UNIVERSIDADE
DE SÃO PAULO

Escola de Artes, Ciências e Humanidades

Gabriel José Magalhães Barbosa

Rubens Victor Gomes

Análise dos Erros de Digitação em Línguas com Acentos e sua Aplicação ao LibreOffice Writer

São Paulo

Janeiro de 2019

Universidade de São Paulo

Escola de Artes, Ciências e Humanidades

Gabriel José Magalhães Barbosa

Rubens Victor Gomes

Análise dos Erros de Digitação em Línguas com Acentos e sua Aplicação ao LibreOffice Writer

Monografia apresentada à Escola de Artes, Ciências e Humanidades, da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2018 – Projeto Supervisionado ou de Graduação II, para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Norton Trevisan Roman

Alunos: Gabriel José M. Barbosa (7163395)

Rubens Victor Gomes (8598878)

Modalidade: TCC Longo (1 ano) – em grupo

São Paulo

Janeiro de 2019

Universidade de São Paulo
Escola de Artes, Ciências e Humanidades

Gabriel José Magalhães Barbosa

Rubens Victor Gomes

Análise dos Erros de Digitação em Línguas com Acentos e sua Aplicação ao LibreOffice Writer

Orientação/Supervisor

_____
Prof. Dr. Norton Trevisan Roman

Banca Examinadora:

_____

Prof. Dr. _____

São Paulo, Janeiro de 2019

Agradecimentos

À Aurea Castelo Branco pelo tempo despendido para a correção dos erros para a língua francesa.

À Gökçen Piwek pelo tempo despendido para a correção dos erros para a língua turca.

Glossário

API: *Application Programming Interface.*

LO Writer: *LibreOffice Writer – editor de textos.*

SDK: *Software Development Kit.*

Resumo

Este projeto visa a implementar quatro algoritmos que se baseiam em distância de edição utilizando estatística de erros que ocorrem em quatro línguas que utilizam diacríticos para verificar se há melhorias nas ordens das listas de sugestões apresentadas por um editor de texto comercial. Inicialmente, foi feita uma listagem dos países cujas línguas utilizam diacríticos. Após a listagem, quatro línguas foram escolhidas tendo como critério de seleção: a quantidade de diacríticos utilizados nas línguas; o número de caracteres que utilizam diacríticos; a existência de pacotes para as línguas no *LibreOffice Writer*; a ausência de literatura sobre erros de grafia nas línguas em questão; e a existência de um nativo da língua para a correção dos erros. As línguas escolhidas foram o italiano, húngaro, francês e o turco. Após a escolha foi necessário fazer a coleta dos corpora digitados nas línguas em questão via *web crawler* construídos para esse fim e erros de digitação foram marcados, corrigidos e classificados.

Palavras chaves: Correção Ortográfica, Distância de Edição, N-grama, Inteligência Artificial, Processamento de Linguagem Natural.

Abstract

This project aims to implement four algorithms that are based on distance editing using error statistics that occur in four languages that use diacritics to check for improvements to the order of suggestion lists presented by a commercial text editor. Initially, a list was made of the countries whose languages use diacritics. After the listing, four languages were chosen with selection criteria: the amount of diacritics used in the languages; the number of characters using diacritics; the existence of packages for languages in LibreOffice Writer; the lack of literature on spelling errors in the languages concerned; and the existence of a native tongue for the correction of errors. The languages chosen were Italian, Hungarian, French and Turkish. After the choice it was necessary to collect the corpora typed in the languages in question via web crawler built for this purpose and typos were marked, corrected and classified.

Keywords: Orthographic Correction, Editing Distance, N-gram, Artificial Intelligence, Natural Language Processing.

Lista de Tabelas

Lista de Quadros

Lista de Figuras

Sumário

Introdução

A correção ortográfica consiste na tarefa de detecção e correção de erros ortográficos que ocorrem ao digitar um texto em um computador. Geralmente, para lidar com esses problemas, utilizamos ferramentas para correção ortográfica que nos auxiliem nessa tarefa. As ferramentas utilizadas na correção automática da ortografia são denominadas corretores ortográficos, que estão presentes em aplicações como serviços de e-mail e editores de texto.

A correção ortográfica consiste em fazer uma verificação de quais palavras estão ortograficamente incorretas e corrigi-las automaticamente, ou então prover para o autor uma lista de sugestões de palavras que podem substituí-las. Durante a fase de verificação, dois tipos de erros podem ser encontrados: (1) palavras mal formadas, encontradas por meio de consultas a um dicionário preexistente; ou (2) palavras existentes, ou seja, palavras presentes nos dicionários, mas que resultam de erro ortográfico, dependendo assim de uma análise do contexto para serem encontradas e corrigidas (KUKICH, 1992).

Corretores ortográficos comerciais focam principalmente em erros que dão origem a palavras mal formadas, comparando o que é digitado pelo usuário com as palavras de um dicionário interno. No que diz respeito a editores de texto comerciais, a ordem com que as palavras são apresentadas ao usuário é deficiente. Por exemplo, ao digitar 'Nao' no *LibreOffice Writer*[1] (*LO Writer*), a palavra correta 'Não' fica em sexto lugar e palavra 'Ao' é colocada na primeira posição. Esse fato nos dá um indício de que há uma necessidade de se dar uma atenção especial para línguas que utilizam diacríticos (GIMENES; ROMAN; CARVALHO, 2015).

A razão para essa desordenação reside no fato do corretor ortográfico do editor de texto se basear em dados estatísticos obtidos a partir da análise de textos produzidos na língua inglesa (DAMERAU, 1964). Segundo esses dados, 80% dos erros do tipo palavras mal formadas, tem, pelo menos, uma ocorrência de um de quatro tipos de erros que ele classifica como de (1) inserção: adição desnecessária de um caractere; (2) omissão: falta de um caractere (3) transposição: inversão da posição de dois caracteres adjacentes ou (4) substituição: substituição de caractere existente por outro.

Para línguas que utilizam diacríticos, como é o caso do português, uma reordenação dessas listas pode ser necessária. Um estudo realizado por GIMENES, ROMAN e CARVALHO (2015) mostrou que reordenar o ranking de sugestões para palavras em português com erros ortográficos no *LO Writer*, pode gerar uma melhoria de 10-21%, dependendo do corpus de teste.

A questão permanece, contudo, em relação a se essa melhoria deu-se apenas no português, ou seria algo que outras línguas que fazem uso intensivo de acentos poderiam utilizar. O projeto se propõe a determinar se essas estatísticas ainda seriam válidas em se tratando de quatro línguas que utilizam diacríticos (francês, turco, italiano e húngaro). Como o projeto será realizado em dupla, cada um dos autores ficará responsável por duas línguas.

As atividades relacionadas com as línguas francesa e turca foram realizadas pelo Gabriel e as atividades relacionadas às línguas italiana e húngara foram realizadas pelo Rubens.

Objetivos

Objetivo Geral

Implementar quatro algoritmos que se baseiam em distância de edição utilizando estatística de erros que ocorrem em duas línguas que utilizam diacríticos para verificar se há melhorias nas ordens das listas de sugestões apresentadas por um editor de texto comercial.

Objetivos Específicos

Os objetivos específicos deste trabalho são:

1) Analisar, através de um conjunto de textos digitados em quatro línguas que utilizam diacríticos, os erros que podem ocorrer em textos digitados, bem como suas respectivas frequências;

2) Desenvolver o algoritmo descrito em (PETERSON, 1980) para as quatro línguas, levando em consideração as frequências encontradas no item anterior;

3) Comparar as ordens sugeridas por esse algoritmo com as ordens sugeridas pelo editor de texto LO Writer e identificar possíveis melhorias;

4) Definir um modelo bayesiano para reordenação da lista de sugestões do *LO Writer,* usando os resultados dos itens 1 e 2;

5) Comparar a ordem sugerida pelo quarto item com a ordem obtida pelo segundo item e com as ordens sugeridas pelo editor de texto *LO Writer;*

6) Determinar se erros ortográficos envolvendo diacríticos tem origem ortográfica ou tipográfica.

Revisão Bibliográfica

Tipos de erros de digitação

No campo de estudo da aprendizagem motora, o termo habilidade é utilizado para designar uma tarefa com uma finalidade específica a ser atingida (MAGILL, 2000). A habilidade de digitar em um teclado pode ser classificada como uma habilidade motora fina e serial. É uma habilidade motora fina uma vez que requer maior controle de músculos pequenos, que envolvem os músculos responsáveis na coordenação mãos-olhos, e exige um alto grau de precisão no movimento da mão e dos dedos (MAGILL, 2000). Também é uma habilidade motora serial, já que para se digitar uma palavra é necessário executar movimentos discretos para pressionar as teclas do teclado numa ordem serial definida (MAGILL, 2000).

Segundo MAGILL (2000), a habilidade de escrever à mão não depende apenas do aspecto motor. Para executá-lo é necessária a ação simultânea de vários processos de controle. Esses mesmos processos podem ser facilmente estendidos à habilidade de digitar em um teclado. Para digitar uma sentença, o indivíduo precisa utilizar tanto processos léxicos semânticos quanto de controle motor. O ato de digitar requer que o indivíduo recupere palavras da memória, palavras estas que precisam ter um significado que se ajuste ao que o escritor deseja expressar. Para digitar uma sentença é preciso dispor de uma construção gramatical. Para compor uma palavra, é preciso digitar certas letras na sequência correta. Além disso, o indivíduo precisa pressionar a tecla com a força necessária para produzir as letras. Qualquer falha nessa cadeia de processos de controle pode gerar um erro de grafia.

Os erros de grafia podem ser divididos segundo a sua origem ou conforme o seu resultado (já explicado anteriormente). Segundo a sua origem, os erros podem ser classificados como ortográficos ou tipográficos. Os erros ortográficos são erros cognitivos, que consistem na grafia errada no lugar da correta quando o autor não conhece ou está incerto quanto à grafia correta, ocorrendo devido à existência de semelhança fonológica entre a palavra digitada incorretamente e a palavra que se pretendia escrever (BERKEL e SMEDT, 1988). Já os erros tipográficos são erros motores causados ao se digitar uma sequência incorreta de letras que dependem do teclado utilizado, pois a proximidade com

que as letras aparecem no teclado e a língua em que se está escrevendo podem interferir na grafia correta da palavra (BERKEL e SMEDT, 1988).

Layout dos Teclados

WAGNER et al. (2003) definem um teclado como um instrumento que serve para inserir uma sequência de caracteres em uma memória, tocando uma sequência de teclas. Uma definição abstrata de teclado, portanto, consiste em um conjunto de caracteres possíveis, as respectivas sequências de teclas que permitem sua construção e uma configuração geométrica das teclas. A maioria dos teclados comerciais tem um layout que corresponde a um retângulo com linhas e colunas. Para digitar, cada coluna é geralmente atribuída a um dedo e a chamada linha inicial é definida onde os dedos permanecem em uma posição relaxada. Um modelo geral de teclado deve levar em conta esses aspectos para permitir a descrição de uma grande família de teclados.

Introduzido pelos irmãos Sholes em 1873, o arranjo de teclas conhecidos como QWERTY, foi reconhecido em 1971 pela *International Standards Organization* como o arranjo a ser usado no teclado padrão (NOYES, 1983). Inicialmente projetados em um contexto de língua inglesa, eles foram ligeiramente alterados e adaptados para outras línguas, como o francês e o alemão. Isso deu origem às versões AZERTY e QWERTZ (WAGNER et al., 2003). Segundo WAGNER et al. (2003), O arranjo não tem ordem lógica alfanumérica nem é otimizado de acordo com uma alta taxa de acertos ou conforto ergonômico. De fato, é mais o resultado do desenvolvimento histórico do mercado de máquinas de escrever, e notavelmente o domínio alcançado da máquina de escrever *Remington II*, que impôs o arranjo.

Em defesa do layout QWERTY, KINKEAD (1975) citado por NOYES (1983) descobriu que o layout padrão QWERTY pode ser operado durante a digitação a velocidades quase máximas para a língua Inglesa. Ele fez uma análise dos tempos de digitação e concluiu que o teclado ideal deve garantir, durante a digitação, que quase toda tecla digitada seja digitada de forma alternada entre as mãos. O teclado QWERTY faz isso. Seja por acidente ou design, o teclado padrão é quase ideal em termos de velocidade. O problema de um teclado ideal em termos de velocidade, reside no princípio fundamental do desempenho motor de que existe um compromisso entre a velocidade e precisão. Quando o indivíduo dá

mais ênfase à velocidade, a precisão é reduzida, o que leva a um maior número de erros (MAGILL, 2000).

Correção Ortográfica

Correção ortográfica consiste em detectar e corrigir erros. Esses subproblemas podem ser abordados de duas maneiras: (1) Na detecção e correção de erros de palavras isoladas, cada palavra é verificada separadamente, abstraindo de seu contexto; ou (2) Na detecção e correção de erros utilizando contexto. Muitos verificadores ortográficos contemporâneos usam a primeira abordagem, uma vez que a detecção e correção de erros utilizando contexto exigem análise gramatical e, portanto, são mais complexos e dependentes do idioma (DEOROWICS e CIURA, 2005; PETERSON, 1986). Uma terceira abordagem seria analisar a legibilidade do texto como um todo (MEDEIROS, 1995).

A correção ortográfica pode ser realizada de forma interativa ou automática. Na correção ortográfica interativa, a correção é feita pelo usuário e, para facilitar esta tarefa, uma lista de correções é apresentada e devem ser classificadas por ordem decrescente de probabilidade (PETERSON, 1986). Na correção ortográfica automática, a palavra correta é escolhida sem interação do usuário. Portanto, apenas uma palavra correta deve ser gerada (DEOROWICS e CIURA, 2005).

Os Corretores ortográficos modernos não são perfeitos. A maioria dos corretores ortográficos utiliza a abordagem de detecção e correção de erros de palavras isoladas, falhando em detectar erros do tipo palavras existentes (DEOROWICS e CIURA, 2005). Dentre as várias técnicas para a correção de palavras isoladas, as mais comuns são: técnica de similaridade de palavras, técnicas probabilísticas e a técnica de distância de edição, sendo essa última a mais utilizada (JURAFSKY e MARTIN, 2009; DEOROWICS e CIURA, 2005; PETERSON, 1986).

Metodologia

Inicialmente foi feita uma listagem dos países cujas línguas utilizam diacríticos. Após a listagem, duas línguas foram escolhidas tendo como critério de seleção: a quantidade de diacríticos utilizados nas línguas; o número de caracteres que utilizam diacríticos; a existência de pacotes para as línguas no *LO Writer*; a ausência de literatura sobre erros de grafia nas línguas em questão; e a existência de um nativo da língua para a correção dos erros. Após a escolha, foi necessário fazer a coleta dos corpora digitados nas línguas em questão via *web crawler* e erros de digitação foram marcados e pré-corrigidos. Mais detalhes estão descritos abaixo.

O editor de texto comercial *LO Writer* foi escolhido por este ser um software de código aberto que fornece um *Software Development Kit* (SDK), o que torna possível a implementação de pacotes para o mesmo. A *Application Programming Interface* (API) do *LibreOffice* é independente da linguagem, permitindo a programação em diferentes linguagens de programação como C++, Python, CLI, StarBasic, JavaScript, OLE e Java. Esses fatores serão importantes para a segunda etapa do projeto.

A listagem de países cujas línguas utilizam diacríticos focou-se no continente europeu. A lista foi ordenada segundo uma soma obtida do número de diacríticos e do número de caracteres que fazem uso desses sinais gráficos. A lista também contém, para cada país, quais diacríticos são utilizados e quais caracteres utilizam os mesmos.

Após a listagem das línguas, os layouts dos teclados das línguas foi verificado. Em princípio, a análise dos layouts é importante como critério de seleção porque erros envolvendo diacríticos, em tese, ocorrem pelo fato de se precisar bater em duas teclas para produzir um caractere. Nos teclados em que isso não ocorre, o custo de se digitar um caractere acentuado seria exatamente o mesmo que uma tecla comum, e erros de acentuação não deveriam diferir de erros em teclas sem acento.

