se obtenha uma segmentação monotônica de pares de sentenças bilíngues, permitindo capturar o contexto e reordenar as informações dentro das estruturas. Foram implementados alguns refinamentos, para diminuir as limitações do sistema, como incorporar palavras, poda de tuplas de vocabulário, tuplas com fontes vazias de um lado, entre outras. O sistema de tradução tem melhores resultados quando traduz de espanhol para inglês e não o contrário. Um dos motivos é a concordância no espanhol, pois ao se traduzir do inglês para o espanhol surgem alguns erros devido às peculiaridades do espanhol. O sistema não suporta reordenamento de tuplas, portanto pode não suportar pares de linguagens mais distantes.

O sistema desenvolvido por Bergsma (2010) possui um bom desempenho por não depender de um domínio específico. As abordagens supervisionadas incluem duas classes de *features*: 1) N-gram (N-GM), que possui contagem dos gramas que aparecem no *corpus* da *web*; 2) identificação de léxicos especiais (LEX), não utilizando gramas, que simplesmente indica se uma *string* aparece ou não em uma posição da entrada. As tarefas analisadas foram reordenação de adjetivos prenominais, correção ortográfica sensível ao contexto, agrupamento de substantivos compostos e desambiguação de verbos de parte da fala (*part-of-speech*).

O método criado por Suzuki (2002) visa identificar o idioma, o alfabeto e a codificação de documentos de textos codificados por computador. O mecanismo utilizado foi uma versão diferenciada dos modelos n-gramas vistos anteriormente, chamada de *shift-codons*, que são subsequências dos bytes de um código de texto. A detecção se dá através de uma fórmula que utiliza as taxas em que os *shift-codons* aparecem no texto, cujo resultado é comparado a valores pré-determinados. O retorno será "*correct answer*" (resposta correta) ou "*unable to detect*" (incapaz de detectar). Um problema é que as amostras precisam de um tamanho mínimo, ou o método retornará "*unable to detect*".

Jess não será comparada aos trabalhos de Mariño, de Bergsma e de Suzuki, por não apresentarem resultados quanto à detecção de linguagem em si, visto que executam outros tipos de tarefas, como tradução no primeiro trabalho, ordenação de adjetivos e outras tarefas no segundo, e identificação de alfabeto e codificação juntamente com idioma no terceiro, dificultando uma possível análise de desempenho somente para a tarefa de identificação da linguagem.

3 FUNDAMENTAÇÃO TEÓRICA

A detecção de linguagem pode ser considerada um ramo da classificação de textos (PENG, 2003). Esta possui diversas abordagens, e algumas serão introduzidas na próxima seção.

3.1 Classificação de textos

A classificação de textos surgiu da necessidade de guardar um grande volume de documentos para leitura e posterior pesquisa. Em coleções de tamanho pequeno, até é possível verificar sequencialmente em busca de determinada referência, mas conforme aumenta a quantidade de textos, aumenta a dificuldade de encontrar a informação precisa sem nenhuma outra informação.

Para resolver tal problema, passou-se a catalogar os textos de acordo com alguma característica - assunto, autor, idioma (no caso deste trabalho), entre outras. Com a coleção separada por classes, pesquisar determinado assunto se torna mais fácil e rápido, o que torna o processo mais eficiente, visto que a quantidade de documentos que serão verificados é bem menor.

O problema da classificação de textos pode ser definido da seguinte forma (adaptado de (BAEZA-YATES, 2011)): dada uma coleção D de documentos e um conjunto $C = \{c_1,c_2, \ldots, c_N\}$ de N classes e seus respectivos rótulos, um classificador de textos é uma função F: D x C $\rightarrow$ {0,1}, tal que para cada documento d, existe um único c cujo par retorne o valor 1, ou seja, para cada par $[d_i,c_j]$, com $d_i \in D$ e $c_j \in C$, é atribuído o valor 1, se o documento i pertence à classe j, e 0, caso contrário, de maneira que cada documento pertença a apenas uma classe.

A maior parte dos algoritmos desenvolvidos para a classificação de textos são relacionados com a Aprendizagem de Máquina. A Aprendizagem de Máquina é um ramo da Inteligência Artificial que lida com o desenvolvimento de algoritmos capazes de aprender um padrão automaticamente, utilizando-o para tomar decisões inteligentes sobre dados de entrada. Existem 3 tipos de aprendizado que concentram os algoritmos de aprendizagem de máquina: 1) o Aprendizado Supervisionado, que exige treinamento a partir de dados de entrada fornecidos por um especialista; 2) Aprendizado Não-Supervisionado, onde o mais conhecido é o modelo de rede neural, no qual não há treinamento, dependendo da rede a inferência das relações; e 3) Aprendizado Semi-Supervisionado, que une um pequeno conjunto já classificado e um grande conjunto não classificado, para melhorar as predições (BAEZA-YATES, 2011).

Neste trabalho, será utilizado o Aprendizado Supervisionado, por possuir melhores resultados, que será explorado a seguir.

3.1.1 Aprendizado Supervisionado

Algoritmos são classificados como de aprendizado supervisionado quando utilizam informações providas por humanos (especialistas). Basicamente, tem-se um conjunto de classes e documentos pertencentes a estas classes - esses documentos foram classificados por especialistas, e formam o conjunto de treinamento, que servirá de aprendizado para a função de classificação. Com essa função, é possível classificar novos documentos entre as classes desse conjunto (BAEZA-YATES, 2011).

Um ponto importante na escolha do conjunto de treinamento é diversificá-lo. No caso deste trabalho, caso o conjunto de treinamento seja focado apenas em textos da área de informática, por exemplo, a função de classificação se ajustaria muito às amostras de treinamento, tendo problemas para identificar textos de áreas diferentes. Esse fenômeno é conhecido como Sobreajuste, e deve ser evitado (WITTEN, 2005). Pensando nisso, foi decidido que o conjunto de testes deve ter o vocabulário mais abrangente o possível, tomando para o conjunto de treinamento textos de diversas áreas (ver a seção 5.5, Treinamento e Testes).

A validação do classificador de textos é feita quando se executa a função de classificação para um conjunto de dados que não pertence ao treinamento, cujo resultado seja conhecido. Após a classificação, é comparada a classe atribuída pelo classificador e a classe definida pelo especialista e se ambas são a mesma, a função e o classificador funcionam como o esperado (BAEZA-YATES, 2011).

Os algoritmos para classificação de texto com aprendizado supervisionado podem ser representados pelas seguintes abordagens segundo (BAEZA-YATES, 2011): *decision trees*, *nearest neighbors*, *relevance feedback*, *naive Bayes*, *support vector machines* e *ensemble*.

Decision Tree, ou árvore de decisão, é um método que utiliza o conjunto de treinamento para construir regras de classificação, organizadas como caminhos de uma árvore. Esses caminhos então são usados para classificar amostras desconhecidas sem precisar testar todos os seus atributos. Cada nó intermediário da árvore indica um teste de atributo, sendo escolhido aquele atributo que trará uma maior qualidade de classificação. Os nós-folhas da árvore indicam os resultados da classificação. Uma vantagem é que a árvore de decisão possui uma estrutura simples: os dados estão nos caminhos, provendo uma fácil interpretação dos resultados do processo de classificação (BAEZA-YATES, 2011).

k-Nearest Neighbors, ou k-vizinhos mais próximos, é um classificador por demanda, também chamado de “preguiçoso”, porque o classificador gera um modelo a partir de uma entrada, e não previamente. A partir desta entrada, são encontrados *k* vizinhos próximos a ela, e a partir das classes desses vizinhos é que é definida a classe do documento de entrada. A vantagem do *k-nn* é que são considerados apenas os atributos do documento de entrada para a geração do modelo, ao invés de considerar todos os atributos da base, descartando atributos que não contribuem para a classificação do documento de entrada (YANG, 1999).

Relevance feedback, ou classificador de Rocchio, permite modificações na consulta do usuário baseadas em *feedbacks* do usuário, com o objetivo de melhor aproximar o resultado do esperado pelo usuário (MANNING, 2008). O conjunto de treinamento é considerado como *feedback*: termos que estão presente em um documento de uma classe *C* são considerados como de *feedbacks* positivos, enquanto termos que estão fora

deste documento para a mesma classe são considerados de *feedbacks* negativos. Os termos então são organizados como um vetor centroide, e novos documentos são classificados conforme a distância até o centroide (BAEZA-YATES, 2011).

Naive Bayes prevê uma probabilidade para cada par documento d_i classe c_j, indicando que o documento d_i pertença à classe c_j. Assim, com as probabilidades calculadas para todos os pares de [documento, classe], o documento é classificado como pertencente à classe de maior probabilidade. Um ponto interessante é que o modelo é considerado de alta independência das características: uma característica estar presente (ou ausente) não interfere na presença (ou ausência) de outra característica, o que traz variedade à classe (LEWIS, 1998).

Support vector machine, ou SVM, é um método de espaço vetorial para problemas de classificação binária. Os termos do documento formam um *index* que compõem o espaço vetorial, no qual os documentos são pontos (ou vetores). Com os documentos da base de dados representados como vetores, o objetivo é encontrar uma superfície de decisão (hiperplano) que melhor separe os documentos em duas classes distintas. O hiperplano é criado a partir do conjunto de treinamento, de maneira que todos os documentos de uma classe permaneçam de um lado do plano, enquanto os documentos da outra classe permanecem no outro (HOTHO, 2005).

Ensembles classifiers combinam os resultados de classificadores independentes com o objetivo de melhorar a qualidade do resultado (KUNCHEVA, 2003). Desde que os classificadores sejam precisos (resultados que sejam melhores que simples palpites) e diversos (produzem resultados diferentes para entradas diferentes), o método possui um bom resultado.

Esses são os seis principais tipos de algoritmos para classificação a partir de aprendizado supervisionado. A aplicação desenvolvida aqui utiliza uma abordagem semelhante a das SVM, já que a utilização de vetores é simples e a classificação conta com apenas duas categorias, adaptando-se melhor ao problema: os documentos do conjunto de treinamento são representados por vetores e estes se encontram no espaço vetorial, onde os vetores de documentos similares se encontram mais próximos. Para evitar comparar o documento de consulta com todos os documentos da base (o que seria custoso em termos de processamento, dependendo do tamanho da base de dados), as classes são representadas como vetores com todos os termos dos documentos que nelas estão contidos, e a função que classifica os documentos é uma função que calcula a distância entre a consulta e os vetores que representam as classes (ver seção 4.2, Vetores de Linguagens). Uma explicação mais detalhada será dada nas seções a seguir.