Após determinamos as principais línguas candidatas a serem utilizadas no projeto, buscamos entrar em contato com nativos das mesmas. Outro critério de seleção foi a inexistência de trabalhos científicos sobre a distribuição de erros de digitação nessas línguas.

Para cada língua cogitada, em que um nativo em potencial já estava definido, foi feita uma pesquisa no *Google Scholar*[2], digitando-se: <língua> *typing error* e <língua> *spelling error* adicionando *diacritics* e *accents* a cada um desses (adiciona primeiro um, depois o outro). Os artigos encontrados foram baixados, e sua utilidade foi analisada. Também foi verificada a existência de módulos no *LO Writer* que trate dessas línguas.

Em seguida, foram definidos blogs de viagem, e textos foram coletados destes via *web crawler*. Foi seguido o modelo apresentado no artigo publicado por GIMENES, ROMAN e CARVALHO (2015). Os textos e os comentários dos blogs foram coletados. Um único site foi utilizado para cada idioma, mantendo uma compatibilidade com o número de erros do corpus inicial apresentado no artigo (perto de 1.200 erros).

Após a escolha foi desenvolvido o *web crawler* para coleta dos *corpora* para as línguas francesa e italiana, no qual foi utilizada a linguagem Python e o *framework* Scrapy, este escolhido para suprir as necessidades técnicas de ferramentas com este objetivo, como navegação pelo navegador programaticamente, onde seria realizada uma navegação pelas páginas. No geral, o processo de coleta ocorria da seguinte maneira: visita na página de diários de viagens; visita dos links que eram separados por hotéis, trajetos e cidades; e coleta dos comentários que constituíram os *corpora*. O código do *crawler* pode ser visto nos apêndices.

Para as línguas húngara e turca, esperava-se inicialmente que o mesmo modelo de *web crawler* fosse utilizado, porém foi necessário um novo projeto, em linguagem diferente, utilizando outras bibliotecas e *pipeline* de processamento. Isso ocorreu devido à restrições que o Python e o *framework* utilizado causaram no carregamento e interação com as páginas dos blogs, pois estas utilizavam de recursos da linguagem Javascript junto a recursos do navegador para carregamento dinâmico do conteúdo das postagens.

A solução encontrada foi o desenvolvimento de um novo *web crawler* utilizando a linguagem Javascript, a biblioteca NightmareJS para a navegação no navegador e o interpretador NodeJS para o *pipeline* de execução do código Javascript. Estas ferramentas possibilitaram maior compatibilidade do *web crawler* com as páginas, permitindo o controle do carregamento dinâmico. O código do *crawler* pode ser visto no APÊNDICE G.

Após a coleta, os textos coletados para as línguas francesa e turca foram abertos no *LO Writer*, erros de digitação foram marcados e a sentença em que os mesmos estão inseridos foram extraídas para que pudessem ser mostradas a um nativo da língua. Para a marcação da

palavra errada foram utilizados colchetes duplos na forma: [[E]], onde E é a palavra errada identificada pelo corretor ortográfico. Para facilitar a tarefa dos nativos, na medida do possível, os erros foram pré-corrigidos, utilizando o *Google Translate*[3] e a lista de sugestões dos corretores ortográficos. Os erros cuja correção não foi possível foram encaminhados para correção pelos nativos.

O cronograma das atividades realizadas na primeira etapa do projeto está apresentado no quadro 1.

Quadro 1 – Cronograma das atividades realizadas

	Março	Abril	Maio	Junho
Listagem dos países				
Análise dos layouts dos teclados				
Revisão Bibliográfica				
Busca por nativos				
Busca por trabalhos científicos				
Busca por verificadores ortográficos				
Definição dos blogues				
Desenvolvimento dos *web crawler's*				
Coleta dos corpora				
Anotação e pré-correção dos erros (Frances e Turco)				
Escrita do relatório parcial				

Para o italiano Inicialmente era selecionada uma palavra do grupo de potenciais erros apontados pelo *Hunspell* juntamente ao *WebSpellChecker*, uma *API* de correção ortográfica, e era realizada a correção da palavra utilizando dicionários online em Italiano e pesquisas em páginas de correção automática, analisando se havia erros de acentuação ou escrita, além de espaçamento, após isso era validado no corretor do *LO Writer* para uma segunda análise. Por fim, a correção era validada juntamente às ferramentas online de verificação ortográfica que permitiam a sugestão de contextos e textos com exemplos, permitindo uma validação de concordância e interpretação da palavra.

Para a segunda parte do projeto o objetivo foi implementar quatro extensões (uma para cada língua selecionada) para o *LO Writer*, escritas em Java, com a função de corretores ortográficos. Cada extensão utilizará a distribuição de tipos de erros obtida através das análises que serão realizadas para os tipos de erros encontrados nos corpora. O serviço por

elas modificado será o de corretor ortográfico, que faz parte dos serviços de linguística do *LO Writer*.

Para implementar as extensões, o algoritmo proposto por Peterson (1980) é o que mais se adapta para a finalidade proposta neste projeto, pela possibilidade de se verificar o tipo de erro presente em uma dada palavra e identificar a operação necessária para torná-la em alguma palavra existente no dicionário. Adaptações serão necessárias uma vez que os sistemas propostos são apenas ordenadores, fazendo com que a comparação não seja feita com um dicionário, mas sim com a própria lista oferecida pelo corretor ortográfico do editor de textos.

O algoritmo especifica que apenas serão consideradas para correção as palavras que tenham uma determinada distância de edição em relação à errada, sendo o custo de cada operação dado pela probabilidade de se encontrar erros com esse tipo de operação. Para tanto, o resultado obtido a partir das análises dos tipos de erros será utilizado como estimativa para essa probabilidade, com base na frequência com que cada tipo de erro acontece nos corpora. Assim, para cada sugestão de correção será dado um peso, dependendo da correção que será necessária, de acordo com a probabilidade desse tipo de erro ocorrer. Por fim, a lista de sugestões é reordenada, sendo colocada na primeira posição a palavra que apresenta o maior peso. Para a segunda etapa do projeto as seguintes atividades foram realizadas:

1) Análise dos tipos de erros cometidos nos corpora de estudo, conforme classificação da literatura relacionada;

Devido a dificuldades encontradas ao decorrer do projeto, as seguintes atividades ainda estão em desenvolvimento:

2) Implementação do algoritmo descrito em (PETERSON, 1980);

3) Incorporação de uma análise de bigramas ou trigramas, baseada em aprendizado bayesiano (JURAFSKY e MARTIN, 2009);

4) Reordenação das listas sugeridas pelo *LO Writer*, usando os novos protótipos desenvolvidos, e comparação dos resultados;

O cronograma das atividades realizadas na segunda etapa do projeto está apresentado no quadro 2.

Quadro 2 – Cronograma das atividades realizadas

	Agosto	Setembro	Outubro	Novembro
Correção dos erros para o italiano				
Classifcação dos erros				
Implementação dos algoritmos				
Incorporação de uma análise de Bigramas ou trigramas				
Reordenação das listas sugeridas pelo *LO Writer*				
Comparação dos resultados				
Escrita da monografia				

Resultados e discussão

Listagem dos países

A listagem dos países foi elaborada pelo aluno Gabriel, cruzando as informações contidas no mapa elaborado por MARIAN (2017) sobre diacríticos utilizados em línguas europeias, e as informações contidas em WIKIPÉDIA (2018) sobre diacríticos.

No total foram listados 25 países. Para cada país a lista contém o número de diacríticos utilizados e quais são esses diacríticos; o número de caracteres que utilizam os diacríticos e quais caracteres os utilizam, e um score obtido através da soma do número de diacríticos com o número de caracteres. A lista foi ordenada segundo o score. Na Tabela 1 estão apresentados os 14 países europeus cujas línguas mais utilizam diacríticos. A lista completa pode ser observada no APÊNDICE A. Como já existem resultados para o português e o espanhol essas línguas foram descartadas. O alemão também foi descartado pelo fato de utilizar apenas um diacrítico.

Tabela 1 – Lista dos quatorze países que mais utilizam diacríticos

Pais	Número de Diacríticos	Número de Caracteres	Score
República Eslovaca	4	14	18
Tcheca	3	13	16
República da Letônia	3	11	14
França	6	6	12
Portugal	6	6	12
República da Lituânia	4	7	11
República da Polônia	3	7	10
Espanha	3	6	9
Islândia	2	6	8
Hungria	3	5	8
República da Estônia	3	5	8
República da Turquia	3	5	8
Itália	2	5	7
Romênia	3	4	7

Layout dos teclados

A análise dos layouts dos teclados foi realizada utilizando os layouts disponibilizados por SHARKEY e WENZEL (2017), pelo aluno Gabriel. Durante a análise, foi notado que, saindo da península ibérica, os teclados passam a conter muitos caracteres já acentuados. Na Tabela 2 estão apresentados o número de caracteres já acentuados presentes em alguns teclados europeus. Observando a tabela, nota-se que os teclados português e espanhol, embora possuam um alto número de caracteres acentuados, são exceções nesse contexto e apresentam apenas um caractere já acentuado presente em seus respectivos teclados. Os layouts dos teclados exibidos na Tabela 2 estão disponíveis no APÊNDICE B.

Tabela 2 – Teclados europeus e caracteres já acentuados

Teclado	Número de caracteres que utilizam diacríticos	Número de caracteres acentuados presentes no teclado
Tcheco	15	11
Húngaro	9	9
Turco	6	6
Francês	13	5
Italiano	6	5
Alemão	3	3
Português	12	1
Espanhol	7	1

Embora a presença dos caracteres já acentuados nos teclados diminuam as chances de erros de origem tipográfica, já que não é necessário pressionar mais de uma tecla para digitá-los, os diacríticos têm função fonológica, ou seja, os erros envolvendo diacríticos parecem ser mais ortográficos do que tipográficos. Esses fatos levaram a questionar a relevância do layout do teclado para erros envolvendo diacríticos. O questionamento levou a elaboração de duas hipóteses:

1) Erros envolvendo diacríticos advém de um desconhecimento com relação à grafia correta da palavra, ou seja, têm origem ortográfica;
2) Erros envolvendo diacríticos advém de problemas motores, pelo fato de se precisar digitar 2 teclas para produzir um caractere, ou seja, têm origem tipográfica;

A hipótese 2 encontra subsídios em um artigo publicado por MAX e WISNIEWSKI (2010), que encontraram uma grande proporção de erros envolvendo diacríticos na língua francesa, 32.39%, mesmo o teclado francês contendo os caracteres já acentuados (cinco dos treze caracteres que utilizam diacríticos já estão presentes no teclado).

Por conta dessas hipóteses, ficou decidido que o teclado não influenciaria na escolha das línguas. Se, ao final, for notada diferenças entre as distribuições de erros de acentuação, quando comparando línguas cujos teclados incluem o caractere acentuado e as que não, então essa diferença pode ter sido por conta do teclado. Se não observarmos diferença, então foi desconhecimento da palavra.

Busca por nativos, trabalhos científicos e verificadores ortográficos

Conforme mencionado, um ponto importante do projeto é a existência de nativos das línguas para a correção dos erros encontrados, uma vez que não somos fluentes em nenhuma das línguas listadas que não foram descartadas. Buscamos contato com nativos para as seguintes línguas: francês; turco; italiano; húngaro; e tcheco. Após o contato, conseguimos nativos voluntários para as línguas francesa, turca, italiana e húngara. No decorrer do projeto tivemos problemas de disponibilidade com os nativos para as línguas italiana e húngara. Como o italiano é uma língua próxima e encontramos uma *API* que ajudava a identificar os erros, optamos por mantê-la no projeto.

Durante a busca por trabalhos científicos, nenhum artigo foi encontrado para o turco e italiano. Já para o francês, foram encontrados dois artigos. No primeiro, já mencionado, MAX e WISNIEWSKI (2010) analisaram os históricos de revisões de páginas da *Wikipédia* francesa. No artigo eles constataram que a maior parte das correções de palavras mal formadas, 32.39%, estavam relacionados ao uso de diacríticos. O artigo foi considerado incompleto uma vez que não apresentava as mesmas distribuições de erro apresentadas por GIMENES, ROMAN e CARVALHO (2015), não permitindo comparação. No segundo, STARLANDER e POPESCU-BELIS (2002) buscaram descrever um método de avaliação de verificadores de ortografia e gramática e mostraram os resultados da sua aplicação a dois verificadores franceses. Em seu corpus de estudo 4.8% dos erros eram erros de acentuação, mas novamente o artigo foi considerado incompleto por não apresentava as mesmas distribuições de erro apresentadas por GIMENES, ROMAN e CARVALHO (2015), não permitindo comparação.

Como os artigos encontrados para o francês foram considerados incompletos, e para o turco nenhum foi encontrado, só faltava verificar a existência de pacotes de verificação ortográfica para o *LO Writer*. Para o francês foi encontrado o *Dictionnaires francais*[4], para o turco foi encontrado o *Turkish Spellcheck Dictionary*[5] e para o italiano foi encontrado o *Italian dictionary*[6]. Assim, para a realização do projeto foram escolhidas as línguas francesa, turca e italiana.

Definição dos blogues e coleta dos corpora

Foi definido que o site do *TripAdvisor* seria utilizado para a coleta dos corpora francês e turco. A escolha pelo site *TripAdvisor* foi feita por ele apresentar pessoas narrando suas viagens e outras comentando, como em um dos corpora de GIMENES, ROMAN e CARVALHO (2015). Para o francês foi utilizado o *TripAdvisor FRANCE*[7], para o turco foi utilizado o *TripAdvisor TÜRKIYE*[8] e para o italiano foi utilizado o *TripAdvisor ITALIA*[9].

Existem dois problemas ao se coletar textos da web que precisam ser destacados: (1) não temos como saber se os textos foram escritos via teclado em um computador ou via teclado em dispositivo móvel (celular ou tablet); e (2) navegadores web e dispositivos móveis possuem verificadores ortográficos embutidos. O primeiro não é grave porque os layouts dos teclados para dispositivos móveis seguem o layout dos teclados para computadores e, o segundo, só implica em um número menor de erros nos comentários.

Tabela 3 – Informações dos corpora coletados

Corpus	Número de páginas	Número de palavras	Número de comentários
Francês	3.076	762.294	6.857
Italiano	3.196	2.085.730	10.416
Húngaro	2.941	1.226.064	12.403
Turco	3.763	1.378.782	16.282

Foram coletados textos datados de 11/05/2007 até 9/05/2018 via *web crawler*. Após a coleta, os corpora foram abertos no *LO Writer* para uma análise inicial. O resultado da coleta está apresentado na Tabela 3. Observando a Tabela 3 nota-se que os *corpora* apresentam tamanhos diferentes, mas isso não é um problema. Um problema encontrado em todos os *corpora* é que, além dos comentários, eles continham marcadores e informações sobre cada comentário que são desnecessários e dificultam o processamento. Assim, para ambos os corpus foram removidas as informações desnecessárias. O resultado da remoção está apresentado na Tabela 4.

Tabela 4 – Informações dos corpora coletados após a limpeza

Corpus	Número de páginas	Número de palavras	Número de comentários
Francês	1.156	607.168	6.857
Italiano	3.881	1.917.880	10.416
Húngaro	2.231	1.012.995	12.403
Turco	2.326	984.210	16.282

Anotação e correção dos erros para as línguas francesa e turca

Conforme mencionado, o objetivo era que os corpora apresentassem uma compatibilidade com o número de erros do corpus inicial apresentado por GIMENES, ROMAN e CARVALHO (2015), ou seja, cerca de 1.200 erros.

Assim, os corpora foram processados na ordem em que foram baixados, manualmente, comentário a comentário pelo aluno Gabriel, até que fosse encontrado o número de erros desejado. Uma vez atingido um número aceitável de erros, foi realizado um corte no corpus original, que era muito grande para ser processado inteiro. Cada palavra errada foi marcada utilizando-se colchetes duplos na forma: [[E]], onde E é a palavra errada identificada pelo corretor ortográfico.

Após cada marcação, a sentença em que a palavra errada estava inserida era separada e uma tentativa de correção era feita, utilizando a lista de sugestões e o *Google Translate.* Em caso de sucesso, a sentença era adicionada a uma lista de erros pré-corrigidos e a palavra correta foi colocada dentro dos colchetes duplos na forma: [[E (C)]], onde C é a palavra correta. Caso contrário, a sentença era adicionada a uma lista de erros para correção a serem encaminhados ao nativo. Detalhes da metodologia estão disponíveis no APÊNDICE D.

A pré-correção dos erros foi realizada visando a minimizar a carga de trabalho dos nativos, uma vez que só temos um nativo para cada língua e os mesmos têm disponibilidade de tempo limitada. Assim, aos nativos ficou incumbida a tarefa de corrigir os erros cuja correção não foi possível. Na Tabela 5 estão apresentadas as informações dos corpora processados.

Tabela 5 – Informações dos corpora processados

Corpus	Número de páginas	Número de palavras	Número de comentários	Número de palavras erradas
Francês	239	158.652	1.146	1.286
Turco	66	36.260	462	2.023

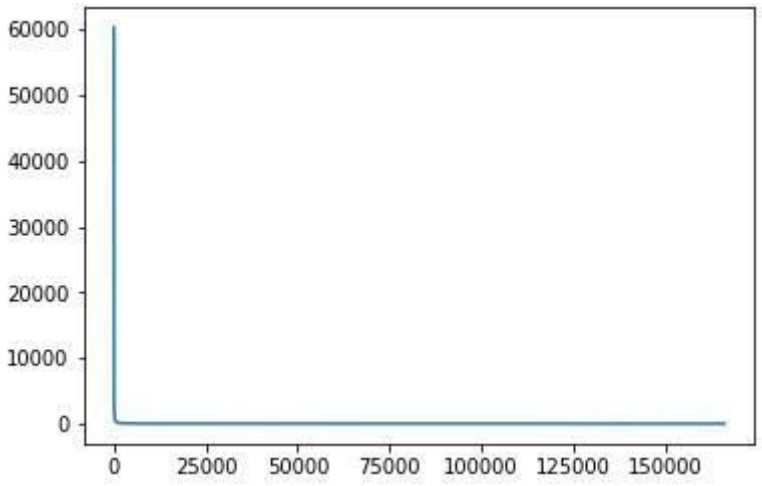
Correção dos erros para a língua italiana

Para a identificação dos erros para o italiano, realizada pelo Rubens, inicialmente haveria a participação de dois voluntários que iriam realizar a identificação das palavras com erro nas postagens utilizando um *website* desenvolvido, porém decidiram não continuar a realizar as correções devido ao alto volume e esforço exigido dos voluntários.

Então foi desenvolvido um programa que se conectava ao *WebSpellChecker* e realizava a verificação das postagens, identificando quais palavras eram potencialmente erros e quais correções sugeridas. Após isso fora construída uma tabela constituída pelas 10.855 palavras distintas identificadas como erros no *WebSpellChecker* e suas respectivas sugestões de correções, das quais foram corrigidas 1.008 utilizando de dicionários online e pesquisas em sites de correção ortográfica em italiano, formando o Grupo 1 de correções no total de 142 correções válidas distintas.

O Grupo 2 foi formado por palavras que não haviam sido identificadas pelo *WebSpellChecker* e que continham acentos que resultaram em 114 correções distintas. O Grupo 3 de correções foi formado pelas correções realizadas após a verificação da distribuição de todo o vocabulário do corpus.

Figura 1 – Palavras x Ocorrências no corpus



Após a visualização da figura 1, percebeu-se que a frequência de palavras estava com alta concentração, o que indicava a alta representatividade do corpus em apenas poucas palavras. A partir desta análise foi construído o terceiro grupo de palavras acentuadas e não identificadas pelo *Hunspell* no total de 252 ordenadas por frequência a fim de maximizar a representatividade da distribuição do corpus.

Este grupo foi corrigido utilizando as mesmas ferramentas que o Grupo 1 e 2, resultando na adição de 92 palavras corrigidas distintas. Por fim, foram combinados os grupos, totalizando 344 correções distintas e 6.872 ocorrências de erros.

Classificação dos Erros

Como forma de classificação dos erros encontrados, foram consideradas as categorias originalmente definidas por Damerau (1964) e acrescentado erros as classes de erros apresentadas por GIMENES, ROMAN e CARVALHO (2015). Dentro dessas categorias estão inclusos os erros com acentuação, divididos em e cinco subtipos: (1) falta de acento, (2) colocação desnecessária do acento, (3) acento correto colocado na letra errada, (4) acento errado colocado na letra certa, ou (5) erro com "ç". A Tabela 6 ilustra esses resultados. Detalhes sobre a classificação estão disponíveis no APÊNDICE E.

Tabela 6 – Frequência e distribuição de erros nos *corpora*

	Erros		
	Turco	**Francês**	**Italiano**
Categoria do erro	**Total (%)**	**Total (%)**	**Total (%)**
Inserção	196(6,97)	65(4,71)	134(2,03)
Omissão	144(5,12)	139(10,07)	32(0,48)
Transposição	9(0,32)	11(0,80)	4(0,06)
Substituição	821(29,19)	57(4,13)	11(0,17)
Falta de diacrítico	215(7,64)	821(59,45)	30(0,45)
Adição de diacrítico	6(0,21)	21(1,52)	42(0,64)
Diacrítico errado na letra certa	8(0,28)	56(4,06)	6220(94,16)
Diacrítico correto na letra errada	1(0,04)	2(0,14)	0(0,00)
Falta de cedilha	351(12,48)	37(2,68)	0(0,00)
Substituição por espaço	6(0,21)	9(0,65)	0(0,00)
Inserção de espaço	22(0,78)	7(0,51)	0(0,00)
Transposição de espaço	0(0,00)	1(0,07)	0(0,00)
Falta de espaço	282(10,02)	10(0,72)	17(0,26)
Outros	6(0,21)	20(1,45)	0(0,00)
Múltiplos erros	746(26,52)	125(9,05)	116(1,76)
Total	2813(100,00)	1381(100)	6606(100)

Corretor Automático

O corretor automático foi desenvolvimento a partir da implementação apresentada em GIMENES, ROMAN e CARVALHO (2015) para os idiomas francês, turco e italiano. O mesmo está apresentado no Anexo J. A língua húngara não foi incluída pois a etapa de classificação não foi finalizada até o final deste trabalho devida a falta de voluntários nativos.

Para o italiano foram comparadas as listas de sugestões de 218 palavras distintas com erros obtidos no processo de identificação dos erros no corpus, mas não foram obtidas melhorias na ordenação da lista de sugestões, conforme podemos verificar nas imagens abaixo.

Imagem 2 - Sugestões para a palavra *pò* do LO Writer

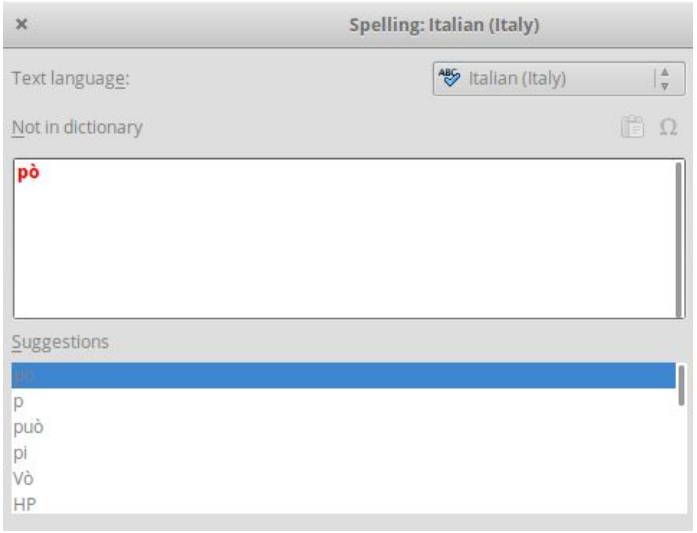


Imagem 3 - Sugestões para a palavra *pò* reordenadas utilizando distância Levenshtein

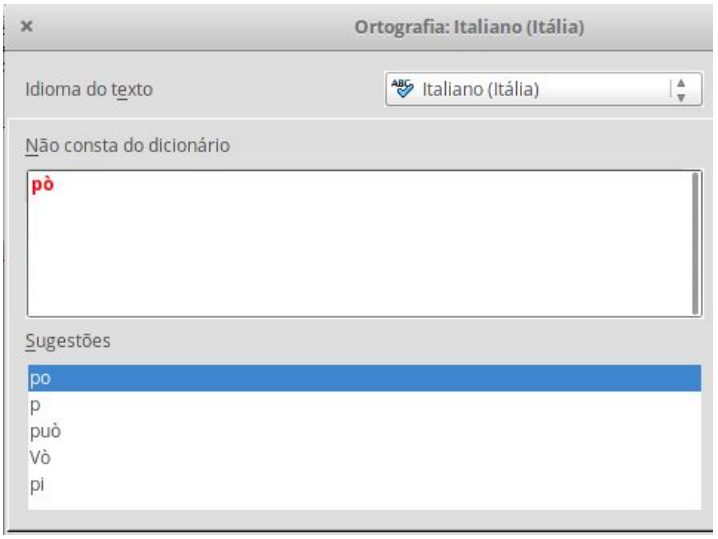


Imagem 4 - Sugestões para a palavra *piu'* do LO Writer

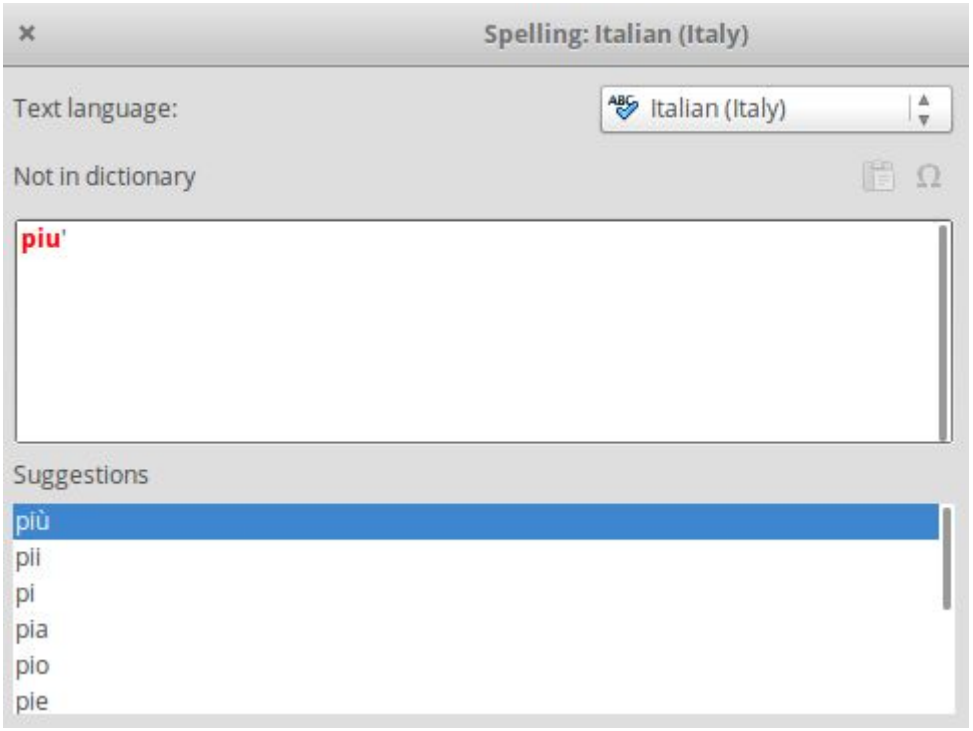


Imagem 5 - Sugestões para a palavra *piu'* reordenadas utilizando distância Levenshtein

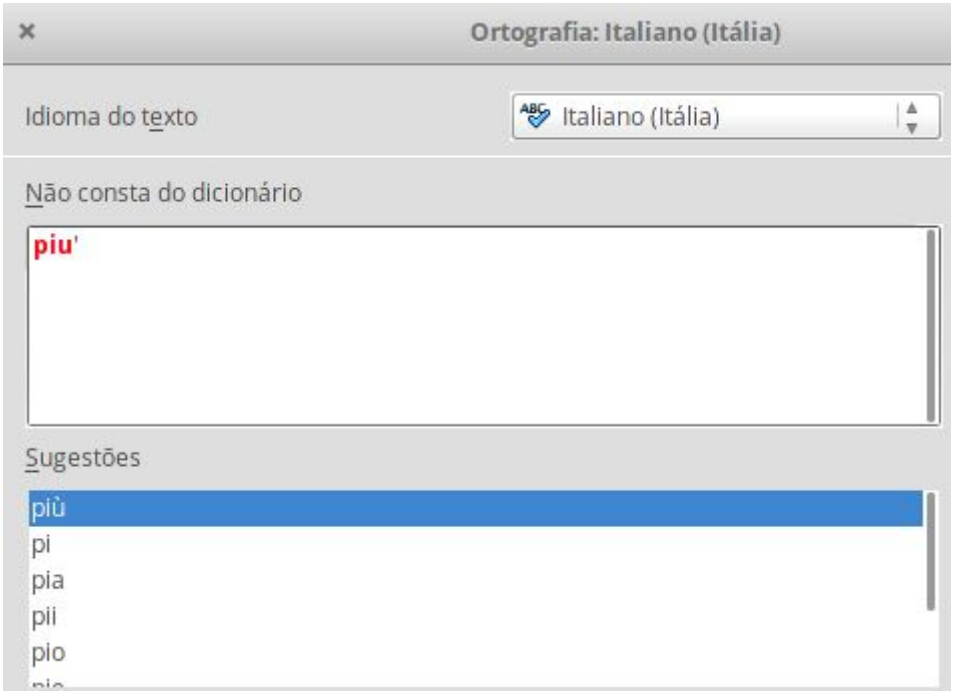


Imagem 6 - Sugestões para a palavra *citta'* do LO Writer

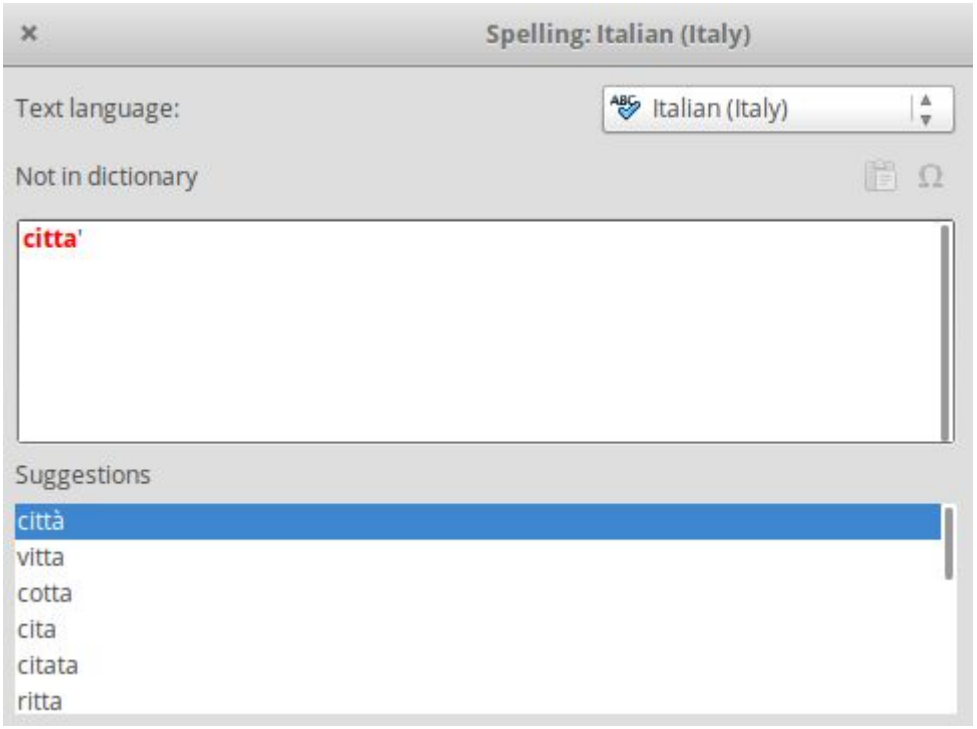


Imagem 7 - Sugestões para a palavra *citta'* reordenadas utilizando distância Levenshtein