3.2 Recuperação de Informação

Recuperação de Informação (RI) é a área da Informática que lida com armazenamento de documentos e a recuperação automática de informações relacionadas a eles. RI é utilizada no problema da detecção de linguagem de modo que se aprenda quais os idiomas de textos da nossa base de dados e verificando qual documento possui conteúdo mais similar ao da consulta, determinando então que ambos os textos estão escritos na mesma linguagem (MANNING, 2008).

Os documentos são compostos de termos de indexação, que são palavras que servem para descrever determinado documento. Cabe ao modelo de RI determinar quais palavras serão termos de indexação, assim como qual será a representação dos

documentos e como será feita a comparação entre eles. São considerados os Modelos Clássicos da RI: o Booleano, o Vetorial e o Probabilístico.

No modelo Booleano, trabalha-se com a teoria dos conjuntos para representar os termos - cada conjunto representa os documentos que contêm determinado termo - e as consultas são feitas utilizando os operadores booleanos E, OU e NÃO. Por ser muito simples, o modelo booleano não permite uma maior expressividade na consulta, além de o conjunto de termos ter um tamanho gigantesco, dependendo do tamanho da base.

No modelo vetorial, os documentos são considerados vetores de termos, e utilizam-se operadores vetoriais para comparação de documentos. É um dos modelos mais utilizados em RI, e será aprofundado ao longo do trabalho por ser o escolhido.

O modelo Probabilístico lida com a relevância de documentos. Considerando que alguns documentos são determinados relevantes e outros não, pode-se estimar a probabilidade de um termo aparecer em um documento relevante.

3.2.1 Modelo Vetorial

Após estudar os três modelos clássicos de recuperação de informação, foi escolhido o modelo vetorial para abordar o problema, visto que é uma das abordagens mais utilizadas com bons resultados nesta área, além da simplicidade de implementação.

O modelo vetorial representa os objetos em geral como um vetor de identificadores, no caso deste modelo, os objetos são os documentos e as consultas, representados como vetores de termos. Cada elemento do vetor representa um termo, resultando que a dimensão do vetor é a quantidade de termos distintos presentes no documento. Como os documentos são representados com vetores, é possível utilizar operações vetoriais para compará-los, o que facilita muito o trabalho (MANNING, 2008).

Dado um documento, é criado um vetor de termos, que possui um vetor de frequências associado (Figura 3.1). Assim, os valores de frequência são todos positivos, visto que se um termo não está presente no documento, ele não estará no vetor. Outra implicação é que termos de um vetor podem não estar presentes em outro, podendo ser ignorados no cálculo de comparação. No vetor, todo o termo é passado para letras minúsculas, além de a acentuação ser ignorada.

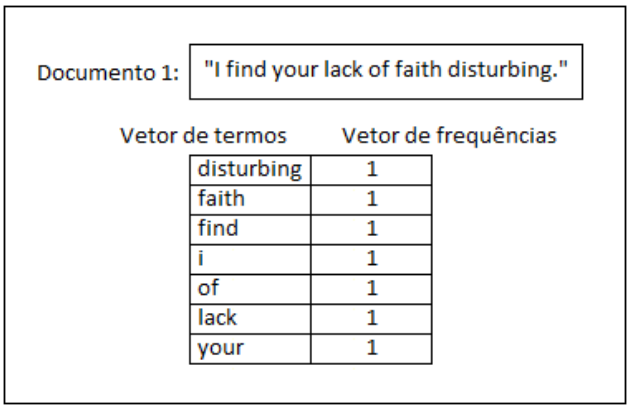

Figura 3.1: Exemplo de texto de entrada e respectivos vetores de termos e de frequências associados.

Fonte: elaborada pela autora.

Com as quantidades dos termos, pode-se utilizar esquemas de ponderação (explorados na seção 4.3.2) para determinar se algum termo é mais interessante que outro na comparação.

Os vetores dos documentos estão localizados no espaço. Com o vetor de consulta e os perfis normalizados, são calculadas as distâncias entre o vetor consulta e os documentos através do cosseno do ângulo entre eles - o tamanho do vetor é desconsiderado. Vetores mais próximos possuem maior similaridade (Figura 3.2), logo, o idioma escolhido é aquele que está associado ao documento cujo vetor esteja mais próximo do vetor da consulta, sendo considerado o documento mais similar.

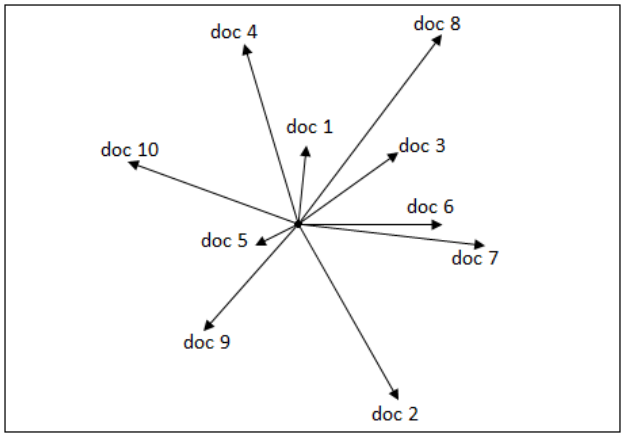

Figura 3.2: Vetores de termos no espaço – doc4 é o mais similar a doc 1 por ser o mais próximo.

Fonte: elaborada pela autora.

4 PROPOSTA PARA DETECÇÃO DE LINGUAGEM

Aqui serão apresentados os detalhes de modelagem e implementação da Jess, ferramenta para detecção de linguagem proposta. Jess classifica textos entre os idiomas comparando os vetores que representam cada idioma com o vetor que representa o texto de entrada através de uma função de similaridade, após uma aplicação de pesos nas frequências dos termos nos vetores. Inicialmente, são abordados os idiomas escolhidos e como serão tratados, os esquemas de ponderação utilizados e como é feito o cálculo de similaridade que classifica o texto.

4.1 Idiomas

Os idiomas escolhidos para o processo de detecção de linguagem são o inglês e o português – português por ser nossa linguagem natural, inglês por ser o idioma mais comum na internet. Como não é feita nenhuma filtragem específica para esses idiomas, pode-se adicionar idiomas em um trabalho futuro, bastando para isso alguns ajustes e treinar a função com uma base de dados do novo idioma.

Os idiomas inglês e português possuem padrões consideravelmente distintos. No inglês, por exemplo, a letra mais utilizada é o "e", enquanto no português é o "a". Com exceção de nomes próprios e palavras estrangeiras aportuguesadas, não se vê o dígrafo "th" na língua portuguesa, enquanto na língua inglesa é o dígrafo mais comum.

Em construções textuais, geralmente se encontram termos de ligação de orações (preposições, conjunções, artigos, entre outras classes gramaticais) que se tornam comuns a muitos documentos, como por exemplo, "*o"*, *"de"*, "*e"*, "*para"*, "*em"* para o caso do português, e "*the"*, "*of"*, "*and"*, *"to"*, "*in"* para o caso do inglês. Essas são palavras que geralmente não acrescentam muita informação ao documento, não sendo consideradas palavras-chaves. Esse tipo de termo costuma ser chamado de *stop-words*, e geralmente é excluído da análise. Como o objetivo do trabalho é identificar o idioma do texto e não seu conteúdo, e esse texto pode ser muito curto, carecendo de informações mais relevantes, os *stop-words* permanecem na análise, e muitas vezes são eles que definem o idioma do texto - quando, por exemplo, o texto é um *post* de blog de duas linhas. Quanto mais informação o texto contiver, melhor é o resultado.

Com esse tipo de análise, verificando os termos e gramas mais comuns em cada linguagem, a detecção da linguagem já atinge bons resultados, ao menos para textos puros, isto é, totalmente escritos em determinado idioma.

Português e inglês são as duas classes trabalhadas na Jess, e cada uma é representada por um vetor de linguagem, detalhado a seguir.

4.2 Vetores de Linguagens

Comparar o texto de entrada com cada documento da base de dados pode se tornar algo ineficiente, dependendo do tamanho da base. Por essa razão, existem técnicas que propõem uma comparação com apenas alguns documentos, como, por exemplo, o *k-nn* (mencionado na seção 3.1.1), que compara a consulta com os k documentos mais próximos no espaço vetorial.

Mas, dependendo de quantas classes estão sendo consideradas, comparar com apenas alguns documentos pode se tornar ineficaz, por não haver dados suficientes para garantir um resultado. Para que se chegue a um meio-termo, optou-se por utilizar representações de cada classe, que são os vetores de linguagens. Para cada linguagem, durante o treinamento, os termos de cada documento são inseridos no vetor de linguagem, contando suas frequências e em quantos documentos o termo em questão aparece (dado que será utilizado no esquema de ponderação *tf-idf*, ver seção 4.3.2). Na prática, é como se todos os documentos de um idioma fossem compilados em um único documento e a partir desses documentos cria-se o vetor de termos, com o vetor de frequências associadas.

Comparar a consulta com o vetor de linguagem é como comparar a consulta com todos os documentos da classe simultaneamente. Um problema que poderia surgir seria o tamanho desse vetor tornar a comparação um processo pesado, porém, a função de comparação foi feita de maneira a comparar os vetores de maneira a utilizar o menor dos dois como base, ou seja, os termos do menor vetor são procurados no maior, o que diminui o tempo de processamento. Assim, tem-se o melhor dos dois: compara-se a consulta com apenas um vetor de cada classe e com todos os termos da base.

4.3 Técnicas utilizadas

Aqui serão introduzidas as técnicas que foram implementadas na ferramenta para detecção de linguagem em texto Jess. Inicialmente, a normalização dos vetores será apresentada e exemplificada, para então entrar no detalhamento dos esquemas de ponderação, que são a chave do programa, no cálculo de similaridade e por fim, explorar os n-gramas.

4.3.1 Normalização

Nem todos os textos possuem o mesmo tamanho: alguns são bem longos, enquanto outros são extremamente curtos. Simplesmente comparar as frequências dos termos independente do tamanho do texto não é eficaz, visto que ao se comparar um texto de 20 linhas, por exemplo, com outro de uma linha poderia gerar um resultado pouco similar, mesmo que todas as palavras do segundo texto estejam contidas no primeiro.

Para evitar uma tendência para documentos mais longos, os vetores de frequências são normalizados para que seu comprimento seja igual a 1. Suponha o seguinte texto de entrada (Figura 4.1) e o vetor de frequências associado a ele (Figura 4.2):

"Where other men blindly follow the truth, remember...
Nothing is true.
When other men are limited, by morality or law, remember...
Everything is permitted.
We work in the dark to serve the light.
We are Assassins."
- Assassin's Creed 2

Figura 4.1: Exemplo de texto de entrada do programa.