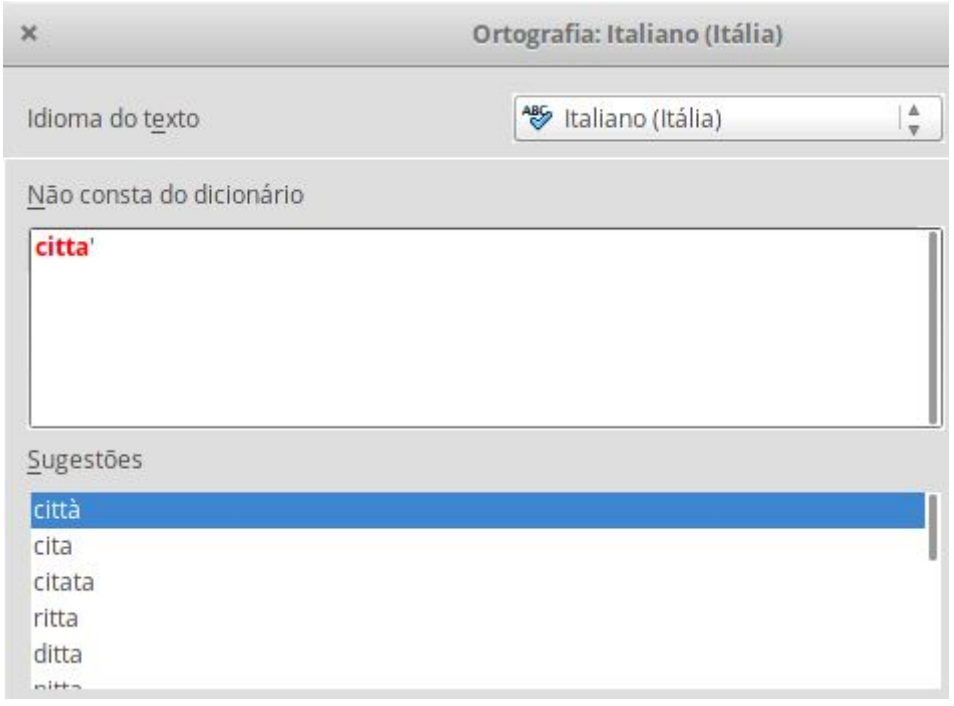


Imagem 8 - Sugestões para a palavra *c'e'* do LO Writer

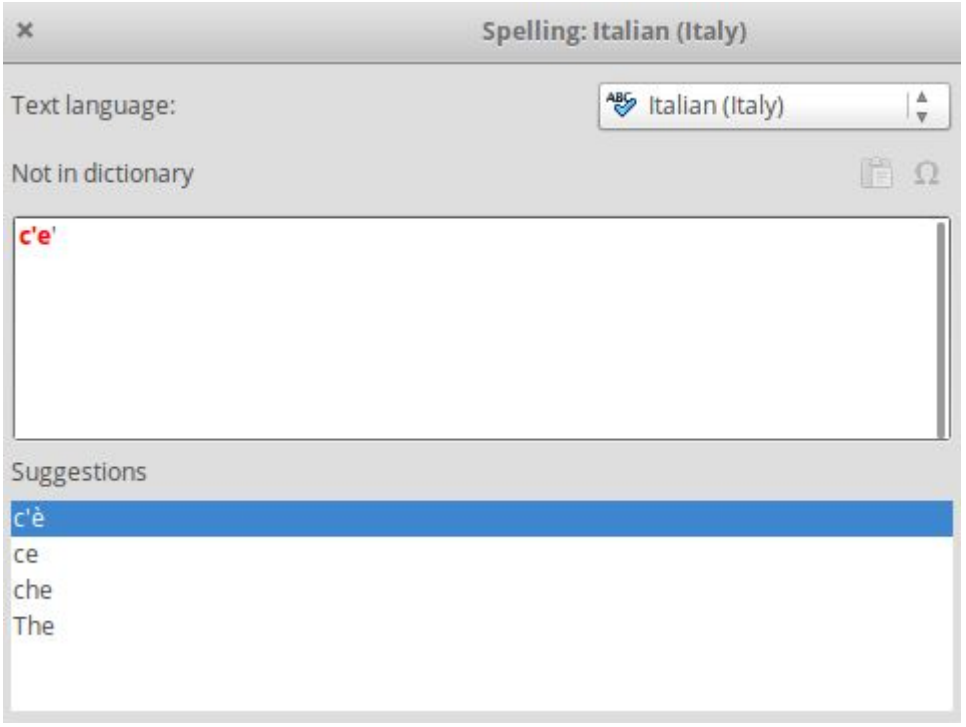
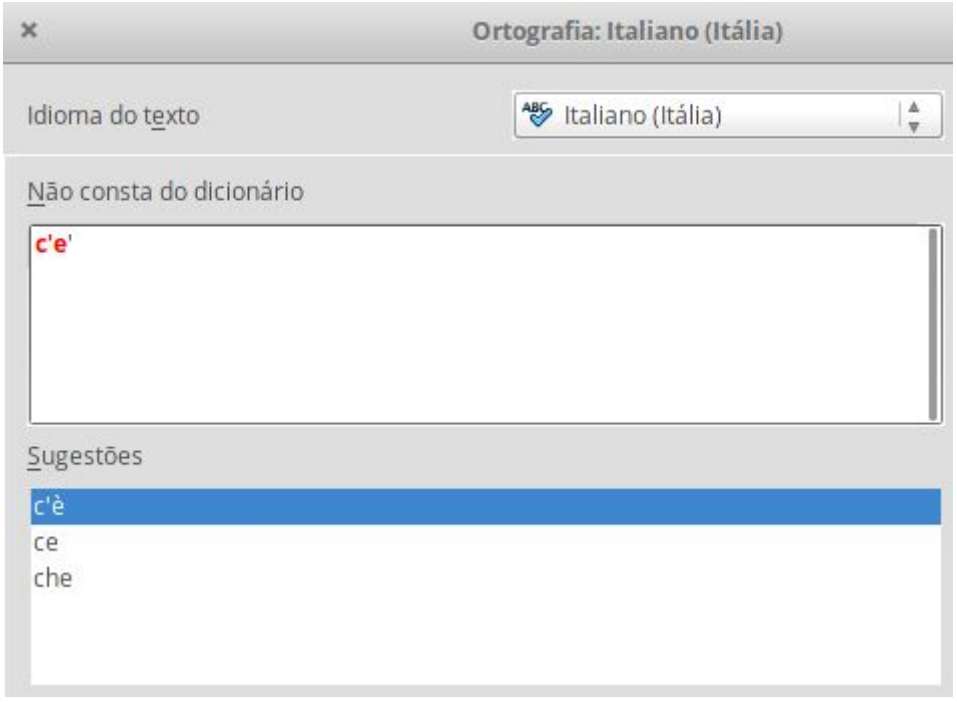


Imagem 9 - Sugestões para a palavra *c'e'* reordenadas utilizando distância Levenshtein



Podemos atribuir os resultados das reordenações à medida utilizada como fator de reordenação da lista. Ambas implementações de medidas de distância de edição utilizadas no corretor, Levensthein e Damerau-Levensthein.

A reordenação com a utilização de probabilidade bayesiana, assim como análise de trigramas, não foram completadas neste trabalho devido à complexidade e tamanho do corpus. Trabalhos futuros podem aplicar tais medidas com pré processamento do corpus para padronizar os tamanhos dos textos conforme apresentado no corpus GIMENES, ROMAN e CARVALHO (2015), simplificando a análise e o tempo computacional requerido.

Conclusão

Com a utilização de distâncias de edição como medida para ordenação de sugestões de corretor ortográfico de idiomas com utilização de diacríticos não aumentam a acurácia das sugestões. Tal resultado pode ser atribuído a simplicidade de tais medidas quando comparadas ao corretor original que possui melhores implementações em seu sistema de sugestões.

Além disso, pode-se sugerir que a utilização de um corpus menor e com melhor processamento de texto para uma maior padronização dos diálogos apresentariam uma maior acurácia nas sugestões e flexibilidade nas medidas para ordenação, como um modelo bayesiano de probabilidade para as sugestões de correção.

Por fim, este trabalho abre outros pontos a serem explorados por futuros estudos, sendo o principal deles a modelagem de um *dataset* para aplicações de inteligência artificial, utilizando os corpora obtidos pelos *web crawlers*.

Além da aplicação de um modelo bayesiano com análise de trigramas como fator de ordenação na lista de sugestões.

Referências Bibliográficas

BERKEL, B.; SMEDT, K. Triphone analysis: a combined method for the correction of orthographical and typographical errors. In: Annual meeting of the association for computational linguistics (ACL), 30, 1988, **Austin. Proceedings of the 30th annual meeting of the association for computational linguistics (ACL)**, Austin, 1988, p. 77 –83.

DAMERAU, F. J. A technique for computer detection and correction of spelling errors. **Communications of the ACM**, v. 7, n. 3, p. 171-176, 1964.

DEOROWICZ, S.; CIURA, M. G. Correcting spelling errors by modeling their causes. **Internacional Journal of Applied Mathematics and Computer Science**, v. 15, n. 2, p. 275-285, 2005.

GIMENES, P.; ROMAN, N. T.; CARVALHO, A. M. B. R. Spelling Error Patterns in Brazilian Portuguese. **Computational Linguistics**, v. 41, n. 1, p. 175–183. 2015

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2Nd Ed. New Jersey: Prentice-Hall, 2009.

KUKICH, K. Techniques for automatically correcting words in text. **ACM Computing Surveys**, v. 24, n. 4, p. 377-439, 1992.

MAGILL, R. A. Aprendizagem Motora: conceitos e aplicações. São Paulo: Edgard Blücher, 2000.

MARIAN, J. Map of languages and language families of Europe, 2017. Disponível em: <https://jakubmarian.com/map-of-languages-and-language-families-of-europe/>. Acesso em: Acesso em: 30 abr. 2018.

MAYS, E.; DAMERAU, F. J.; MERCER, R. L. Context based spelling correction.

Information Processing & Management, v.27, n.5, p.517 – 522, 1991.

MEDEIROS, J. C. **Processamento Morfológico e Correcção Ortográfica do Português**. Dissertacao (Mestrado em Engenharia Eletrotecnica e de Computadores) - InstitutoSuperior Tecnico, Universidade Tecnica de Lisboa, Lisboa, 1995.

NOYES, J. The QWERTY keyboard: a review. **International Journal of Man-Machine Studies**. v. 18, n. 3, p. 265-281. 1983.

PETERSON, J. L. Computer programs for detecting and correcting spelling errors. **Communications of the ACM**, v. 23, n. 12, p. 676 - 687, 1980.

PETERSON, J. L. A note on undetected typing errors. **Communications of the ACM**, v.29, n. 7, p. 633 - 637, 1986.

WAGNER, O. M.; YANNOU, B; SKEHL, S.; FEILLET, D.; EGGERS, J. Ergonomic Modelling and Optimization of the Keyboard Arrangement with an Ant Colony Algorithm. **Journal of Engineering Design – Taylor & Francis.** v. 14, n. 2, pp.187-208. 2003.

Diacritic. In: WIKIPÉDIA: the free encyclopedia. Wikimedia, 2018. Disponível em: <https://en.wikipedia.org/wiki/Diacritic>. Acesso em: 30 abr. 2018.

WENZEL, K.; SHARKEY, M. Windows Keyboard Layouts, 2017. Disponível em: <https://docs.microsoft.com/pt-br/globalization/windows-keyboard-layouts>. Acesso em: 30 abr. 2018.

APÊNDICE A – Lista de Países

Notas:

Ordenada por score. O score é a soma do número de sinais gráficos utilizados e do número de caracteres que utilizam diacríticos.

1) República Eslovaca

1.1 Sinais gráficos utilizados: 4

Acento agudo (´);

Caron (ˇ);

Anel (°);

Corno (’).

1.2 Caracteres que utilizam sinais gráficos: 14

a(2), c, d, e, i, l(2), n, o(2), r, s, t, u, y, z.

1.3 Score: 18

2) Tcheca

2.1 Sinais gráficos utilizados: 3

Acento agudo (´);

Caron (ˇ);

Anel (°);

2.2 Caracteres que utilizam sinais gráficos: 13

a, c, d, e(2), i, n, o, r, s, t, u(2), y, z.

2.3 Score: 16

3) República da Letônia

3.1 Sinais gráficos utilizados: 3

Acento mácron (¯);

Caron (ˇ);

Cedilha (¸).

3.2 Caracteres que utilizam sinais gráficos: 11

a, c, e, g, i, k, l, n, s, u, z.

3.3 Score: 14

4) Portugal

4.1 Sinais gráficos utilizados: 6

Acento grave (`);

Acento agudo (´);

Acento circunflexo (^);

Trema (¨);

Til (~);

Cedilha (¸).

4.2 Caracteres que utilizam sinais gráficos: 6

a(4), c, e(2), i, o(3), u.

4.3 Score: 12

5) França

5.1 Sinais gráficos utilizados: 6

Acento grave (`);

Acento agudo (´);

Acento circunflexo (^);

Trema (¨);

Til (~);

Cedilha (¸).

5.2 Caracteres que utilizam sinais gráficos: 6

a(2), c, e(4), i(2), o, u(3).

5.3 Score: 12

6) República da Lituânia

6.1 Sinais gráficos utilizados: 4

Ponto mediano (·);

Gancho polaco (˛);

Caron (ˇ);

Acento mácron (¯).

6.2 Caracteres que utilizam sinais gráficos: 7

a, c, e(2), i, s, u(2), z.

6.3 Score: 11

7) República da Polônia

7.1 Sinais gráficos utilizados: 3

Gancho polaco (˛);

Acento agudo (´);

Ponto mediano (·).

7.2 Caracteres que utilizam sinais gráficos: 7

a, c, e, n, o, s, z(2).

7.3 Score: 10

8) Espanha

8.1 Sinais gráficos utilizados: 3

Acento agudo (´);

Trema (¨);

Til (~).

8.2 Caracteres que utilizam sinais gráficos: 6

a, e, i, n, o, u(2).

8.3. Score: 9

9) Hungria

9.1 Sinais gráficos utilizados: 3

Acento agudo (´);

Trema (¨);

Acento agudo duplo (˝).

9.2 Caracteres que utilizam sinais gráficos: 5

a, e, i , o(3), u(3).

9.3 Score: 8

10) República da Turquia

10.1 Sinais gráficos utilizados: 3

Trema (¨);

Acento braquia (˘);

Cedilha (¸);

10.2 Caracteres que utilizam sinais gráficos: 5

c, g, o, s, u.

10.3 Score: 8

11) República da Estônia

11.1 Sinais gráficos utilizados: 3

Trema (¨);

Til (~);

Caron (ˇ).

11.2 Caracteres que utilizam sinais gráficos: 5

a, o(2), s, u, z.

11.3 Score: 8

12) Islândia

12.1 Sinais gráficos utilizados: 2

Acento agudo (´);

Trema (¨).

12.2 Caracteres que utilizam sinais gráficos: 6

a, e, i, o(2), u, y.

12.3 Score: 8

13) Romênia

13.1 Sinais gráficos utilizados: 3

Acento braquia (˘);

Acento circunflexo (^);

Cedilha (¸).

13.2 Caracteres que utilizam sinais gráficos: 4

a(2), i, s, t.

13.3 Score: 7

14) Itália

14.1 Sinais gráficos utilizados: 2

Acento grave (`);

Acento agudo (´);

14.2 Caracteres que utilizam sinais gráficos: 5

a, e(2), i, o, u.

14.3 Score: 7

15) República da Irlanda

15.1 Sinais gráficos utilizados: 1

Acento agudo (´).

15.2 Caracteres que utilizam sinais gráficos: 5

a, e, i, o, u.

15.3. Score: 6

16) Países Baixos (Holanda)

16.1 Sinais gráficos utilizados: 1

T**rema (¨);**

16.2 Caracteres que utilizam sinais gráficos: 5

a, e, i, o, u.

16.3 Score: 6

17) República da Eslovênia

17.1 Sinais gráficos utilizados: 2

Caron (ˇ);

Acento agudo (´);

17.2 Caracteres que utilizam sinais gráficos: 3

c, s, z.

17.3 Score: 5

18) República da Sérvia - República da Croácia 18.1 Sinais gráficos utilizados: 2

Caron (ˇ);

Acento agudo (´).

18.2 Caracteres que utilizam sinais gráficos: 3

c(2), s, z.

18.3 Score: 5

19) República da Albânia

19.1 Sinais gráficos utilizados: 2

Trema (¨);

Cedilha (¸).

19.2 Caracteres que utilizam sinais gráficos: 2

c, e.

19.3 Score: 4

20) Suécia

20.1 Sinais gráficos utilizados: 2

Anel (°);

Trema (¨);

20.2 Caracteres que utilizam sinais gráficos: 2

a(2), o.

20.3 Score: 4

21) Grão-Ducado de Luxemburgo 21.1 Sinais gráficos utilizados: 2

Trema (¨);

Acento agudo (´).

21.2 Caracteres que utilizam sinais gráficos: 2

a, e(2).

21.3 Score: 4

22) República da Finlândia

22.1 Sinais gráficos utilizados: 2

Trema (¨);

Anel (°);

22.2 Caracteres que utilizam sinais gráficos: 2

a, o.

22.3 Score: 3

23) Alemanha

23.1 Sinais gráficos utilizados: 1

Trema (¨);

23.2 Caracteres que utilizam sinais gráficos: 3

a, o, u.

23.3 Score: 4

24) Reino da Noruega

24.1 Sinais gráficos utilizados: 1

Anel (°);

24.2 Caracteres que utilizam sinais gráficos: 1

a.

24.3 Score: 2

25) Reino da Dinamarca

25.1 Sinais gráficos utilizados: 1

Anel (°);

25.2 Caracteres que utilizam sinais gráficos: 1

a.

25.3. Score: 2

APÊNDICE B – Layouts de teclados

1. Francês

1.1 caracteres acentuados

Figura 1 – Lista de caracteres que utilizam diacríticos na língua francesa.

1.2 Layout do teclado

Figura 2 – Layout do teclado francês.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que cinco dos treze caracteres já estão presentes no teclado.

1.2.1 Segurando a tecla Shift

Figura 3 – Layout do teclado francês ao se apertar a tecla Shift.

1.2.2 Segurando a tecla Alt Gr

Figura 4 – Layout do teclado francês ao se apertar a tecla Alt Gr.

2. Italiano

2.1 Caracteres acentuados

Figura 5 – Lista de caracteres que utilizam diacríticos na língua italiana.

2.2 Layout do teclado

Figura 6 – Layout do teclado Italiano.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que cinco dos seis caracteres já estão presentes no teclado.

2.2.1 Segurando a tecla Shift

Figura 7 – Layout do teclado italiano ao se apertar a tecla Shift.

2.2.2 Segurando a tecla Alt Gr

Figura 8 – Layout do teclado italiano ao se apertar a tecla Alt Gr.

3. Turco

3.1 Caracteres acentuados

Figura 9 – Lista de caracteres que utilizam diacríticos na língua turca.

3.2 Layout do teclado

Figura 10 – Layout do teclado Turco.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que seis dos seis caracteres já estão presentes no teclado.

3.2.1 Segurando a tecla Shift

Figura 11 – Layout do teclado turca ao se apertar a tecla Shift.

3.2.2 Segurando a tecla Alt Gr

Figura 12 – Layout do teclado turca ao se apertar a tecla Alt Gr.

4. Tcheco

4.1 Caracteres acentuados

Figura 13 – Lista de caracteres que utilizam diacríticos na língua tcheca.

4.2 Layout do teclado

Figura 14 – Layout do teclado Tcheco.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que onze dos quinze caracteres já estão presentes no teclado.

4.2.1 Segurando a tecla Shift

Figura 15 – Layout do teclado tcheco ao se apertar a tecla Shift.

4.2.2 Segurando a tecla Alt Gr

Figura 16 – Layout do teclado tcheco ao se apertar a tecla Alt Gr.

5. Português

5.1 Caracteres acentuados

Figura 17 – Lista de caracteres que utilizam diacríticos na língua portuguesa.

5.2 Layout do teclado

Figura 18 – Layout do teclado português.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que um dos doze caracteres já está presentes no teclado.

5.2.1 Segurando a tecla Shift

Figura 19 – Layout do teclado português ao se apertar a tecla Shift.

5.2.2 Segurando a tecla Alt Gr

Figura 20 – Layout do teclado português ao se apertar a tecla Alt Gr.

6. Espanhol

6.1 Caracteres acentuados

Figura 21 – Lista de caracteres que utilizam diacríticos na língua espanhola.

6.2 Layout do teclado

Figura 22 – Layout do teclado espanhol.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que um dos sete caracteres já está presentes no teclado.

6.2.1 Segurando a tecla Shift

Figura 23 – Layout do teclado espanhol ao se apertar a tecla Shift.

6.2.2 Segurando a tecla Alt Gr

Figura 24 – Layout do teclado espanhol ao se apertar a tecla Alt Gr.

7. Alemão

7.1 Caracteres acentuados

Figura 25 – Lista de caracteres que utilizam diacríticos na língua alemã.

7.2 Layout do teclado

Figura 26 – Layout do teclado alemão.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que três dos três caracteres já está presentes no teclado.

7.2.1 Segurando a tecla Shift

Figura 27 – Layout do teclado alemão ao se apertar a tecla Shift.

7.2.2 Segurando a tecla Alt Gr

Figura 28 – Layout do teclado alemão ao se apertar a tecla Alt Gr.

8. Húngaro

8.1 Caracteres acentuados

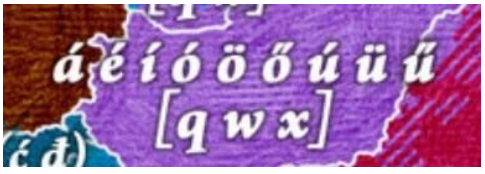


Figura 29 – Lista de caracteres que utilizam diacríticos na língua húngara.

8.2 Layout do teclado

Figura 30 – Layout do teclado húngaro.

Na figura acima estão destacados os caracteres já acentuados que estão presentes no teclado. Observa-se que nove dos nove caracteres já está presentes no teclado.

8.2.1 Segurando a tecla Shift

Figura 31 – Layout do teclado húngaro ao se apertar a tecla Shift.

8.2.2 Segurando a tecla Alt Gr

Figura 32 – Layout do teclado húngaro ao se apertar a tecla Alt Gr.

APÊNDICE C – Metodologia de correção (Francês e Turco)

Justificativas para a separação em duas metodologias

1. O corretor do francês é bom. Isso facilitou a correção e só foi necessário usar o *Google Translate* (GT) como auxílio.

2. O corretor do turco é ruim. Tive que pesquisar palavras no Google além de utilizar o GT. O processo está descrito abaixo.

3. O francês me parece uma língua mais contexto dependente do que o turco. No francês uma mesma palavra errada pode ter palavras corretas diferentes dependendo do contexto. Por exemplo a palavra errada *coute*, ela pode ter a sua forma correta como *coûte* ou *coûté* dependendo do contexto (conjugação verbal). No turco isso não acontece.

Metodologia de correção dos erros para o francês

Notas:

* Em todos os casos que isso ocorreu a sentença com a palavra sugerida foi colocada no GT para verificar se a mesma fazia sentido quando traduzida para o português, em todos os casos a sentença fazia sentido.

1. Para cada palavra errada detectada pelo corretor do *LibreOffice Writer* (*LO Writer*), clicar na palavra errada e verificar a lista de sugestões de correção.

1.1. Se na lista de sugestões aparecer apenas uma palavra e a diferença entra a palavra errada e a palavra certa for miníma (um acento, uma letra, transposição de duas letras, etc), assume-se que essa é a correção correta*.

1.2. Se na lista de sugestões aparecerem mais de uma palavra, a sentença que a mesma está inserida é copiada e colada no GT.

1.2.1. Se a sentença for muito grande pode ser que o GT traduza a frase mas não apresente nenhuma sugestão de correção. Exemplo:

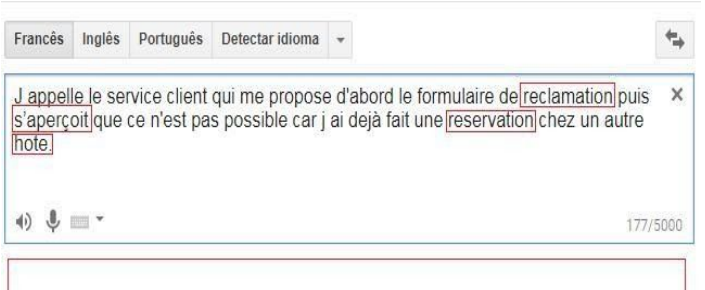


Figura 1 –

Sentença muito grande sem sugestão de correção.

1.2.1.1. Se isso ocorrer deve-se reduzir a sentença em sentenças menores para que as sugestões de correções apareçam. Exemplo:

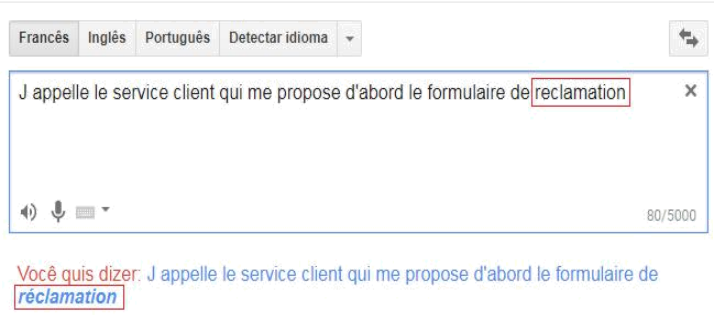


Figura 2 – Sentença menor com sugestão de correção.

1.2.1.2. Se mesmo com a sentença reduzida não houver sugestão de correção, a sentença deve ser separada para ser enviada a um nativo para correção.

1.3. Procura-se a sugestão oferecida pelo GT na lista de sugestões do corretor do *LO Writer*.

1.3.1. Caso seja encontrada e a sentença traduzida faça sentido, assume-se que essa é a correção correta.

1.3.2. Caso contrário, a sentença deve ser separada para ser enviada a um nativo para correção.

Metodologia de correção dos erros para o turco

1. Para cada palavra errada detectada pelo corretor do *LO Writer*, selecionar a sentença em que uma palavra errada está inserida e colar a mesma no GT.

2. Se o GT oferecer uma sugestão, corrigir e verificar se a tradução da sentença com a correção faz sentido.

2.1. Se a tradução da sentença com a correção fizer sentido, deve-se verificar se o corretor do *LO Writer* reconhece a palavra.

2.1.1. Se o corretor ortográfico reconhecer a palavra, assume-se que está é a palavra correta.

2.1.2. Se o corretor ortográfico não reconhecer a palavra, deve-se pesquisar a palavra errada original no Google.

2.1.2.1. Se a palavra errada for encontrada em uso no resultado da pesquisa, deve-se separar a sentença para que seja enviada a um nativo para correção. Exemplo:

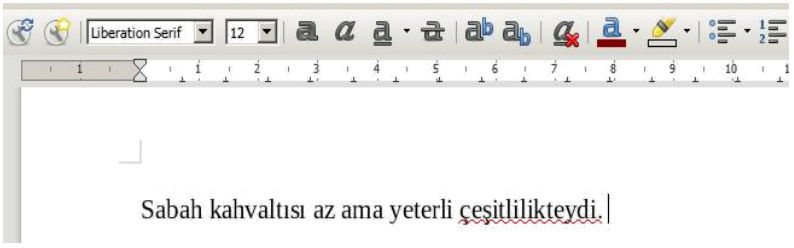


Figura 3 – O corretor identifica uma palavra como errada.

Figura 4 – O GT sugere uma correção.

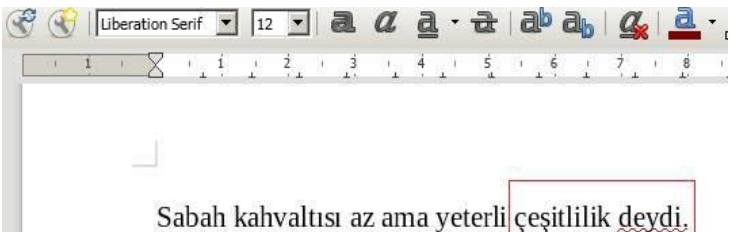


Figura 4 – O corretor não reconhece a correção.

Figura 5 – A palavra buscada aparece sendo utilizada diversas vezes.

3. Se o GT não oferecer uma sugestão de correção. Exemplo:

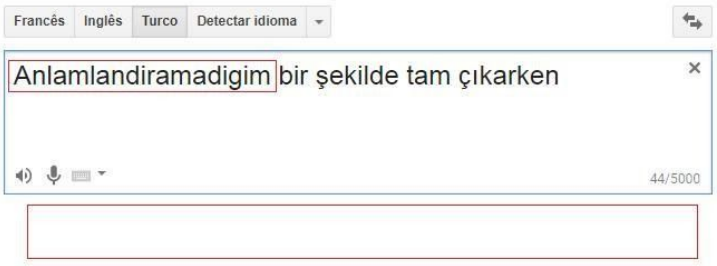


Figura 6 – Sentença sem sugestão de correção.

3.1. Selecionar a palavra errada e pesquisá-la no Google. Exemplo:

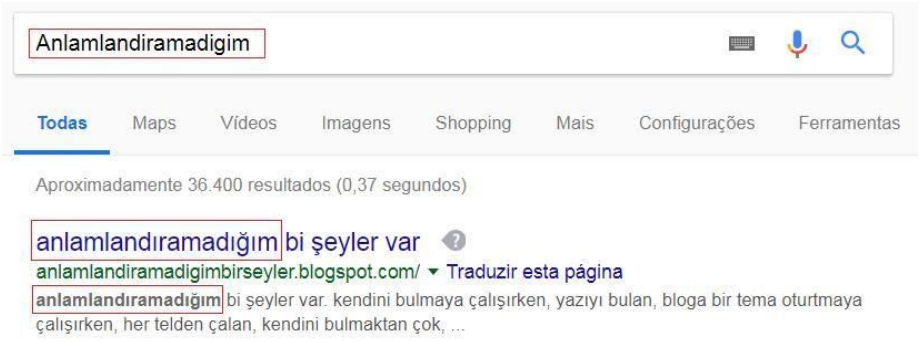


Figura 7 – Palavra errada pesquisada no Google.

3.1.1. Se alguma palavra semelhante for encontrada a mesma deve ser colocada na sentença e deve-se verificar se a tradução faz sentido.

3.1.1.1. Se o corretor ortográfico do *LO Writer* reconhecer a palavra e a tradução da sentença fizer sentido, assume-se que está é a palavra correta.

3.1.1.2. Caso contrário, a sentença deve ser separada para ser enviada a um nativo para correção.

3.1.2. Se a palavra pesquisada for idêntica às palavras encontradas nos resultados das pesquisas, a sentença deve ser separada para ser enviada a um nativo para correção.