Fonte: elaborada pela autora.

Termo	Frequência
2	1
are	2
assassins	1
assassin's	1
blindly	1
by	1
creed	1
dark	1
everything	1
follow	1
in	1
is	2
law	1
light	1
limited	1
men	2

Termo	Frequência
morality	1
nothing	1
or	1
other	2
permitted	1
remember	2
serve	1
the	3
to	1
true	1
truth	1
we	2
when	1
where	1
work	1

Figura 4.2: Vetor de frequências associado ao documento de entrada.

Fonte: elaborada pela autora.

O comprimento do vetor v é tomado pela raiz quadrada do somatório dos quadrados das frequências, como indica a fórmula a seguir:

$$|v| = \sqrt[2]{\sum_{i=1}^{n} f_i^2}$$

onde v é o vetor a ser normalizado, n é a quantidade de componentes do vetor (neste caso os termos) e f é o valor de frequência. Logo, para normalizar, basta dividir os valores de frequência de cada termo pelo comprimento do vetor, calculado a seguir:

$$|v| = \sqrt[2]{\begin{matrix}(1^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 2^2 + 1^2 + 1^2 + 1^2 + 2^2 \\ +1^2 + 1^2 + 1^2 + 2^2 + 1^2 + 2^2 + 1^2 + 3^2 + 1^2 + 1^2 + 1^2 + 2^2 + 1^2 + 1^2 + 1^2)\end{matrix}}$$

$$|v| = 7{,}549834$$

Termo	Frequência
2	0,132453
are	0,264906
assassins	0,132453
assassin's	0,132453
blindly	0,132453
by	0,132453
creed	0,132453
dark	0,132453
everything	0,132453
follow	0,132453
in	0,132453
is	0,264906
law	0,132453
light	0,132453
limited	0,132453
men	0,264906

Termo	Frequência
morality	0,132453
nothing	0,132453
or	0,132453
other	0,264906
permitted	0,132453
remember	0,264906
serve	0,132453
the	0,397360
to	0,132453
true	0,132453
truth	0,132453
we	0,264906
when	0,132453
where	0,132453
work	0,132453

Figura 4.3: Vetor de frequências normalizado.

Fonte: elaborada pela autora.

Ao se aplicar a fórmula do comprimento do vetor aos novos valores, o resultado deve chegar a 1, ou algo muito próximo, devido a precisão. Com os vetores normalizados, não só é evitada a tendência a documentos mais longos, como também é simplificado o cálculo de comparação dos vetores através do cosseno do ângulo. A normalização ocorre após a aplicação dos pesos, explicada a seguir.

4.3.2 Esquemas de ponderação

Esquemas de ponderação (*weighting schemes*) servem para atribuir pesos variados aos termos, de acordo com a abordagem utilizada (MANNING, 2008). Um termo que aparece muitas vezes pode ser considerado de duas maneiras: a) como uma palavra-chave que define o documento ou b) como uma palavra que aparece tantas vezes e em tantos documentos que sua presença não serve para indicativo, sendo melhor ignorá-la. Considerações semelhantes ocorrem com palavras que aparecem em poucos documentos ou que tenham baixa frequência. Com esquemas de ponderação, pode-se escolher a maneira como tratar esse tipo de termos.

Primeiro, a mais simples: *tf* (*term-frequency*). Basicamente, a frequência absoluta dos termos, uma contagem de quantas vezes determinado termo aparece em um documento (LAN, 2005) - geralmente essa contagem é normalizada, para evitar uma pré-disposição aos documentos mais longos (descrita na seção anterior). Considerando nosso problema, a detecção de linguagem, é interessante utilizar a *tf* em gramas, já que contar quantas vezes aparece uma mesma palavras em textos curtos pode não ser muito relevante, mas contar quantas vezes aparece um dígrafo em especial pode ser bem eficaz. A *tf* será aplicada a gramas de tamanhos 2, 3 e 4. A partir da *tf*, surgem outros esquemas.

Um dos esquemas mais conhecidos e utilizados, o *tf-idf* (*term frequency – inverse document frequency*) é um complemento ao clássico *tf*: enquanto este verifica o quanto

um termo aparece em um documento, o *tf-idf* verifica quão raro é um termo para o conjunto total de documentos (LAN, 2005). O *idf* é calculado dividindo a quantidade total de documentos (D) pela quantidade de documentos (d) em que o termo (t) aparece, e então tomando o logaritmo; para chegar ao *tf-idf*, basta multiplicar pelo valor de *tf*, conforme mostrado abaixo:

$$idf(t, D) = \log_{10}(D/d), onde\ d \in D\ e\ t \in d$$

$$tf - idf(t, d, D) = tf(t, d) * idf(t, D)$$

O *tf-idf* será utilizado com as palavras inteiras, visto que pode reduzir a quantidade de documentos a serem comparados, em uma espécie de busca por palavras-chave.

O último esquema também é bem simples: o booleano. Basicamente, atribui valor 1 se o termo está presente no documento *d* e 0 caso não esteja (LAN, 2005). Como é difícil que não haja a ocorrência de determinado grama pelo menos uma vez, esta abordagem será utilizada também com palavras inteiras.

4.3.3 Similaridade

No modelo vetorial, os documentos são vetores que se localizam no espaço. A similaridade entre dois vetores será calculada através de *Cosine Similarity*, que calcula a semelhança de vetores com base na distância entre eles, através do cosseno do ângulo entre eles (HUANG, 2008). O vetor mais próximo do vetor do texto de entrada é aquele cujo conteúdo é o mais similar, e pode-se concluir que ambos pertencem à mesma classe, ou seja, linguagem. Conforme explicado anteriormente, ao invés de se comparar o vetor consulta com todos os documentos da base, ele será comparado com os vetores de linguagem.

O cosseno do ângulo entre dois vetores pode ser calculado através do produto interno entre eles, divido pelo produto dos comprimentos dos vetores, como mostra a fórmula abaixo:

$$sim(d, q) = cos(\vec{d}, \vec{q}) = \frac{\vec{d} * \vec{q}}{|\vec{d}| * |\vec{q}|}$$

onde *d* é o vetor de um documento da base de dados e *q* é o vetor de um documento de entrada. Como os vetores estão normalizados, seus comprimentos são iguais a 1 e o denominador será 1, tornando o cálculo da similaridade simples ao calcular apenas o produto interno dos vetores.

Suponha *q* uma consulta e doc1, doc2 e doc3 documentos da base de dados, representados pelos termos *a*, *b* e *c*, cujas frequências já estão normalizadas (note que na consulta *q* não existe o termo b):

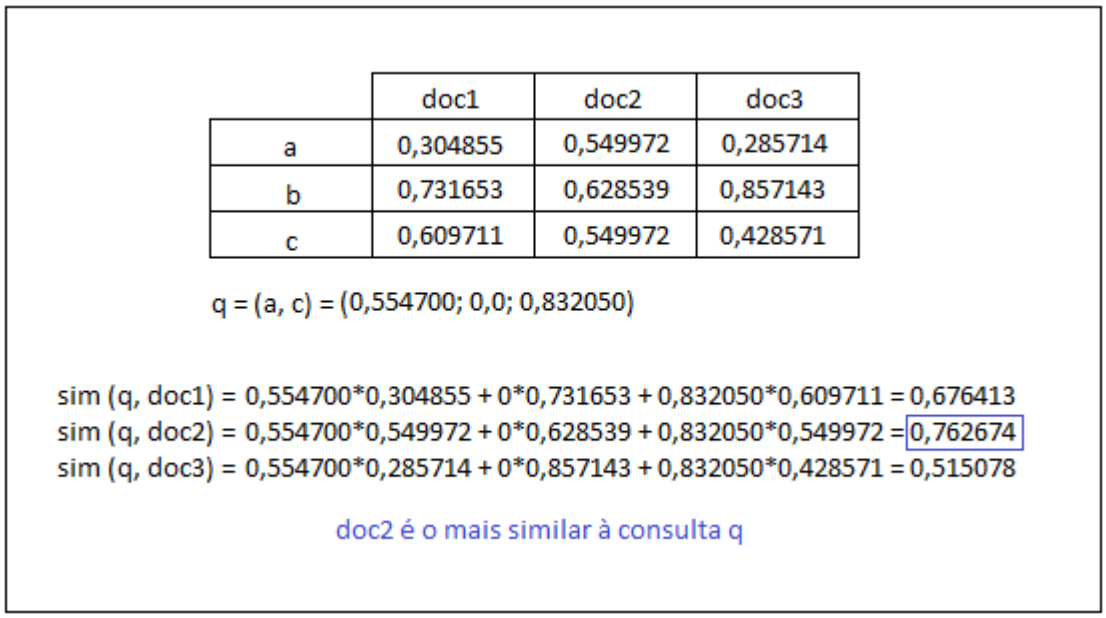

Figura 4.4: Exemplo de cálculo de similaridade entre uma consulta q e três documentos da base - doc1, doc2 e doc3.

Fonte: elaborada pela autora.

No caso deste trabalho, haverá a consulta e os dois vetores de linguagem. Dependendo da precisão desejada para os resultados, pode-se exigir um valor mínimo de similaridade para que a consulta seja classificada como pertencente à determinada linguagem, e caso nenhum dos dois idiomas chegue nesse mínimo, é retornado "Idioma não determinado". Para este trabalho, como português e inglês são idiomas notavelmente diferentes, não é exigido um valor mínimo para similaridade.

4.3.4 N-gramas

Em linguística, n-gramas são sequências contínuas de caracteres de uma sequência de texto, como por exemplo, uma letra, uma sílaba, uma palavra inteira, entre outras (CAVNAR, 1994). São largamente utilizados no processamento de linguagens devido a suas simplicidades e escalabilidades, visto que utilizar gramas de tamanho 2 e gramas de tamanho 4 passam pelo mesmo processo.

Uma das características que se nota ao utilizar n-gramas é a identificação de palavras derivadas, o que ajuda no processo de identificação de similaridade. Ao se considerar uma palavra inteira, um documento em que apareçam as palavras "flor", "floricultura", "florescer" e "floreira", a ocorrência de cada palavra pode ser baixa, mas ao avaliar a ocorrência do grama de tamanho 4, "flor", a frequência se torna mais alta e uma ocorrência relevante para o cálculo de similaridade.

Neste trabalho, serão utilizados n-gramas de tamanhos 2, 3 e 4, por serem os tamanhos cujos resultados são mais interessantes devido ao tamanho dos termos e às peculiaridades dos idiomas, como por exemplo, dígrafos. Gramas de tamanho 5 são prejudicados na contagem final de gramas, visto que muitas palavras, de ligação, por exemplo, têm tamanho menor que 5 letras, e gramas de tamanho 1 são muito simples, permitindo apenas 26 gramas com altas frequências, e devido ao grama ser apenas uma letra, o resultado seria pior que o de gramas de tamanho 2, puxando o resultado para o errado ainda mais.

Os gramas são extraídos de cada termo, a partir do vetor de termos. Caso o texto fosse considerado uma cadeia de caracteres única, haveria pontuação e espaços em brancos para se tratar. Extraindo os gramas do vetor de termos, não só não é necessário tratar desses problemas, como também se evita o problema de um grama desinteressante que não contribui para a solução do problema, por exemplo, um grama formado pelo final e pelo início de outra palavra, cuja ocorrência seja extremamente baixa ou nula para os idiomas. Outra vantagem é que a frequência dos gramas é mais facilmente calculada, bastando multiplicar a frequência do grama no termo pela frequência do próprio termo.

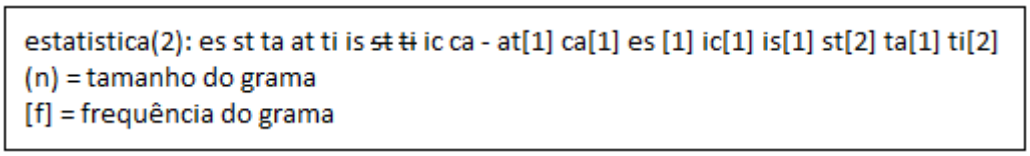

Figura 4.5: Exemplo de extração de gramas de tamanho 2 de um termo e suas frequências.

Fonte: elaborada pela autora.

O modo de criação do vetor de gramas é o mesmo do vetor de termos com um adicional: para cada grama obtido, verifica-se se o mesmo se encontra no vetor e se for o caso, aumenta a quantidade em (frequência do termo x frequência do grama), do contrário, insere no vetor com a quantidade do grama no termo analisado, ou seja, se um termo tem uma frequência 4 e um grama deste termo tem frequência 2 no mesmo termo, a frequência do grama será aumentada em 8 no vetor de gramas.

Os cálculos de similaridade utilizando gramas não passam por processo de aplicação de pesos, pois o esquema de pesos utilizado para eles é o *tf*, bastando extrair os gramas do documento de entrada e dos vetores de linguagem e normalizar os vetores para o cálculo.

5 FERRAMENTA E EXPERIMENTO

Aqui serão introduzidos primeiramente os softwares utilizados para o desenvolvimento da ferramenta, para então apresentar a estrutura da Jess e um pseudo algoritmo do programa. A seguir, são abordados os testes e os resultados obtidos com eles.

5.1 Estrutura

O objetivo do trabalho é comparar os resultados obtidos para detecção de linguagem utilizando métricas diferentes: ponderação *tf-idf*, booleana e *tf* para n-gramas de tamanhos 2, 3 e 4.

Para isso, desenvolveu-se uma ferramenta que lê um texto a partir de uma interface gráfica e então o adiciona como um documento ao *index* da API Lucene, que o converte para o formato de um vetor com todos os termos do texto de entrada, retirando espaços, pontuação, acentuação e passando os termos para minúsculas. Com o documento no *index* da Lucene, é fácil extrair as informações necessárias para a análise, como a frequência dos termos, quantidade de termos do documento, entre outras, a partir de funções específicas da Lucene.

A partir da fase de treinamento, foram criados os vetores de linguagem “ptbr” e “eng”, representando os idiomas português e inglês, respectivamente. Cada vetor é tratado como um documento no *index* do Lucene, e contém todos os termos dos documentos classificados como de seu respectivo idioma, além das informações de quantos documentos estão presentes no vetor, a frequência absoluta de cada termo na categoria e a quantidade de documentos em que cada termo aparece.

O vetor de termos e os vetores de linguagem então são passados para estruturas do tipo *Bag of Words*, que irão passar pelos processos de ponderação. *Bag of Words* é uma das representações mais simples utilizada na área de RI, e consiste em definir que um texto é uma coleção não ordenada de palavras. Para cada esquema, a partir de um *Bag of Words*, é retornado um novo vetor de frequências, de acordo com cada ponderação (SCOTT, 1999).

Segue-se então, para cada esquema de ponderação, uma comparação entre o vetor de consulta e o vetor ptbr, e outra comparação entre o vetor de consulta e o vetor eng, associando o vetor de consulta à mesma categoria do idioma que atingiu maior similaridade.

Ao final, seis resultados são retornados ao usuário: os idiomas selecionados usando ponderação *tf-idf*, booleana, *tf* para gramas de tamanho 2, *tf* para gramas de tamanho 3, *tf* para gramas de tamanho 4 e *tf* em geral (considerando todas as três variações).

5.2 Algoritmo

Nesta seção, serão apresentados um esquema de entrada e saída do programa (Figura 5.1) e um esboço do algoritmo utilizado na implementação do detector de linguagem.

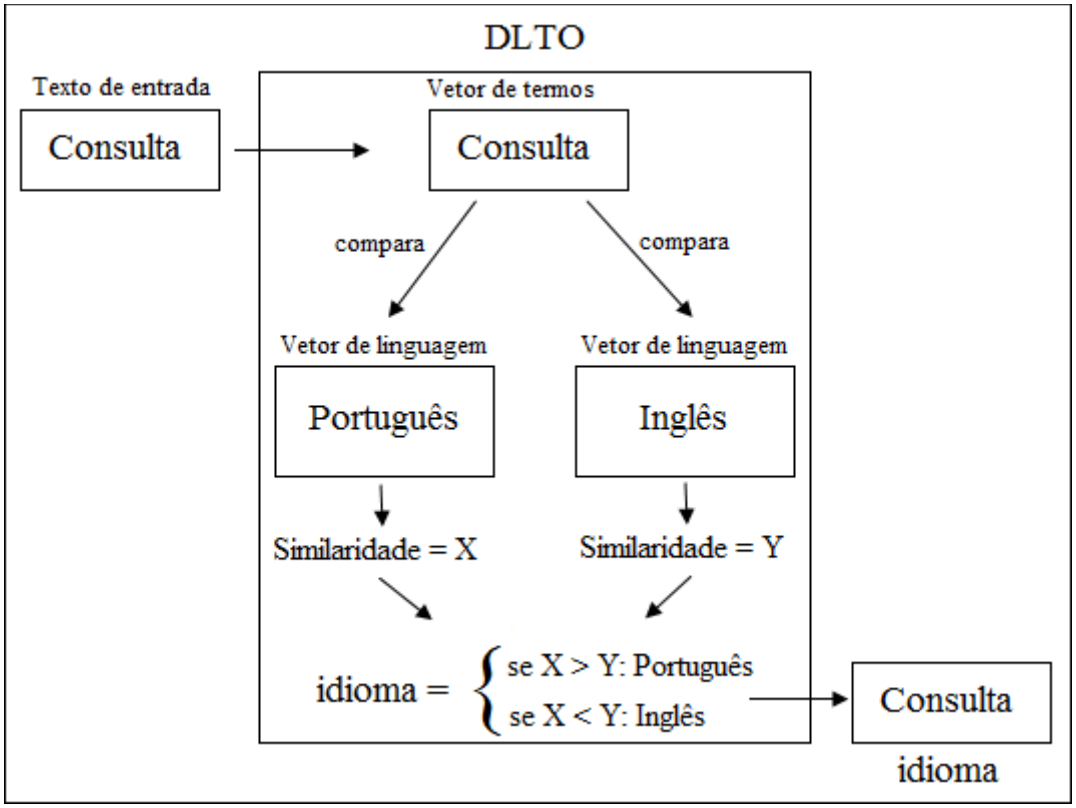

Figura 5.1: Esquema de entrada e saída do programa.

Fonte: elaborada pela autora.

1. São carregados os vetores de linguagem representando os idiomas português e inglês, que serão utilizados para a comparação, passados para *Bag of Words* e aplicados os pesos de cada métrica - logo, são gerados 5 *Bag of Words* para cada idioma.

2. A entrada do programa é um texto (inserido manualmente ou carregado pelo programa), que será tratado por consulta.

3. A consulta é adicionada diretamente a um documento do Lucene (para extração dos termos de forma mais simples).

4. Esse documento é transformado em um vetor de termos, com um vetor de frequências dos termos associado.

5. O vetor de termos é transformado em um *Bag of Words*.

6. Para cada esquema de ponderação:

6.1. É aplicado o peso do esquema na *Bag of Words* da consulta.

6.2. É aplicada a função de similaridade para a consulta com o vetor representando português e depois aplicada com o vetor representando inglês.

6.3. É escolhido o idioma do vetor de linguagem que possuir valor de similaridade mais alto.

6.4. Para o esquema de ponderação *term-frequency*, que envolve a utilização de gramas, o trecho é executado três vezes: para gramas de tamanhos 2, 3 e 4, e ao final é tirada uma média dos resultados obtidos.

7. É retornado ao usuário os resultados das comparações: o idioma selecionado no caso do peso *tf-idf*, do booleano, do *tf* para gramas de tamanho 2, do *tf* para gramas de tamanho 3, do *tf* para gramas de tamanho 4 e do *tf* em geral.

5.3 Lucene

Para a implementação da ferramenta na linguagem Java, foi utilizada a biblioteca Lucene para extração dos termos e gramas dos documentos.

Apache Lucene, ou apenas Lucene, é uma API de recuperação de informação *open source* para indexação e busca de documentos. Lucene não contém um *crawler* ou um *parser*, ficando a critério do usuário adicionar as funcionalidades. São aceitas quaisquer entradas do tipo textual (arquivos textos, html, PDF, entre outros).

Para a implementação da Jess, o programa carrega os arquivos de notícias extraídos da coleção da Reuters e do *site* Wikinotícias, que são adicionados como documentos ao *index* da Lucene. A partir desses documentos são criados os vetores de termos e calculadas as frequências absolutas, que são o núcleo do programa.

Foi utilizada a classe *StandardAnalyzer*, que é a implementação de *Analyzer* mais utilizada para textos na língua inglesa e línguas baseadas em latim (APACHE, 2012). A *StandardAnalyzer* cria um *Tokenizer*, que irá extrair os termos e suas frequências de um texto para análise.