APÊNDICE D – Classificação

Notas*:

1. Exemplos em francês.
2. Em vermelho está destacado o que está errado e em verde o que é correto.

Classificação dos erros

1. **Inserção:** Inserção de um ou mais caracteres em uma palavra. Exemplo:

    1. Palavra errada: immédiatemement
    2. Palavra certa: immédiatement

2. **Omissão:** Omissão de um ou mais caracteres em uma palavra. Exemplo:

    1. Palavra errada: responsabiité
    2. Palavra certa: responsabilité

3. **Transposição:** Troca de posição entre dois caracteres em uma palavra. Exemplo:

    1. Palavra errada: Attnetion
    2. Palavra certa: Attention

4. **Substituição:** Substituição de um caractere por outro em uma palavra. Exemplo:

    1. Palavra errada: examplaire
    2. Palavra certa: exemplaire

5. **Erros de acentuação**
    1. **Falta de acento:** Ausência de acento em um ou mais caracteres que deveriam estar acentuados em uma palavra. Exemplo:
        1. Palavra errada: deja
        2. Palavra certa: déjà

    2. **Colocação desnecessária:** Adição de acento em um ou mais caracteres que não deveriam estar acentuados em uma palavra. Exemplo:
        1. Palavra errada: élévés
        2. Palavra certa: élevés

    3. **Acento errado:** Substituição de um acento por outro em uma palavra. Exemplo:

        1. Palavra errada: complétement
        2. Palavra certa: complètement

    4. **Lugar errado:** Adição de um acento correto em um caractere errado em uma palavra. Exemplo:
        1. Palavra errada: côut
        2. Palavra certa: coût

    5. **Falta de cedilha:** Ausência de cedilha em um ou mais caracteres que deveriam utilizá-lo em uma palavra. Exemplo:
        1. Palavra errada: tronconneuse
        2. Palavra certa: tronçonneuse

6. **Erros envolvendo espaço**

1. **Falta de espaço:** Ausência de espaço entre duas ou mais palavras. Exemplo:

    1. Palavra errada: etvous
    2. Palavra certa: et vous

2. **Substituição por espaço:** Substituição de um caractere por um espaço em uma palavra. Exemplo:
    1. Palavra errada: week end
    2. Palavra certa: week-end

3. **Inserção de espaço:** Adição de espaço desnecessário no meio de uma palavra. Exemplo:
    1. Palavra errada: ré ponse
    2. Palavra certa: réponse

4. **Transposição de espaço:** Troca de posição entre um espaço e um caractere entre duas palavras. Exemplo:
    1. Palavra errada: vo sréponses
    2. Palavra certa: vos réponses

7. **Outros:** Principalmente erros com substantivos próprios com primeira letra minúscula. Exemplo:

    1. Palavra errada: europe
    2. Palavra certa: Europe

8. **Múltiplos erros:** Palavras que contenham mais de um tipo de erro. Exemplo (falta de acento + acento errado):
    1. Palavra errada: cotè

    2. Palavra certa: côté

APÊNDICE E – *Crawler* Italiano

```
import scrapy
from tcc.items import TccItem

class TripAdvisorItalianFullSpider(scrapy.Spider):
    name = 'TripadvisorFull ITA'
    allowed_domains = ['tripadvisor.it']
    start_urls = ['https://www.tripadvisor.it/ShowForum-g1-i13217-Diari_di_viaggio.html']
```

```
    # 1. get the first page
    # 2. get all topic links
    # 3. go for the first topic responses
    # 4. parse the posts
    # 5. get the next page
    # 6. go to 2 or end
    # 7. end

    def parse(self, response):
        # get the first main page links
        list_link = response.css("table.topics tr td b a::attr(href)").extract()
        for topic_link in list_link:
            # for each link go to the posts response
            print(topic_link)
            yield response.follow(topic_link, self.parse_responses_page)

        # get the second main page and beyond
        next_main_page = response.css("a.sprite-pageNext::attr(href)").extract_first()
        yield response.follow(next_main_page, self.parse)

    def parse_responses_page(self, reponsesPage):
        for response in reponsesPage.css("div.postcontent"):
            title = "".join(response.css("div.postTitle ::text").extract())
            text = "".join(response.css("div.postBody ::text").extract())
            date = response.css("div.postDate::text").extract_first()
            source = 'tripadvisorita'
            link = reponsesPage.url

            # before run change the name of the output file in pipelines.py
            parsed = TccItem(title=title, text=text, date=date, source=source, link=link)
            next_page = response.css("a.sprite-pageNext::attr(href)").extract_first()

            # inside a response page navigate
            if next_page is not None:
                yield response.follow(next_page, self.parse_responses_page)

            yield parsed
```

APÊNDICE F – *Crawler* Francês

```
import scrapy
from tcc.items import TccItem

class TripAdvisorItalianFullSpider(scrapy.Spider):
    name = 'TripadvisorFull FRA'
    allowed_domains = ['tripadvisor.fr']
    start_urls = ['https://www.tripadvisor.fr/ShowForum-g1-i11064-General_Discussion.html']

    # 1. get the first page
    # 2. get all topic links
    # 3. go for the first topic responses
    # 4. parse the posts
    # 5. get the next page
    # 6. go to 2 or end
```

```
    # 7. end

    def parse(self, response):
        # get the first main page links
        list_link = response.css("table.topics tr td b a::attr(href)").extract()
        for topic_link in list_link:
            # for each link go to the posts response
            print(topic_link)
            yield response.follow(topic_link, self.parse_responses_page)

        # get the second main page and beyond
        next_main_page = response.css("a.sprite-pageNext::attr(href)").extract_first()
        yield response.follow(next_main_page, self.parse)

    def parse_responses_page(self, reponsesPage):
        for response in reponsesPage.css("div.postcontent"):
            title = "".join(response.css("div.postTitle ::text").extract())
            text = "".join(response.css("div.postBody ::text").extract())
            date = response.css("div.postDate::text").extract_first()
            source = 'tripadvisor_french'
            link = reponsesPage.url

            # before run change the name of the output file in pipelines.py
            parsed = TccItem(title=title, text=text, date=date, source=source, link=link)
            next_page = response.css("a.sprite-pageNext::attr(href)").extract_first()

            # inside a response page navigate
            if next_page is not None:
                yield response.follow(next_page, self.parse_responses_page)

            yield parsed
```

APÊNDICE G – *Crawler* Turco

```
'use strict';

const Nightmare = require('nightmare');
const fs = require('fs');

const nightmare = Nightmare();

// Filters an array return the uniq values
const uniq = require('lodash/uniq');

// Makes an array 1-d [[1,2,3],[[4,5],6]] to [1,2,3,4,5,6]
const flattenDeep = require('lodash/flattenDeep');

// Grab all links from homepage cards
async function fetchHomePageLinks(url) {

        const links = await nightmare.goto(url)
            .scrollTo(999999999, 0)
            .wait(1000)
```

```
                .evaluate(() => new Array(...document.querySelectorAll('li.item a.ui_link')).map(el => [
el.href ]))
                .catch(err => (console.log(err), []));

        return uniq(flattenDeep(links));
}

// Grab the post links from recommendation page
async function fetchFilterPageLinks(url) {

        const links = await nightmare.goto(url)
                .scrollTo(999999999, 0)
                .wait(1000)
                .evaluate(() => new Array(...document.querySelectorAll('a.property_title')).map(el => [
el.href ]))
                .catch(err => (console.log(err), []));

        return uniq(flattenDeep(links));
}

// Grab the texts from the a review post page
async function fetchPagePosts(url) {

        return await nightmare.goto(url)
                .wait('body')
                // clicking on more
                .click('div.review-container p.partial_entry span.ulBlueLinks:first-child')
                .scrollTo(999999999, 0)
                // waiting the content load after clicking
                .wait(500)
                .evaluate(() => new Array(...document.querySelectorAll('p.partial_entry')).map(el => ({
                        title: document.title, text: el.innerText, link: document.URL, source:
'tripadvisortr'
                })))
                // .catch(err => (console.log(err), []));
                .catch(err => []);
}

/* Steps
*  1. Grab the links on homepage cards
*  2. The card links go for a list of recommended reviews
*  3. Grab the list of review post pages
*  4. Go for the review post page and grab the posts
*/
async function crawler(baseURL) {

        console.log('getting the homepage links...');

        const t0 = Date.now();
        const links = await fetchHomePageLinks(baseURL);
        let DATASET = [];
        let PAGES = [];
        console.log(`Got ${links.length} links, now grab the pages`);

        for (let i = 0; i < links.length; i++) {
                const url = links[i];

                const pages = await fetchFilterPageLinks(url);
```

```
            PAGES.push(...pages);
            console.log(`Parsed ${i+1} of ${links.length}`);
            console.log(`${PAGES.length} Pages so far...`);
        }

        for (let j = 0; j < PAGES.length; j++) {
            const url = PAGES[j];
            const data = await fetchPagePosts(url);

            DATASET.push(...data);
            console.log(`Parsing ${j+1} of ${PAGES.length}`);
        }

        console.log('TOTAL PAGES TO BE CRAWLED', PAGES.length);


        fs.appendFile('turkish.txt', JSON.stringify(DATASET, null, 1), (err) => {
            if (err) throw err;
            const t1 = Date.now();
            console.log(`${DATASET.length} posts`);
            console.log(`${PAGES.length} pages`);
            console.log(`Total time: ${((t1 - t0) / 60 / 1000)} minutes`);
        });

        await nightmare.end();
        console.log('DONE');
        process.exit();

}

crawler('https://www.tripadvisor.com.tr');
```

APÊNDICE I – *Crawler* Húngaro

```
'use strict';

const Nightmare = require('nightmare');
const fs = require('fs');

const nightmare = Nightmare();

// Filters an array return the uniq values
const uniq = require('lodash/uniq');

// Makes an array 1-d [[1,2,3],[[4,5],6]] to [1,2,3,4,5,6]
const flattenDeep = require('lodash/flattenDeep');

// Grab all links from homepage cards
async function fetchHomePageLinks(url) {

        const links = await nightmare.goto(url)
            .scrollTo(999999999, 0)
            .wait(1000)
```

```
                .evaluate(() => new Array(...document.querySelectorAll('li.item a.ui_link')).map(el => [
el.href ]))
                .catch(err => (console.log(err), []));

        return uniq(flattenDeep(links));
}

// Grab the post links from recommendation page
async function fetchFilterPageLinks(url) {

        const links = await nightmare.goto(url)
                .scrollTo(999999999, 0)
                .wait(1000)
                .evaluate(() => new Array(...document.querySelectorAll('a.property_title')).map(el => [
el.href ]))
                .catch(err => (console.log(err), []));

        return uniq(flattenDeep(links));
}

// Grab the texts from the a review post page
async function fetchPagePosts(url) {

        return await nightmare.goto(url)
                .wait('body')
                // clicking on more
                .click('div.review-container p.partial_entry span.ulBlueLinks:first-child')
                .scrollTo(999999999, 0)
                // waiting the content load after clicking
                .wait(500)
                .evaluate(() => new Array(...document.querySelectorAll('p.partial_entry')).map(el => ({
                        title: document.title, text: el.innerText, link: document.URL, source:
'tripadvisorhu'
                })))
                // .catch(err => (console.log(err), []));
                .catch(err => []);
}

/* Steps
*  1. Grab the links on homepage cards
*  2. The card links go for a list of recommended reviews
*  3. Grab the list of review post pages
*  4. Go for the review post page and grab the posts
*/
async function crawler(baseURL) {
        console.log('getting the homepage links...');

        const t0 = Date.now();
        const links = await fetchHomePageLinks(baseURL);
        let DATASET = [];
        let PAGES = [];
        console.log(`Got ${links.length} links, now grab the pages`);

        for (let i = 0; i < links.length; i++) {
                const url = links[i];

                const pages = await fetchFilterPageLinks(url);
```

```
            PAGES.push(...pages);
            console.log(`Parsed ${i+1} of ${links.length}`);
            console.log(`${PAGES.length} Pages so far...`);
        }

        for (let j = 0; j < PAGES.length; j++) {
            const url = PAGES[j];
            const data = await fetchPagePosts(url);

            DATASET.push(...data);
            console.log(`Parsing ${j+1} of ${PAGES.length}`);
        }

        console.log('TOTAL PAGES TO BE CRAWLED', PAGES.length);


        fs.appendFile('hungarian.txt', JSON.stringify(DATASET, null, 1), (err) => {
            if (err) throw err;
            const t1 = Date.now();
            console.log(`${DATASET.length} posts`);
            console.log(`${PAGES.length} pages`);
            console.log(`Total time: ${((t1 - t0) / 60 / 1000)} minutes`);
        });

        await nightmare.end();
        console.log('DONE');
        process.exit();

}

crawler('https://www.tripadvisor.co.hu');
```