Lucene automaticamente elimina as *stop-words*, e permite que o usuário defina seu próprio conjunto de *stop-words* como parâmetro para o *StandardAnalyzer*. Como o modelo adotado para a implementação utiliza essas palavras, foi passado como parâmetro um conjunto vazio, que indica que nenhuma palavra deve ser eliminada.

Outra classe utilizada durante o desenvolvimento foi a *Index*, que permite a análise de vários textos em conjunto, para busca e posterior análise de similaridade entre os textos. Posteriormente, a análise de similaridade foi feita entre a consulta e os perfis de linguagem ao invés de entre a consulta e os textos da base, o que fez com que a *Index* seja utilizada apenas para a utilização do *StandardAnalyzer*.

5.4 Interface do programa

Para a execução da Jess com apenas um texto, é possível utilizar a interface do programa, que permite a inserção manual de um texto para identificação do idioma. Por enquanto, a interface apenas permite a inserção manual de um texto e conta com um botão para execução, mas em uma próxima revisão espera-se tornar possível a execução

da ferramenta para uma página *web*, bastando inserir o endereço da página em um campo apropriado da ferramenta. Após a execução, o programa retorna os resultados referentes a cada métrica no campo “Resultados”, contando com o idioma selecionado, os valores de similaridade para ambos os idiomas, e o tempo de processamento para cada métrica. A seguir, exemplos de execução para dois textos diferentes, um em inglês (Figura 5.2) e um em português (Figura 5.3):

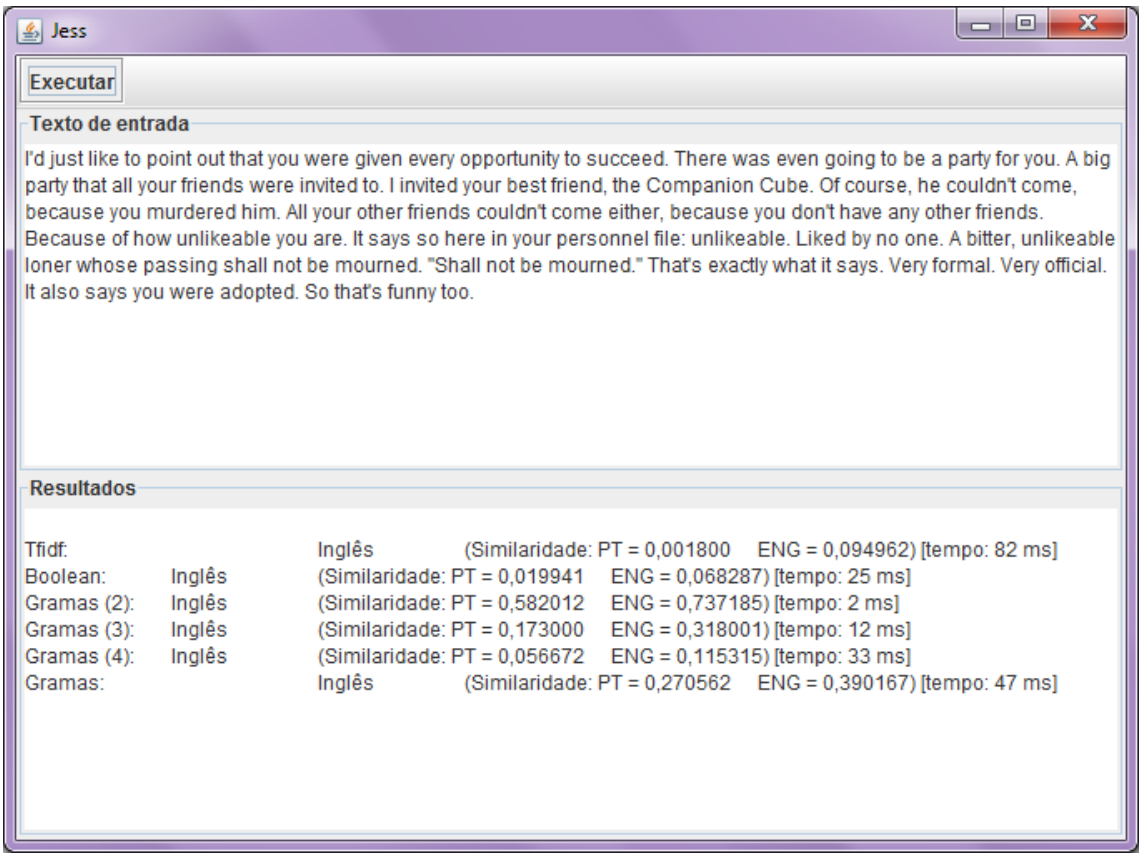

Figura 5.2: Tela de execução do programa para um texto em inglês e o resultado retornado.

Fonte: elaborada pela autora.

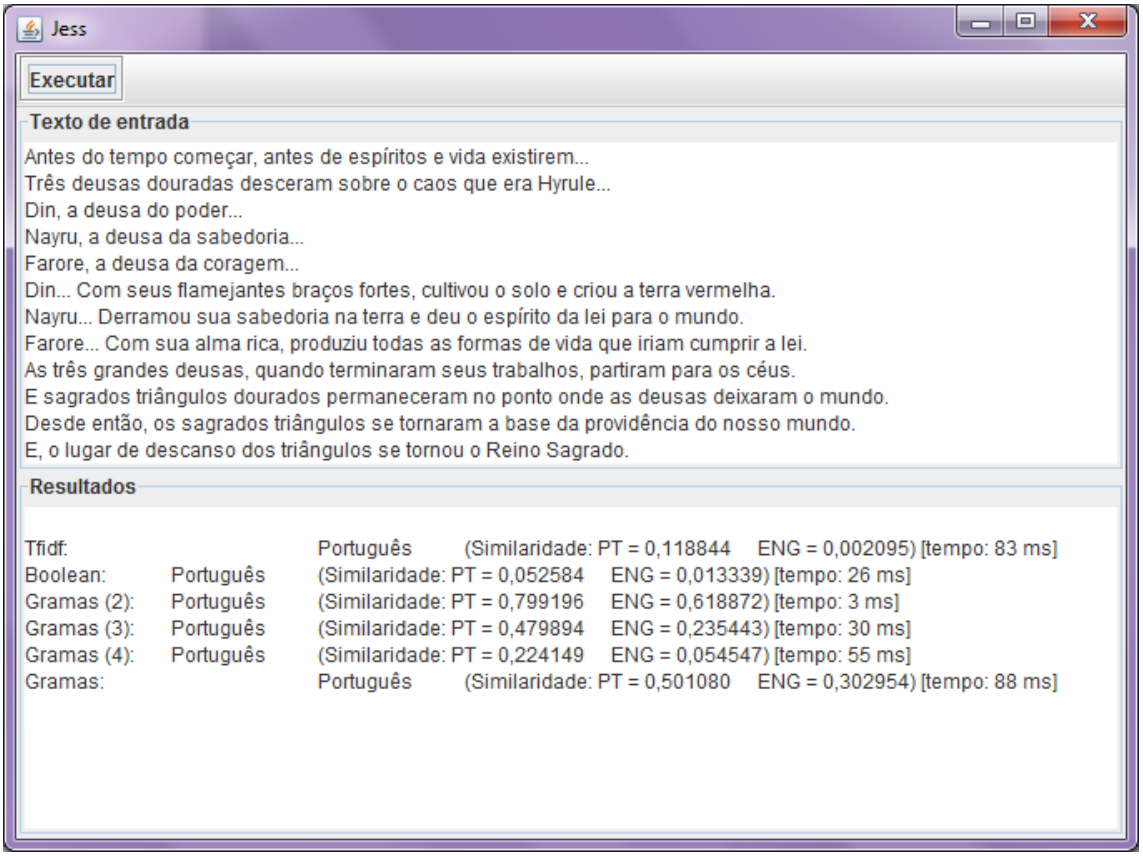

Figura 5.3: Tela de execução do programa para um texto em português e o resultado retornado.

Fonte: elaborada pela autora.

5.5 Treinamento e Testes

A fase de treinamento é obrigatória para algoritmos do tipo Aprendizado Supervisionado, pois é a partir dele que a função vai aprender a identificar os padrões para classificação. O treinamento da Jess consiste em adicionar documentos cujos idiomas já sejam conhecidos nas categorias português e inglês, e esses documentos irão compor os vetores de linguagens.

Para o idioma inglês, foi utilizada a coleção de textos *Reuters-21578, distribution 1.0* (LEWIS, 2012), uma coleção de notícias da agência londrina que dá nome à coleção, e que é dividida em 5 categorias: *exchanges*, *orgs*, *people*, *places* e *topics*. Cada documento representa uma notícia. Para o idioma português, foram selecionadas notícias do *site* Wikinotícias de 5 categorias (para manter a coerência com a outra base): Ciência e tecnologia, Economia e negócios, Cultura e entretenimento, Saúde e Mundo. Cada documento representa uma notícia. A escolha de utilizar mais de uma categoria se deve ao fato de prover uma maior abrangência aos vocabulários ao escolher temas variados.

Para cada idioma, foram coletadas 2000 notícias, que são divididas em dois conjuntos: conjunto de treinamento (d_0), para análise, e conjunto de testes (d_1), para validação. Para utilizar a validação cruzada (*cross-validation*) em sua variação mais simples, a *2-fold cross-validation* (KOHAVI, 1995), para cada idioma, primeiro treina-se com d_0 e então se aplica os testes com d_1, e então treina-se com d_1, seguido de testes com d_0. Como os conjuntos serão invertidos, é usual que ambos tenham o mesmo

tamanho, o que nos leva a que ambos os conjuntos tenham um tamanho de 1000 documentos para cada idioma na fase de treinamento e os 2000 documentos restantes para testes.

Cada rodada de teste é dividida em parte AB e parte BA. São criados 4 conjuntos: PtbrA e PtbrB, cada um com 1000 documentos aleatórios em português, e EngA e EngB, cada um com 1000 documentos aleatórios em inglês. Primeiro, para a parte AB, utiliza-se os conjuntos B para treinamento, testando com os conjuntos A; para a parte BA, inverte-se os conjuntos, treinando-se com os conjuntos A e testando com os conjuntos B. Ao final de cada parte um relatório é gerado (como mostrado na Figura 5.4), contendo o caminho do arquivo testado, a marca [ERRO] caso a linguagem determinada por algum dos métodos difira da linguagem do documento em questão, o idioma determinado por cada métrica junto com os valores de similaridade para cada idioma e o tempo que cada métrica levou para processar, além de um resumo no final do relatório com a quantidade de acertos de cada métrica e uma média do tempo de processamento de cada uma. Para a métrica *tf*, é tirada a média dos resultados obtidos com cada uma das variações de tamanhos de gramas, e o tempo é considerado o total que levou para processar as três variações.

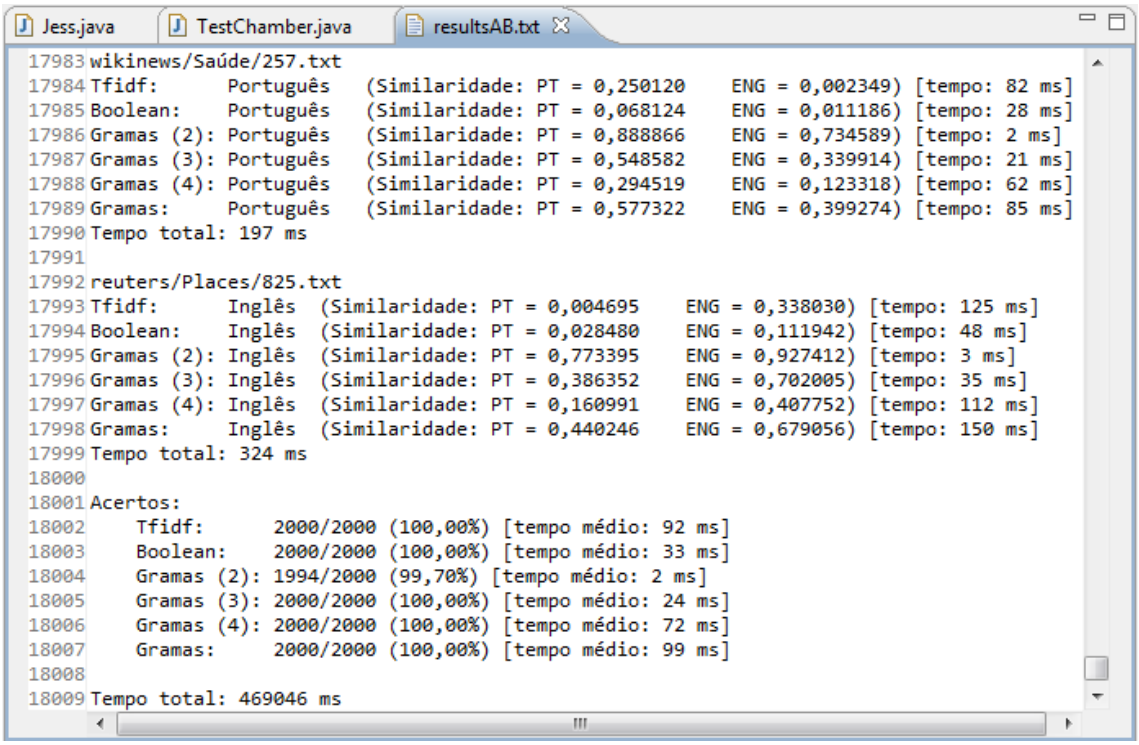

Figura 5.4: Exemplo de relatório gerado pelo programa.

Fonte: elaborada pela autora.

Jess trabalhou com textos a princípio puros (nas notícias é comum encontrar nomes estrangeiros) e cujos tamanhos variam entre 1KB e 9KB. Quanto maior o texto, mais informação é extraída e melhores são os resultados. O processamento das duas partes do teste levou cerca de 20 minutos, e os resultados se encontram na próxima seção.

O objetivo da ferramenta é a detecção de linguagem em textos como *posts* de blog ou de jornais online, e uma tentativa de detectar o idioma de Tweets – a tentativa se deve ao tamanho do Tweet, visto que o tipo de escrita pode variar nessa rede justamente pela quantidade de texto permitida. Para tanto, verificou-se os limites de postagens de

ambos. O Twitter tem um limite de 140 caracteres por mensagem, enquanto o Blogspot não limita o tamanho do *post*, embora haja um limite de 1MB na página do blog, limitando a quantidade de *posts*. Não foi encontrado um limite mínimo para notícias de jornais.

Para os testes simulando *posts* do Blogspot e notícias de jornais, utilizaram-se as notícias coletadas em seus tamanhos originais, que vão de alguns bytes a 9KB de texto. Já os testes simulando o Twitter, optou-se por truncar os textos em 140 caracteres, cuidando para que, caso na posição limite se encontre uma palavra, o texto é truncado antes do início da mesma, para evitar análises equivocadas nas métricas de gramas.

5.6 Resultados

Aqui são apresentados os resultados dos testes descritos na seção anterior. Cada teste possui partes AB e BA, decorrentes da inversão dos conjuntos de teste e de treinamentos. Cada tabela contém a quantidade de acertos em relação ao total de textos analisados e o tempo médio de processamento (em milissegundos).

As primeiras 6 tabelas representam os resultados de textos em tamanho integral, simulando *posts* de blogs.

Tabela 5.1: Resultados dos testes com textos de tamanho integral: *tf-idf*

tf-idf	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	99	2000/2000 (100,00%)	104
2	2000/2000 (100,00%)	92	2000/2000 (100,00%)	102
3	2000/2000 (100,00%)	94	2000/2000 (100,00%)	101
4	2000/2000 (100,00%)	100	2000/2000 (100,00%)	103
5	2000/2000 (100,00%)	103	2000/2000 (100,00%)	105
6	2000/2000 (100,00%)	98	2000/2000 (100,00%)	108
7	2000/2000 (100,00%)	98	2000/2000 (100,00%)	99
8	2000/2000 (100,00%)	99	2000/2000 (100,00%)	104
9	2000/2000 (100,00%)	94	2000/2000 (100,00%)	99
10	2000/2000 (100,00%)	92	2000/2000 (100,00%)	93

Fonte: elaborada pela autora.

Tabela 5.2: Resultados dos testes com textos de tamanho integral: booleano

booleano	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	35	2000/2000 (100,00%)	37
2	2000/2000 (100,00%)	33	2000/2000 (100,00%)	37
3	2000/2000 (100,00%)	33	2000/2000 (100,00%)	36
4	2000/2000 (100,00%)	35	2000/2000 (100,00%)	37
5	2000/2000 (100,00%)	36	2000/2000 (100,00%)	38
6	2000/2000 (100,00%)	35	2000/2000 (100,00%)	40
7	2000/2000 (100,00%)	35	2000/2000 (100,00%)	36
8	2000/2000 (100,00%)	36	2000/2000 (100,00%)	38
9	2000/2000 (100,00%)	34	2000/2000 (100,00%)	36
10	2000/2000 (100,00%)	33	2000/2000 (100,00%)	33

Fonte: elaborada pela autora.

Tabela 5.3: Resultados dos testes com textos de tamanho integral: *tf* (gramas de tamanho 2)

gramas 2	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1997/2000 (99,85%)	2	1996/2000 (99,80%)	2
2	1995/2000 (99,75%)	2	1997/2000 (99,85%)	2
3	1997/2000 (99,85%)	2	1994/2000 (99,70%)	2
4	1993/2000 (99,65%)	2	1999/2000 (99,95%)	2
5	1997/2000 (99,85%)	2	1996/2000 (99,80%)	2
6	1998/2000 (99,90%)	2	1994/2000 (99,70%)	2
7	1995/2000 (99,75%)	2	1997/2000 (99,85%)	2
8	1998/2000 (99,90%)	2	1994/2000 (99,70%)	2
9	1999/2000 (99,95%)	2	1994/2000 (99,70%)	2
10	1994/2000 (99,70%)	2	1998/2000 (99,90%)	2

Fonte: elaborada pela autora.

Tabela 5.4: Resultados dos testes com textos de tamanho integral: *tf* (gramas de tamanho 3)

gramas 3	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1999/2000 (99,95%)	24	2000/2000 (100,00%)	24
2	1999/2000 (99,95%)	24	2000/2000 (100,00%)	24
3	2000/2000 (100,00%)	24	2000/2000 (100,00%)	22
4	1999/2000 (99,95%)	24	2000/2000 (100,00%)	25
5	1999/2000 (99,95%)	24	1999/2000 (99,95%)	24
6	2000/2000 (100,00%)	24	1999/2000 (99,95%)	26
7	1998/2000 (99,90%)	24	2000/2000 (100,00%)	24
8	1998/2000 (99,90%)	24	1999/2000 (99,95%)	24
9	2000/2000 (100,00%)	24	1998/2000 (99,90%)	24
10	2000/2000 (100,00%)	24	1999/2000 (99,95%)	23

Fonte: elaborada pela autora.

Tabela 5.5: Resultados dos testes com textos de tamanho integral: *tf* (gramas de tamanho 4)

gramas 4	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	74	2000/2000 (100,00%)	74
2	2000/2000 (100,00%)	80	2000/2000 (100,00%)	75
3	2000/2000 (100,00%)	69	2000/2000 (100,00%)	71
4	2000/2000 (100,00%)	81	2000/2000 (100,00%)	88
5	2000/2000 (100,00%)	80	2000/2000 (100,00%)	72
6	2000/2000 (100,00%)	93	2000/2000 (100,00%)	94
7	2000/2000 (100,00%)	78	2000/2000 (100,00%)	72
8	2000/2000 (100,00%)	79	2000/2000 (100,00%)	76
9	2000/2000 (100,00%)	73	2000/2000 (100,00%)	73
10	2000/2000 (100,00%)	72	2000/2000 (100,00%)	71

Fonte: elaborada pela autora.

Tabela 5.6: Resultados dos testes com textos de tamanho integral: *tf* (gramas em geral)

gramas (geral)	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	100	2000/2000 (100,00%)	101
2	2000/2000 (100,00%)	107	2000/2000 (100,00%)	101
3	2000/2000 (100,00%)	96	2000/2000 (100,00%)	96
4	2000/2000 (100,00%)	108	2000/2000 (100,00%)	115
5	2000/2000 (100,00%)	107	2000/2000 (100,00%)	99
6	2000/2000 (100,00%)	120	2000/2000 (100,00%)	123
7	2000/2000 (100,00%)	106	2000/2000 (100,00%)	99
8	2000/2000 (100,00%)	105	2000/2000 (100,00%)	103
9	2000/2000 (100,00%)	99	2000/2000 (100,00%)	99
10	2000/2000 (100,00%)	99	2000/2000 (100,00%)	97

Fonte: elaborada pela autora.

A métrica booleana, que é a mais simples, acerta todos os casos e tem o melhor tempo de processamento. Como a métrica apenas leva em conta se o termo da consulta também consta na base, a escolha da base de dados é fundamental, o que traria erros nos resultados caso houvesse sobreajuste.