APÊNDICE J – Código fonte *CorretorOrtografico.java*

```
package com.example;

import com.sun.star.lang.Locale;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lib.uno.helper.Factory;
import com.sun.star.lang.XSingleComponentFactory;
import com.sun.star.registry.XRegistryKey;
import com.sun.star.lib.uno.helper.ComponentBase;
import com.sun.star.linguistic2.SpellFailure;
import com.sun.star.linguistic2.XSpellAlternatives;
import com.sun.star.beans.PropertyValue;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import java.nio.file.Files;
```

```java
import java.util.Collections;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Scanner;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Stream;
import java.util.Arrays;

// Exceptions
import com.sun.star.lang.IllegalArgumentException;
import java.util.ArrayList;
import java.util.List;

import com.sun.star.linguistic2.XSpellAlternatives;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.UnsupportedEncodingException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;

public final class CorretorOrtografico extends ComponentBase
      implements com.sun.star.lang.XInitialization,
      com.sun.star.lang.XServiceInfo,
      com.sun.star.lang.XServiceDisplayName,
      com.sun.star.linguistic2.XLinguServiceEventBroadcaster,
      com.sun.star.linguistic2.XSpellChecker {

   private final XComponentContext m_xContext;
   private static final String m_implementationName
         = CorretorOrtografico.class.getName();
   private static final String[] m_serviceNames = {
      "com.sun.star.linguistic2.SpellChecker"};

   /**
    *  ERROR PROBABILITIES SETTINGS
    *  Replace the probabilities after classifying errors
    *
    *  Tipo de Erro          Total de Ocorrências     Porcentagem
    *  Acento Errado               6220      94.16%
    *  Colocação Desnecessária 42        0.64%
    *  Falta de Acento             30        0.45%
    *  Falta de Espaço              17        0.26%
    *  Inserção                   134         2.03%
```

```
 *  Múltiplos Erros             116         1.76%
 *  Omissão                   320.48%
 *  Substituição               11          0.17%
 *  Transposição               4           0.06%
 *  Total       6606    100.00%
 */

private double WRONG_ACCENT_PROBABILITY = 0.9416;
private double UNECESSARY_CHAR_PROBABILITY = 0.064;
private double ACCENT_OMISSION_PROBABILITY = 0.045;
private double SPACE_OMISSION_PROBABILITY = 0.026;
private double CHAR_INSERTION_PROBABILITY = 0.203;
private double MULTIPLE_ERRORS_PROBABILITY = 0.176;
private double CHAR_OMISSION_PROBABILITY = 0.048;
private double CHAR_SUBSTITUITION_PROBABILITY = 0.017;
private double CHAR_TRANSPOSITION_PROBABILITY = 0.006;

public CorretorOrtografico(XComponentContext context) {
   // A method created for calculate the suggestions performance
   //calculatePerformance();

   // Inital calculations for a trigram analysis
   //trainNaiveBayes();
   m_xContext = context;
}

public static XSingleComponentFactory __getComponentFactory(String sImplementationName) {
   XSingleComponentFactory xFactory = null;
   if (sImplementationName.equals(m_implementationName)) {
      xFactory
            = Factory.createComponentFactory(CorretorOrtografico.class, m_serviceNames);
   }
   return xFactory;
}

public static boolean __writeRegistryServiceInfo(XRegistryKey xRegistryKey) {
   return Factory.writeRegistryServiceInfo(m_implementationName,
         m_serviceNames,
         xRegistryKey);
}

// com.sun.star.lang.XInitialization:
public void initialize(Object[] aArguments) throws com.sun.star.uno.Exception {
}
// com.sun.star.lang.XServiceInfo:

public String getImplementationName() {
   return m_implementationName;
}

public boolean supportsService(String sService) {
```

```
    int len = m_serviceNames.length;
    for (int i = 0; i < len; i++) {
        if (sService.equals(m_serviceNames[i])) {
            return true;
        }
    }
    return false;
}

public String[] getSupportedServiceNames() {
    return m_serviceNames;
}
// com.sun.star.lang.XServiceDisplayName:

public String getServiceDisplayName(com.sun.star.lang.Locale aLocale) {
    return new String();
}
// com.sun.star.linguistic2.XLinguServiceEventBroadcaster:

public boolean
        addLinguServiceEventListener(com.sun.star.linguistic2.XLinguServiceEventListener xLstnr) {
    return false;
}

public boolean hasLocale(com.sun.star.lang.Locale aLocale) {
    boolean bRes = false;
    if (IsEqual(aLocale, new Locale("it", "IT", ""))) {
        bRes = true;
    }
    return bRes;
}
// com.sun.star.linguistic2.XSpellChecker:

public boolean isValid(String aWord, com.sun.star.lang.Locale aLocale,
        com.sun.star.beans.PropertyValue[] aProperties) throws
        com.sun.star.lang.IllegalArgumentException {
    if (IsEqual(aLocale, new Locale()) || aWord.length() == 0) {
        return true;
    }

    if (!hasLocale(aLocale)) {
        return true;
    }
    return GetSpellFailure(aWord,aLocale, aProperties) == -1;
}

private short GetSpellFailure(
    String aWord,
    Locale aLocale,
    PropertyValue[] aProperties )
{
```

```
        short nRes = -1;

        try {
            if
(Hunspell.getInstance().getDictionary("/home/rgomes/dictionaries/dictionaries/it/index").misspelled(aWor
d))
                return SpellFailure.SPELLING_ERROR;
        } catch (FileNotFoundException ex) {
            return nRes;
        } catch (UnsupportedEncodingException ex) {
            return nRes;
        }

        return nRes;
    }

    private String formata(String s) {
      return s.replaceAll("\\p{P}", "").replaceAll("[0-9]","").toLowerCase();
    }
    File f1;
    Hashtable palavra1 = new Hashtable<String, Hashtable>();
    double V = 0.0;

    public void treina() throws FileNotFoundException {
        System.out.println("treinando");
        boolean existePalavraAntAnt = false;
        boolean existePalavraAnt = false;
        boolean existePalavraAtual = false;
        f1 = new File("/home/rgomes/tcc-corretor/it-treino.txt");
        Scanner sc = new Scanner(new FileInputStream(f1));
        sc.useDelimiter("\\n");
        String palavraAtual = "";
        String palavraAnt = "nulo";
        String palavraAntAnt = "nulo";
        boolean nulo = false;
        while (true) {
            try {
                int qtde = 0;
                String linha = sc.nextLine();
                //System.out.print(linha);
                if (linha.equals("================================================")) {
                    linha = sc.nextLine();
                }
                StringTokenizer st = new StringTokenizer(linha);
                while (st.hasMoreTokens()) {
                    palavraAtual = formata(st.nextToken());
                    V++;
                    if (nulo) {
                        qtde++;
                    }
                    if (nulo && qtde == 1) {
```

```
    palavraAnt = "nulo";
    palavraAntAnt = "nulo";
}
if (nulo && qtde == 2) {
    nulo = false;
    qtde = 0;
}
if (palavraAtual.endsWith(".")) {
    nulo = true;
    palavraAtual = palavraAtual.replace('.', (char) 0);
}
int posicao = 0;
Set<String> set1 = palavra1.keySet();
String str;
Iterator<String> itr1 = set1.iterator();
while (itr1.hasNext()) {
    str = itr1.next();
    Hashtable h2 = (Hashtable) palavra1.get(str);
    Set<String> set2 = h2.keySet();
    String str2;
    Iterator<String> itr2 = set2.iterator();
    while (itr2.hasNext()) {
        str2 = itr2.next();
        Hashtable h3 = (Hashtable) h2.get(str2);
        Set<String> set3 = h3.keySet();
        if (set3.contains(palavraAtual)) {
            LinkedList<Integer> list
                    = (LinkedList<Integer>) h3.get(palavraAtual);
            posicao = posicao + list.size();
        }
    }
}
existePalavraAntAnt = false;
existePalavraAnt = false;
existePalavraAtual = false;
Hashtable ht3 = new Hashtable<String, LinkedList<Integer>>();
LinkedList<Integer> listH3 = new LinkedList<Integer>();
Hashtable ht2 = new Hashtable<String, Hashtable>();
Set<String> setH1 = palavra1.keySet();
String strH1;
Iterator<String> itrH1 = setH1.iterator();
while (itrH1.hasNext()) {
    strH1 = itrH1.next();
    if (strH1.equals(palavraAntAnt)) {
        existePalavraAntAnt = true;
        ht2 = (Hashtable) palavra1.get(strH1);
        Set<String> set2 = ht2.keySet();
        String str2;
        Iterator<String> itr2 = set2.iterator();
        while (itr2.hasNext()) {
            str2 = itr2.next();
```

```
                    if (str2.equals(palavraAnt)) {
                        existePalavraAnt = true;
                        ht3 = (Hashtable) ht2.get(str2);
                        Set<String> set3 = ht3.keySet();
                        if (set3.contains(palavraAtual)) {
                            existePalavraAtual = true;
                            listH3
                                    = (LinkedList<Integer>) ht3.get(palavraAtual);
                            listH3.add(posicao + 1);
                        }
                    }
                }
            }
        }
        if (!existePalavraAntAnt) {
            LinkedList<Integer> list = new LinkedList<Integer>();
            list.add(posicao + 1);
            Hashtable h3 = new Hashtable<String, Hashtable>();
            h3.put(palavraAtual, list);
            Hashtable h2 = new Hashtable<String, Hashtable>();
            h2.put(palavraAnt, h3);
            palavra1.put(palavraAntAnt, h2);
        }
        if (!existePalavraAnt) {
            LinkedList<Integer> list = new LinkedList<Integer>();
            list.add(posicao + 1);
            Hashtable h3 = new Hashtable<String, Hashtable>();
            h3.put(palavraAtual, list);
            ht2.put(palavraAnt, h3);
        }
        if (!existePalavraAtual) {
            LinkedList<Integer> list = new LinkedList<Integer>();
            list.add(posicao + 1);
            ht3.put(palavraAtual, list);
        }
        palavraAntAnt = palavraAnt;
        palavraAnt = palavraAtual;
    }

} catch (NoSuchElementException e) {
    System.out.println("Treino finalizado");
    break;
}
}
}
double Voc = 0.0;
LinkedList<String> voc = new LinkedList<String>();

public void calculaVoc() {
Set<String> set1 = palavra1.keySet();
String str;
```

```
    Iterator<String> itr1 = set1.iterator();
    while (itr1.hasNext()) {
        str = itr1.next();
        Hashtable h2 = (Hashtable) palavra1.get(str);
        Set<String> set2 = h2.keySet();
        String str2;
        Iterator<String> itr2 = set2.iterator();
        while (itr2.hasNext()) {
            str2 = itr2.next();
            Hashtable h3 = (Hashtable) h2.get(str2);
            Set<String> set3 = h3.keySet();
            Iterator<String> itr3 = set3.iterator();
            while (itr3.hasNext()) {
                String st = itr3.next();
                if (!voc.contains(st)) {
                    voc.add(st);
                    Voc = Voc + 1;
                }
            }
        }
    }
}

File f;
Hashtable todasPalavras = new Hashtable<String, Integer>();
Hashtable corpusTeste1 = new Hashtable<String, Hashtable>();

public void corpusTeste() throws FileNotFoundException {
    boolean existePalavraAntAnt = false;
    boolean existePalavraAnt = false;
    boolean existePalavraAtual = false;
    f = new File("/home/rgomes/tcc-corretor/it-teste.txt");
    Scanner sc = new Scanner(new FileInputStream(f));
    String linha;
    String palavraAtual = "";
    String palavraAnt = "nulo";
    String palavraAntAnt = "nulo";
    boolean nulo = false;
    while (true) {
        try {
            int qtde = 0;
            linha = sc.nextLine();
            if (linha.equals("================================================")) {
                linha = sc.nextLine();
            }
            StringTokenizer st = new StringTokenizer(linha);
            while (st.hasMoreTokens()) {
                palavraAtual = formata(st.nextToken());
                String palavra = palavraAtual;
                palavra.replace('.', (char) 0);
                todasPalavras.put(palavra, 0);
```

```
if (nulo) {
    qtde++;
}
if (nulo && qtde == 1) {
    palavraAnt = "nulo";
    palavraAntAnt = "nulo";
}
if (nulo && qtde == 2) {
    nulo = false;
    qtde = 0;
}
if (palavraAtual.endsWith(".")) {
    nulo = true;
    palavraAtual = palavraAtual.replace('.', (char) 0);
}
int posicao = 0;
Set<String> set1 = corpusTeste1.keySet();
String str;
Iterator<String> itr1 = set1.iterator();
while (itr1.hasNext()) {
    str = itr1.next();
    Hashtable h2 = (Hashtable) corpusTeste1.get(str);
    Set<String> set2 = h2.keySet();
    String str2;
    Iterator<String> itr2 = set2.iterator();
    while (itr2.hasNext()) {
        str2 = itr2.next();
        Hashtable h3 = (Hashtable) h2.get(str2);
        Set<String> set3 = h3.keySet();
        if (set3.contains(palavraAtual)) {
            LinkedList<Integer> list
                    = (LinkedList<Integer>) h3.get(palavraAtual);
            posicao = posicao + list.size();
        }
    }
}
existePalavraAntAnt = false;
existePalavraAnt = false;
existePalavraAtual = false;
Hashtable ht3 = new Hashtable<String, LinkedList<Integer>>();
LinkedList<Integer> listH3 = new LinkedList<Integer>();
Hashtable ht2 = new Hashtable<String, Hashtable>();
Set<String> setH1 = corpusTeste1.keySet();
String strH1;
Iterator<String> itrH1 = setH1.iterator();
while (itrH1.hasNext()) {
    strH1 = itrH1.next();
    if (strH1.equals(palavraAntAnt)) {
        existePalavraAntAnt = true;
        ht2 = (Hashtable) corpusTeste1.get(strH1);
        Set<String> set2 = ht2.keySet();
```

```
                    String str2;
                    Iterator<String> itr2 = set2.iterator();
                    while (itr2.hasNext()) {
                        str2 = itr2.next();
                        if (str2.equals(palavraAnt)) {
                            existePalavraAnt = true;
                            ht3 = (Hashtable) ht2.get(str2);
                            Set<String> set3 = ht3.keySet();
                            if (set3.contains(palavraAtual)) {
                                existePalavraAtual = true;
                                listH3
                                        = (LinkedList<Integer>) ht3.get(palavraAtual);
                                listH3.add(posicao + 1);
                            }
                        }
                    }
                }
            }
            if (!existePalavraAntAnt) {
                LinkedList<Integer> list = new LinkedList<Integer>();
                list.add(posicao + 1);
                Hashtable h3 = new Hashtable<String, Hashtable>();
                h3.put(palavraAtual, list);
                Hashtable h2 = new Hashtable<String, Hashtable>();
                h2.put(palavraAnt, h3);
                corpusTeste1.put(palavraAntAnt, h2);
            }
            if (!existePalavraAnt) {
                LinkedList<Integer> list = new LinkedList<Integer>();
                list.add(posicao + 1);
                Hashtable h3 = new Hashtable<String, Hashtable>();
                h3.put(palavraAtual, list);
                ht2.put(palavraAnt, h3);
            }
            if (!existePalavraAtual) {
                LinkedList<Integer> list = new LinkedList<Integer>();
                list.add(posicao + 1);
                ht3.put(palavraAtual, list);
            }
            palavraAntAnt = palavraAnt;
            palavraAnt = palavraAtual;
        }
    } catch (NoSuchElementException e) {
        System.out.println("Teste finalizado");
        break;
    }
  }
}

public interface SpellErrorType {
  public static final int INSERTION = 1;
```

```
        public static final int OMISSION = 2;
        public static final int TRANSPOSITION = 3;
        public static final int SUBSTITUITION = 4;
        public static final int ACCENT_OMISSION = 5;
        public static final int ACCENT_INSERTION = 6;
        public static final int ACCENT_WRONG = 7;
        public static final int ACCENT_MISPLACED = 8;
        public static final int ACCENT_MISSING_CEDILHA = 9;
        public static final int SPACE_MISSING = 10;
        public static final int SPACE_SUBSTITUITION = 11;
        public static final int SPACE_INSERTION = 12;
        public static final int SPACE_TRANSPOSITION = 13;
        public static final int OTHERS = 14;
        public static final int MULTIPLE_ERRORS = 15;
    }

    /*
    * Calculate a naive bayes approach of the train dataset, which contains a
    * smaller subset of the posts, populating a trigram list
    * that will be used on spellchecker suggestions sorting.
    */
    private void trainNaiveBayes() {

        List<String> lines = null;
        List<String> vocabulary = new ArrayList<String>();
        Map<String, Integer> bow = new HashMap<String, Integer>();

        List<Trigrama> bowTrigrama = new ArrayList<>();

        List<Trigrama> trigramaClasse = new ArrayList<>();

        List<Trigrama> trigrama = new ArrayList<>();

        f1 = new File("/home/rgomes/tcc-corretor/it-treino.txt");
        Scanner sc = null;
        try {
            sc = new Scanner(new FileInputStream(f1));
        } catch (FileNotFoundException ex) {
            Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
        }
        sc.useDelimiter("\\n");

        while (true) {
            try {
                String linha = sc.nextLine();
                //System.out.print(linha);
                if
(linha.equals("==========================================================================
==========")) {
                    linha = sc.nextLine();
                }
```

```
        String[] tokens = linha.replaceAll("\\p{P}", "").replaceAll("[0-9]","").toLowerCase().split(" ");

        for (int i = 0; i < tokens.length; i++) {
            String token = tokens[i].trim();
             if (!"".equals(token) && !vocabulary.contains(token)) {
                vocabulary.add(token);
                bow.put(token, 1);
            } else if (!"".equals(token)) {
                bow.put(token, bow.get(token) + 1);
            }
        }

    } catch (NoSuchElementException e) {
        System.out.println("Vocabulario: " + vocabulary.size());
        break;
    }
}

System.out.println("Starting Trigrama");
for (int i = 0; i < vocabulary.size() - 2; i++) {

    System.out.print(".");
    String pA = vocabulary.get(i);
    String pB = vocabulary.get(i + 1);