A métrica *tf-idf* é a que leva mais informações em consideração: leva em conta quantos documentos na base de dados contêm o termo e o tamanho da base em si. Seu resultado foi o esperado para uma métrica tão utilizada na área de classificação de textos e dadas as informações analisadas.

A métrica *tf*, utilizando gramas, mostrou também que uma análise simples gera bons resultados. Previsivelmente, os gramas de tamanho 2 são os que obtêm o pior resultado, visto que analisam apenas 2 letras, e os valores de similaridade ficam muito próximos, mas mesmo assim atingem no mínimo 99% de precisão. Mas quando se analisam as três variações juntas, o resultado é praticamente o mesmo de *tf-idf*, sem a necessidade de recorrer a informações como tamanho da base de dados.

Em geral, os resultados foram excelentes. As diferenças de similaridade foram em geral altas para os padrões *tf-idf* e booleano, que lidaram com as palavras inteiras, não deixando muita margem para erros. Quando se analisaram os erro de gramas de tamanho 2, verificou-se que as similaridades com os dois idiomas foram muito parecidas, pendendo para o idioma certo em mais de 99% dos casos. O que contou para essa quantidade de acertos foi a grande diferença entre os dois idiomas analisados, embora algumas palavras possuam as mesmas grafias.

As próximas 6 tabelas representam os resultados dos textos truncados para 140 caracteres, simulando *posts* do Twitter.

Tabela 5.7: Resultados dos testes com textos de tamanho 140 bytes: *tf-idf*

tf-idf	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	55	2000/2000 (100,00%)	68
2	2000/2000 (100,00%)	57	2000/2000 (100,00%)	62
3	2000/2000 (100,00%)	57	2000/2000 (100,00%)	59
4	2000/2000 (100,00%)	57	2000/2000 (100,00%)	58
5	2000/2000 (100,00%)	57	2000/2000 (100,00%)	58
6	2000/2000 (100,00%)	59	2000/2000 (100,00%)	61
7	2000/2000 (100,00%)	57	2000/2000 (100,00%)	57
8	2000/2000 (100,00%)	58	2000/2000 (100,00%)	56
9	2000/2000 (100,00%)	63	2000/2000 (100,00%)	59
10	2000/2000 (100,00%)	57	2000/2000 (100,00%)	61

Fonte: elaborada pela autora.

Tabela 5.8: Resultados dos testes com textos de tamanho 140 bytes: booleano

booleano	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	18	2000/2000 (100,00%)	22
2	2000/2000 (100,00%)	18	2000/2000 (100,00%)	20
3	2000/2000 (100,00%)	18	2000/2000 (100,00%)	19
4	2000/2000 (100,00%)	19	2000/2000 (100,00%)	19
5	2000/2000 (100,00%)	19	2000/2000 (100,00%)	19
6	2000/2000 (100,00%)	19	2000/2000 (100,00%)	20
7	2000/2000 (100,00%)	18	2000/2000 (100,00%)	19
8	2000/2000 (100,00%)	19	2000/2000 (100,00%)	18
9	2000/2000 (100,00%)	21	2000/2000 (100,00%)	19
10	2000/2000 (100,00%)	19	2000/2000 (100,00%)	20

Fonte: elaborada pela autora.

Tabela 5.9: Resultados dos testes com textos de tamanho 140 bytes: *tf* (gramas de tamanho 2)

gramas 2	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1947/2000 (97,35%)	0	1947/2000 (97,35%)	1
2	1951/2000 (97,55%)	0	1939/2000 (96,95%)	0
3	1948/2000 (97,40%)	0	1940/2000 (97,00%)	0
4	1954/2000 (97,70%)	0	1936/2000 (96,80%)	0
5	1945/2000 (97,25%)	0	1946/2000 (97,30%)	0
6	1951/2000 (97,55%)	0	1937/2000 (96,85%)	0
7	1942/2000 (97,10%)	0	1948/2000 (97,40%)	0
8	1949/2000 (97,45%)	0	1944/2000 (97,20%)	0
9	1956/2000 (97,80%)	0	1940/2000 (97,00%)	0
10	1958/2000 (97,90%)	0	1929/2000 (96,45%)	0

Fonte: elaborada pela autora.

Tabela 5.10: Resultados dos testes com textos de tamanho 140 bytes: *tf* (gramas de tamanho 3)

gramas 3	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1929/2000 (96,45%)	7	1929/2000 (96,45%)	8
2	1935/2000 (96,75%)	7	1952/2000 (97,60%)	7
3	1936/2000 (96,80%)	7	1945/2000 (97,25%)	7
4	1949/2000 (97,45%)	7	1933/2000 (96,65%)	7
5	1949/2000 (97,45%)	7	1933/2000 (96,65%)	7
6	1941/2000 (97,05%)	7	1944/2000 (97,20%)	7
7	1933/2000 (96,65%)	7	1952/2000 (97,60%)	7
8	1943/2000 (97,15%)	7	1943/2000 (97,15%)	7
9	1944/2000 (97,20%)	7	1934/2000 (96,70%)	7
10	1939/2000 (96,95%)	7	1943/2000 (97,15%)	7

Fonte: elaborada pela autora.

Tabela 5.11: Resultados dos testes com textos de tamanho 140 bytes: *tf* (gramas de tamanho 4)

gramas 4	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1961/2000 (98,05%)	19	1961/2000 (98,05%)	23
2	1966/2000 (98,30%)	20	1969/2000 (98,45%)	21
3	1964/2000 (98,20%)	20	1970/2000 (98,50%)	21
4	1970/2000 (98,50%)	20	1968/2000 (98,40%)	20
5	1975/2000 (98,75%)	21	1964/2000 (98,20%)	21
6	1972/2000 (98,60%)	26	1966/2000 (98,30%)	24
7	1967/2000 (98,35%)	23	1971/2000 (98,55%)	22
8	1970/2000 (98,50%)	21	1968/2000 (98,40%)	21
9	1969/2000 (98,45%)	21	1966/2000 (98,30%)	20
10	1969/2000 (98,45%)	20	1974/2000 (98,70%)	20

Fonte: elaborada pela autora.

Tabela 5.12: Resultados dos testes com textos de tamanho 140 bytes: *tf* (gramas em geral)

gramas (geral)	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1961/2000 (98,05%)	27	1961/2000 (98,05%)	32
2	1962/2000 (98,10%)	28	1969/2000 (98,45%)	29
3	1972/2000 (98,60%)	28	1973/2000 (98,65%)	29
4	1979/2000 (98,95%)	28	1963/2000 (98,15%)	28
5	1978/2000 (98,90%)	29	1966/2000 (98,30%)	29
6	1976/2000 (98,80%)	35	1966/2000 (98,30%)	33
7	1971/2000 (98,55%)	31	1970/2000 (98,50%)	30
8	1974/2000 (98,70%)	29	1969/2000 (98,45%)	29
9	1977/2000 (98,85%)	29	1966/2000 (98,30%)	28
10	1977/2000 (98,85%)	28	1967/2000 (98,35%)	29

Fonte: elaborada pela autora.

Novamente, as métricas *tf-idf* e booleana tiveram os melhores resultados em todos os casos, visto que lidam com o termo inteiro – e dificilmente os termos de um idioma se encontram na base de dados do outro. A métrica *tf*, por outro lado, teve um decréscimo de precisão, apesar de se manter acima dos 96% em seu pior resultado. Isso

se deve à pequena quantidade de texto para analisar, gerando poucas ocorrências de gramas e resultando em similaridades muito parecidas, algumas inclusive pendendo para o idioma errado, o que era esperado. Novamente, ao se analisar o geral da métrica tf, considerando as 3 variações de tamanho, o resultado é bem satisfatório para a simplicidade da função.

A nível de comparação da eficácia da ferramenta com os trabalhos relacionados (detalhados no Capítulo 2), verificou-se que TextCat atingiu 99,8% em suas melhores configurações - arquivos com mais de 300 bytes e perfil com quantidades 300 e 400 gramas selecionados -, e TypeAny detectou a linguagem correta em 96,7% dos textos do *corpora* multilingual gerado artificialmente. Comparando-se os resultados de TypeAny com os de Jess para o segundo conjunto de testes, onde o Jess atinge, no pior dos casos, 99,35% de precisão, ao se considerar o geral de *tf*, mostra-se que a análise da Jess, apesar de mais simples, apresenta resultados semelhantes.

Para comparar com o resultado do TextCat para textos com menos de 300 bytes, foram feitos testes na Jess onde os arquivos de teste foram truncados em 300 bytes (com os mesmos cuidados da truncagem de 140 bytes). Os resultados da Jess seguem nas tabelas a seguir:

Tabela 5.13: Resultados dos testes com textos de tamanho 300 bytes: *tf-idf*

tf-idf	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	63	2000/2000 (100,00%)	67
2	2000/2000 (100,00%)	66	2000/2000 (100,00%)	65
3	2000/2000 (100,00%)	68	2000/2000 (100,00%)	65
4	2000/2000 (100,00%)	69	2000/2000 (100,00%)	67
5	2000/2000 (100,00%)	67	2000/2000 (100,00%)	69
6	2000/2000 (100,00%)	63	2000/2000 (100,00%)	67
7	2000/2000 (100,00%)	70	2000/2000 (100,00%)	65
8	2000/2000 (100,00%)	64	2000/2000 (100,00%)	65
9	2000/2000 (100,00%)	66	2000/2000 (100,00%)	67
10	2000/2000 (100,00%)	63	2000/2000 (100,00%)	66

Fonte: elaborada pela autora.

Tabela 5.14: Resultados dos testes com textos de tamanho 300 bytes: booleano

booleano	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	2000/2000 (100,00%)	21	2000/2000 (100,00%)	22
2	2000/2000 (100,00%)	22	2000/2000 (100,00%)	22
3	2000/2000 (100,00%)	23	2000/2000 (100,00%)	22
4	2000/2000 (100,00%)	23	2000/2000 (100,00%)	22
5	2000/2000 (100,00%)	22	2000/2000 (100,00%)	23
6	2000/2000 (100,00%)	21	2000/2000 (100,00%)	22
7	2000/2000 (100,00%)	23	2000/2000 (100,00%)	22
8	2000/2000 (100,00%)	21	2000/2000 (100,00%)	22
9	2000/2000 (100,00%)	22	2000/2000 (100,00%)	23
10	2000/2000 (100,00%)	21	2000/2000 (100,00%)	22

Fonte: elaborada pela autora.

Tabela 5.15: Resultados dos testes com textos de tamanho 300 bytes: *tf* (gramas de tamanho 2)

gramas 2	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1987/2000 (99,35%)	1	1991/2000 (99,55%)	1
2	1989/2000 (99,45%)	1	1991/2000 (99,55%)	1
3	1984/2000 (99,20%)	1	1992/2000 (99,60%)	1
4	1991/2000 (99,55%)	1	1987/2000 (99,35%)	1
5	1991/2000 (99,55%)	1	1988/2000 (99,40%)	1
6	1987/2000 (99,35%)	1	1993/2000 (99,65%)	1
7	1986/2000 (99,30%)	1	1994/2000 (99,70%)	1
8	1990/2000 (99,50%)	1	1988/2000 (99,40%)	1
9	1986/2000 (99,30%)	1	1992/2000 (99,60%)	1
10	1993/2000 (99,65%)	1	1987/2000 (99,35%)	1

Fonte: elaborada pela autora.

Tabela 5.16: Resultados dos testes com textos de tamanho 300 bytes: *tf* (gramas de tamanho 3)

gramas 3	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1994/2000 (99,70%)	10	1995/2000 (99,75%)	10
2	1997/2000 (99,85%)	10	1992/2000 (99,60%)	10
3	1995/2000 (99,75%)	11	1992/2000 (99,60%)	10
4	1993/2000 (99,65%)	11	1992/2000 (99,60%)	10
5	1992/2000 (99,60%)	11	1997/2000 (99,85%)	10
6	1994/2000 (99,70%)	16	1995/2000 (99,75%)	10
7	1994/2000 (99,70%)	10	1993/2000 (99,65%)	10
8	1995/2000 (99,75%)	10	1993/2000 (99,65%)	10
9	1992/2000 (99,60%)	11	1995/2000 (99,75%)	10
10	1993/2000 (99,65%)	17	1994/2000 (99,70%)	10

Fonte: elaborada pela autora.

Tabela 5.17: Resultados dos testes com textos de tamanho 300 bytes: *tf* (gramas de tamanho 4)

gramas 4	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1992/2000 (99,60%)	30	1994/2000 (99,70%)	34
2	1994/2000 (99,70%)	31	1992/2000 (99,60%)	31
3	1995/2000 (99,75%)	32	1991/2000 (99,55%)	32
4	1994/2000 (99,70%)	33	1991/2000 (99,55%)	33
5	1992/2000 (99,60%)	31	1990/2000 (99,50%)	28
6	1996/2000 (99,80%)	50	1990/2000 (99,50%)	31
7	1992/2000 (99,60%)	31	1992/2000 (99,60%)	31
8	1995/2000 (99,75%)	28	1992/2000 (99,60%)	28
9	1992/2000 (99,60%)	36	1994/2000 (99,70%)	30
10	1989/2000 (99,45%)	40	1994/2000 (99,70%)	30

Fonte: elaborada pela autora.

Tabela 5.18: Resultados dos testes com textos de tamanho 300 bytes: *tf* (gramas em geral)

gramas (geral)	*AB*		*BA*	
Teste	*Acertos*	*Tempo (ms)*	*Acertos*	*Tempo (ms)*
1	1999/2000 (99,95%)	41	1998/2000 (99,90%)	47
2	2000/2000 (100,00%)	44	1996/2000 (99,80%)	43
3	1998/2000 (99,90%)	45	1999/2000 (99,95%)	45
4	1999/2000 (99,95%)	45	1997/2000 (99,85%)	45
5	1998/2000 (99,90%)	44	1999/2000 (99,95%)	41
6	1998/2000 (99,90%)	68	2000/2000 (100,00%)	43
7	1996/2000 (99,80%)	43	2000/2000 (100,00%)	43
8	1998/2000 (99,90%)	40	2000/2000 (100,00%)	41
9	1998/2000 (99,90%)	48	1999/2000 (99,95%)	42
10	1999/2000 (99,95%)	59	1999/2000 (99,95%)	42

Fonte: elaborada pela autora.

Como era de se esperar, os pesos *tf-idf* e booleano ainda possuem os melhores resultados, pois mesmo diminuindo a quantidade de informação, os idiomas ainda são diferentes entre si, quando se comparam as palavras inteiras, com algumas exceções de mesma grafia.

Já o peso *tf*, que utiliza gramas, chegou a 99,8% no seu pior caso (considerando o último resultado, o geral), se igualando ao TextCat. Os erros ocorrem pela quantidade de texto não gerar gramas suficientes para uma maior diferenciação dos idiomas.

6 CONCLUSÃO

A identificação do idioma em que um texto foi escrito é essencial para análises de diversos tipos, como correção ortográfica e tradução. Sendo um tipo de classificação de textos, existem diversas abordagens que podem solucionar o problema, e a abordagem deste trabalho resolve o problema satisfatoriamente. O modelo vetorial é de simples entendimento e estrutura, provendo todo o apoio necessário para a ferramenta da forma mais fácil o possível.

O que contou muito para a eficácia da ferramenta foi o conjunto de textos selecionados para treinamento: ambos os conjuntos de notícias estão escritos corretamente, salvo raros casos de codificação de caracteres, e possuem assuntos de áreas distintas, o que tornou o vocabulário abrangente.

Análises simples podem gerar excelentes resultados. Embora levasse mais informações em conta, a métrica *tf-idf* teve o mesmo resultado que a métrica booleana, assim como *tf* chegou praticamente nos mesmos resultados analisando os textos de mais de uma maneira. Como os melhores resultados foram obtidos com a palavra inteira, conclui-se que a identificação do idioma chega a melhores resultados quando se analisa os termos inteiros, pelo menos nos casos português e inglês.

A métrica booleana foi a que teve o melhor desempenho no menor tempo. Como citado anteriormente, isso se deve à escolha do conjunto de treinamento, bem escrito e de vocabulário abrangente.

Outra razão foi a distinção natural entre os dois idiomas, como foi discutido anteriormente. Uma possível continuação desse estudo seria a adição de mais um idioma - por exemplo, espanhol, que é bem similar ao português - e verificar se a eficácia permanece. Como não há funções que tratam os idiomas especificamente, o custo seria encontrar uma base de dados de qualidade e codificar mais um idioma nos resultados. Importante notar que quanto melhor o *corpora* escolhido para o treinamento, melhores os resultados de identificação dos idiomas.

REFERÊNCIAS

BENEDETTO, D.; CAGLIOTI, E.; LORETO, V. Language Trees and Zipping. **Physical Review Letters**, College Park, v.88, n.4, jan. 2002. Disponível em: <http://www.ccs.neu.edu/home/jaa/CSG399.05F/Topics/Papers/BenedettoCaLo.pdf>. Acesso em: nov. 2012.

LEWIS, M. P. Introduction to the Printed Volume. **Ethnologue: Languages of the World**, 16th ed. Dallas, 2009. Disponível em: <http://www.ethnologue.com/**.**>. Acesso em: nov. 2012.

W3TECHS. **Usage of content languages for websites**. [S.l.:s.n]. Disponível em: <http://w3techs.com/technologies/overview/content_language/all>. Acesso em: dez. 2012.

EUROPARL. **A tradução no Parlamento Europeu: de e para as 23 línguas oficiais da União Europeia**. REF. : 20120220STO38573. [S.l.:s.n]. Disponível em: < http://www.europarl.europa.eu/news/pt/headlines/content/20120220STO38573/html/A-tradução-no-Parlamento-Europeu-de-e-para-23-línguas> Acesso em: nov. 2012.

CAVNAR, W. B.; TRENKLE, J. M. N-Gram Based Text Categorization. **Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval**. Las Vegas, p. 161 175, apr. 1994.

EHARA, Y.; TANAKA-ISHII, K. **Multilingual Text Entry using Automatic Language Detection**. Tokyo: Graduate School of Information Science and Technology - University of Tokyo, 2008.

MARIÑO, J. et al. N-gram-based Machine Translation. **Computational Linguistics**. Cambridge: Massachusetts Institute of Technology, v. 32, n. 4, p. 527 549, dez. 2006.

BERGSMA, S.; PITLER, E.; LIN, D. Creating Robust Supervised Classifiers via Web-Scale N-gram Data. **Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics**. [S.l] p. 865 874, 2010.

SUZUKI, I. et al. A Language and Character Set Determination Method Based on N-gram Statistics. **ACM Transactions on Asian Language Information Processing (TALIP)**. New York, v. 1, i. 3, p. 269 278, sep. 2002.

PENG, F.; SCHUURMANS, D.; WANG, S. Language and task independent text categorization with simple language models. **NAACL'03 Proceedings of the 2003 Conference of the North American Chapter of the Association For Computational Linguistics on Human Language Technology**. Stroudsburg, v. 1, p. 110 117, 2003

BAEZA-YATES, R.; RIBEIRO-NETO, B.; GONÇALVES, M. Text Classification. In: BAEZA-YATES, R.; RIBEIRO-NETO, B. **Modern Information Retrieval: The**

Concepts and Technology behind Search. 2nd ed. [S.l.] Addison-Wesley Professional, 2011.

WITTEN, I. H.; FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques**. 2nd ed. San Francisco: Elsevier, 2005.

YANG, Y.; LIU, X. A re-examination of text categorization methods. **SIGIR '99 Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval**. New York, p. 42 49, 1999.

HOTHO, A.; NÜRNBERGER, A; PAAß, G. A Brief Survey of Text Mining. **LDV Forum - GLDV Journal for Computational Linguistics and Language Technology**. 2005

KUNCHEVA, L.; WHITAKER, C. J. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. **Machine Learning**. Netherlands: Kluwer Academic Publishers, v. 51, i. 2, p. 181 207, 2003.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. Cambridge: Cambridge University Press, 2008.

LAN, M. et al. A comprehensive comparative study on term weighting schemes for text categorization with support vector machines. **Proceeding WWW '05 Special interest tracks and posters of the 14th international conference on World Wide Web**. New York, p. 1023 1033, 2005.

HUANG, A. Similarity Measures for Text Document Clustering. **Proceedings of the New Zealand Computer Science Research Student Conference 2008**. Christchurch, [S.n.] 2008.

SCOTT, S.; MATWIN, S. Feature Engineering for Text Classification.**ICML '99 Proceedings of the Sixteenth International Conference on Machine Learning**. San Francisco, p. 379 388, 1999.

LEWIS, D. **Reuters-21578**. [S.l.:s.n]. Disponível em: <http://www.daviddlewis.com/resources/testcollections/reuters21578/> Acesso em out. 2012.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. **IJCAI'95 Proceedings of the 14th international joint conference on Artificial intelligence**. San Francisco, v. 2, p. 1137 1143, 1995.