    String pC = vocabulary.get(i + 2);

    Trigrama tri = new Trigrama(pA + " " + pB + " " + pC);
    trigrama.add(tri);

    if (trigramaClasse.contains(tri)) {
        int index = trigramaClasse.indexOf(tri);
        Trigrama p = trigramaClasse.get(index);
        p.setFrequencia(p.getFrequencia() + tri.getFrequencia());
    } else {
        trigramaClasse.add(tri);
    }
}
System.out.println("FinishedTrigrama");

for (Trigrama tri: trigrama) {
    System.out.println(tri.classe);
    System.out.println(tri.palavra);
    System.out.println(tri.frequencia);
}

int k = 3;
int i = 0;

while (k > 0 && i < trigramaClasse.size()) {
    Trigrama triFreq = trigramaClasse.get(i);
```

```
            if (bowTrigrama.contains(triFreq)) {
                int indexTri = bowTrigrama.indexOf(triFreq);
                Trigrama bowTri = bowTrigrama.get(indexTri);

                bowTri.setFrequencia(bowTri.getFrequencia() + triFreq.getFrequencia());
            } else {
                bowTrigrama.add(triFreq);
                k--;
            }

            i++;
        }

        System.out.println("== Gravando atributos ==");
        int p;
        for (p = 0; p < bowTrigrama.size(); p++) {
            Trigrama attTri = bowTrigrama.get(p);
            System.out.println(attTri.classe + " " + attTri.palavra + " " + attTri.frequencia);

            // Not finished

            // split
            // p1 p2 p3
            // hu.spell(p1)
            // if p1 has error
            // discovery class and sum probabilities
            // (P1P2PC + 1) / (trigramas.size() + words.size())
        }

    }


    /**
     * Performs a performance check after installing and initializing the
     * spellchecker extension
     * Starting from a list of mispelled words, it will generate the list of
     * suggestions from the spell method and save into a file
     * This can be used for comparison purposes
    */
    private void calculatePerformance() {

        FileWriter fileWriter = null;
        try {
            File f = new File("/home/rgomes/tcc/tcc/errors_final_list.txt");
            Scanner sc = null;
            try {
                sc = new Scanner(new FileInputStream(f));
            } catch (FileNotFoundException ex) {
                Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
            }
            sc.useDelimiter("\\n");
```

```
List<String> words = new ArrayList<String>();
while (sc.hasNext()) {
   String word = sc.nextLine();
   words.add(word);
}
Locale locale = new Locale("it", "IT", "");
// File to save the results
fileWriter = new FileWriter("/home/rgomes/tcc/tcc/suggestions_performance.txt", true);
BufferedWriter bufferWriter = new BufferedWriter(fileWriter);
int counter = 0;
for (String word: words) {
   counter++;
   Hunspell h = Hunspell.getInstance();
   Hunspell.Dictionary dic;
   try {

      System.out.println("Evaluating Hunspell :  " + counter + " de "+ words.size());

      dic = h.getDictionary("/home/rgomes/dictionaries/dictionaries/it/index");
      List<String> hu_alternatives = dic.suggest(word);

      try {
         String[] my_alternatives_impl = spell(word, locale, null).getAlternatives();

         List<String> my_alternatives = new ArrayList<String>();

         for (String value: my_alternatives_impl) {
            my_alternatives.add(value);
         }

         // Write hu suggestions
         /*bufferWriter.write("Hunspell,"+word+",");

         for(String hu_alternative : hu_alternatives) {
            bufferWriter.write(hu_alternative+",");
         }*/


         bufferWriter.write("####");
         bufferWriter.newLine();

         // Write my suggestions
         bufferWriter.write(word);
         bufferWriter.newLine();

         for(String my_alternative : my_alternatives) {
            bufferWriter.write(my_alternative);
            bufferWriter.newLine();
         }
```

```
                } catch (IllegalArgumentException ex) {
                    Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
                }


            } catch (FileNotFoundException ex) {
                Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
            } catch (UnsupportedEncodingException ex) {
                Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
        // Close the file writer
        bufferWriter.close();
        fileWriter.close();

    } catch (IOException ex) {
        Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            fileWriter.close();
        } catch (IOException ex) {
            Logger.getLogger(CorretorOrtografico.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}


// Method that implements the classification errors definitions
// Reference:
https://drive.google.com/file/d/189Jk6SLUep5Csw4AFdnqIMrgNPNKsJxL/view?usp=sharing
public int errorClassification(String error, String correction) {

    // italian most common case
    // handle ' case
    int lastCharCode = (int) error.charAt(error.length() - 1);
    if (
        (
            lastCharCode == 96 ||
            lastCharCode == 39 ||
            lastCharCode == 180
        ) || (lastCharCode > 224 &&  lastCharCode < 250)
      ) {
        return SpellErrorType.ACCENT_WRONG;
    }

    if (error.length() == correction.length()) {
        //List errors;

        for (int i = 0; i < correction.length(); i++) {
            char characterCorrection = correction.charAt(i);
```

```
                char characterError = error.charAt(i);
                int codeCorrection = (int) characterCorrection;
                int codeError = (int) characterError;
               if (characterCorrection != characterError) {

                  // Accent Verifications
                  if (codeCorrection > 224 && codeCorrection <= 252) {
                     if (codeError > 224 && codeError <= 252) {
                        //both accented
                        return SpellErrorType.ACCENT_WRONG;
                     } else {
                        // correction has accent but error doesnt
                        return SpellErrorType.ACCENT_OMISSION;
                     }

                   } else if (codeError > 224 && codeError <= 252) {
                         // error has accent but correction doesnt
                         return SpellErrorType.ACCENT_INSERTION;
                      } else {
                         // both not accented
                         return SpellErrorType.SUBSTITUITION;

                         // check for transposition
                         // check if is space character
                      }
                  }
               }
            }

            /*   Exemplo:
                 1. Palavra errada: immédiatememement
                 2. Palavra certa: immédiatement
             */
            if (error.length() < correction.length()) {
               return SpellErrorType.INSERTION;
            }
            /*   Exemplo:
                 1. Palavra errada: responsabiité
                 2. Palavra certa: responsabilité
            */
            if (error.length() > correction.length()) {
               return SpellErrorType.OMISSION;
            }

            return SpellErrorType.OTHERS;
         }

         public XSpellAlternatives spell(String aWord,
               com.sun.star.lang.Locale aLocale, PropertyValue[] aProperties) throws
      com.sun.star.lang.IllegalArgumentException {
```

```
/* Replace these probabilities with your dataset error distribuition */
double INSERTION = 0.203;
double OMISSION = 0.048;
double TRANSPOSITION = 0.006;
double SUBSTITUITION = 0.017;
double ACCENT_OMISSION = 0.045;
double ACCENT_INSERTION = 0.064;
double ACCENT_WRONG = 0.9416;
double ACCENT_MISPLACED = 0;
double ACCENT_MISSING_CEDILHA = 0;
double SPACE_MISSING = 0.026;
double SPACE_SUBSTITUITION = 0;
double SPACE_INSERTION = 0;
double SPACE_TRANSPOSITION = 0;
double OTHERS = 0;
double MULTIPLE_ERRORS = 0;

Hunspell h = null;
Hunspell.Dictionary dic = null;
int qtdeSug = 0;
try {
   h = Hunspell.getInstance();
   dic = h.getDictionary("/home/rgomes/dictionaries/dictionaries/it/index");
   qtdeSug = dic.suggest(aWord).size();
} catch (Exception e) {
   e.printStackTrace();
}

String[] aProposals = new String[qtdeSug];

if (qtdeSug > 0) {
   aProposals = dic.suggest(aWord).toArray(aProposals);
}

List<Palavra> listTot = new ArrayList<Palavra>();

for (int i = 0; i < qtdeSug; i++) {
   String current_suggestion = aProposals[i];

   /* Classify the error by type */
   //int type = errorClassification(aWord, current_suggestion);

   //System.out.println("error: " + aWord);
   //System.out.println("correction: " + current_suggestion);
   //System.out.println("error type: " + type);
   //System.out.println();

   //if (type == SpellErrorType.ACCENT_WRONG) prob = ACCENT_WRONG;
   //if (type == SpellErrorType.ACCENT_OMISSION) prob = ACCENT_WRONG;
   //if (type == SpellErrorType.ACCENT_INSERTION) prob = ACCENT_WRONG;
   //if (type == SpellErrorType.INSERTION) prob = INSERTION;
```

```
            //if (type == SpellErrorType.OMISSION) prob = OMISSION;
            //if (type == SpellErrorType.OMISSION) prob = OMISSION;
            //if (type == SpellErrorType.OTHERS) prob = OTHERS;

            /* Using Levensthein Distance */
            int prob = LevenstheinDistance.distance(aWord , current_suggestion);

            //add probability or distance edition of the current suggestion
            Palavra palavra_suggestion = new Palavra();
            palavra_suggestion.solucao = current_suggestion;
            palavra_suggestion.prob = prob;
            listTot.add(palavra_suggestion);

            //System.out.println(aWord + "/" + current_suggestion + "/" + prob);
        }

        /*for(Palavra temp: listTot){
            System.out.println("unordered " + temp.solucao + " : " + temp.prob);
             }*/


        /* Sorting them, after calculating probabilities or edittion distances of each suggestion */
        Collections.sort(listTot,new OrdenaLista());

        //passa sugestões para aProposals
        int cont = 0;
        Iterator it = listTot.iterator();
        for (int i = 0; i < listTot.size(); i++) {
            if (it.hasNext()) {
                Palavra p = (Palavra) it.next();
                aProposals[cont] = p.solucao;
                cont++;
            }
        }

        /*for(Palavra temp: listTot){
            System.out.println("ordered " + temp.solucao + " : " + temp.prob);
             }*/

        XSpellAlternatives xRes = null;
        if (!isValid( aWord, aLocale, aProperties ))
        {
            xRes = new XSpellAlternatives_impl(aWord, aLocale, SpellFailure.SPELLING_ERROR,
aProposals );
        }
        return xRes;
    }

    /** Another option is spellcheck the word using Naive Bayes Probabilities
     *  With this method, a trigram analysis is performed on the full dataset
     *  before defining the new suggestions
```

```
     *  - This approach does not perform well with bigger datasets
     *  - Some performance work is needed to inscrease performance on this method
     */
    public com.sun.star.linguistic2.XSpellAlternatives spellBayes(String aWord,
            com.sun.star.lang.Locale aLocale, com.sun.star.beans.PropertyValue[] aProperties) throws
com.sun.star.lang.IllegalArgumentException {

        if (palavra1.isEmpty()) {
            try {
                treina();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        if (corpusTeste1.isEmpty()) {
            try {
                corpusTeste();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        //retira pontos especiais que vem com as palavras.
        aWord = formata(aWord);
        aWord.replace('.', (char) 0);
        Set<String> set4 = todasPalavras.keySet();
        if (set4.contains(aWord)) {
            Integer qtde2 = 1 + (Integer) todasPalavras.get(aWord);
            todasPalavras.remove(aWord);
            todasPalavras.put(aWord, qtde2);
        }
        //pega o corretor ortográfico e dicionario pt-br utilizado pelo OO para ver as sugestões.
        Hunspell h = null;
        Hunspell.Dictionary dic = null;
        int qtdeSug = 0;
        try {
            h = Hunspell.getInstance();
            dic = h.getDictionary("/home/rgomes/dictionaries/dictionaries/it/index");
            qtdeSug = dic.suggest(aWord).size();
            for (int i = 0; i < qtdeSug; i++) {
                System.out.println(dic.suggest(aWord).get(i));
            }
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
        LinkedList<Palavra> list = new LinkedList<Palavra>();
        String[] aProposals = new String[qtdeSug];
        for (int i = 0; i < qtdeSug; i++) {
            aProposals[i] = "";
        }
```

```
if (qtdeSug == 0) {
    System.out.println(aWord);
    System.out.println();
    return new com.example.XSpellAlternatives_impl(aWord, aLocale,
            SpellFailure.SPELLING_ERROR, aProposals);
}
if (qtdeSug != 1) {
    String aux = "";
    char[] c = new char[aWord.length()];
    //corrige erro do tipo inserção
    aux = aWord;
    aux.toLowerCase();
    for (int i = 0; i < aWord.length(); i++) {
        Palavra palavra = new Palavra();
        palavra.prob = 0.1054;
        palavra.solucao = aux.substring(0, i) + aux.substring(i + 1);
        if (dic.suggest(aWord).contains(palavra.solucao)) {
            list.add(palavra);
        }
        aux = palavra.solucao;
        aux = aux.substring(0, 1).toUpperCase()
                + aux.substring(1).toLowerCase();
        if (dic.suggest(aWord).contains(aux)) {
            palavra.solucao = aux;
            list.add(palavra);
        }
        aux = aWord;
        aux.toLowerCase();
    }
    //corrige erros do tipo transposição
    aux = aWord;
    aux.toLowerCase();
    for (int i = 0; i < c.length - 1; i++) {
        for (int k = 0; k < aWord.length(); k++) {
            c[k] = aux.charAt(k);
        }
        Palavra palavra = new Palavra();
        palavra.prob = 0.0386;
        char auxiliar = c[i];
        c[i] = c[i + 1];
        c[i + 1] = auxiliar;
        for (int j = 0; j < c.length; j++) {
            palavra.solucao = palavra.solucao + c[j];
        }
        if (dic.suggest(aWord).contains(palavra.solucao)) {
            list.add(palavra);
        }
        aux = palavra.solucao;
        aux = aux.substring(0, 1).toUpperCase()
                + aux.substring(1).toLowerCase();
        if (dic.suggest(aWord).contains(aux)) {
```

```
            palavra.solucao = aux;
            list.add(palavra);
        }
        aux = aWord;
        aux.toLowerCase();
    }
    // corrige erros dos tipos omissao e falta de espaço

    /************************
     * OMISSAO E FALTA DE ESPACO
     *
     */
    aux = aWord;
    aux.toLowerCase();
    char c1;
    // for each word char
    for (int i = 0; i <= aWord.length(); i++) {
        // from space to û
        for (int j = 32; j < 251; j++) {
            Palavra palavra = new Palavra();
            palavra.prob = SPACE_OMISSION_PROBABILITY + CHAR_OMISSION_PROBABILITY;
            c1 = (char) j;
            if (i == 0) {
                palavra.solucao = c1 + aux;
            } else if (i == aWord.length()) {
                palavra.solucao = aux + c1;
            } else {
                palavra.solucao = aux.substring(0, i) + c1 + aux.substring(i);
            }
            if (dic.suggest(aWord).contains(palavra.solucao)) {
                list.add(palavra);
            }
            aux = palavra.solucao;
            aux = aux.substring(0, 1).toUpperCase()
                    + aux.substring(1).toLowerCase();
            if (dic.suggest(aWord).contains(aux)) {
                palavra.solucao = aux;
                list.add(palavra);
            }
            aux = aWord;
            aux.toLowerCase();
            // what?
            if (j == 32) {
                j = 96;
                palavra.prob = 0.0237;
            } else if (j == 122) {
                j = 224;
            } else if (j == 227) {
                j = 230;
            } else if (j == 231) {
                j = 232;
```

```
                } else if (j == 234) {
                    j = 236;
                } else if (j == 237) {
                    j = 242;
                } else if (j == 245) {
                    j = 249;
                }
            }
        }
        //corrige erros dos tipos substituição, acento e cedilha
        /********
         *  SUBSTITUITION, WRONG ACCENT AND CEDILHA OMISSION
         *
         */
        aux = aWord;
        aux.toLowerCase();
        for (int i = 0; i < c.length; i++) {
            aux = aWord;
            aux.toLowerCase();

            // from a-z loop...
            for (int j = 97; j < 123; j++) {
                // for each word in aWord
                for (int k = 0; k < aWord.length(); k++) {
                    c[k] = aux.charAt(k);
                }
                Palavra palavra = new Palavra();
                // checking if is a accented word
                // reference:
https://www.utf8-chartable.de/unicode-utf8-table.pl?utf8=dec&unicodeinhtml=dec
                if ((((
                        (int) c[i] == 225) ||
                        ((int) c[i] == 226) ||
                        ((int) c[i] == 227)) && (j == 97)) ||
                        ((((int) c[i] == 233) ||
                        ((int) c[i] == 234)) && (j == 101)) ||
                        (((int) c[i] == 237) && (j == 105)) ||
                        ((((int) c[i] == 243) ||
                        ((int) c[i] == 244) ||
                        ((int) c[i] == 245)) && (j == 111)) ||
                        // ú and u
                        (((int) c[i] == 250) && (j == 117)) ||
                        // cedilha
                        (((int) c[i] == 231) && (j == 99)))
                {

                    /* Source :
                        As it turns out, errors related to the use of diacritics correspond to approximately
                        half of all spelling errors in both corpora (overall, 47.15% in C1 and 50.08% in C2,
                        corresponding to 49.48% of all single errors in C1 and 58.84% in C2), ...
                    */
```

```
            /*
               Sum of all errors related to diacritics:
               Error Classification    Times   Percentage
               Acento Errado           6220      94.16%
               Colocação
               Desnecessária             42       0.64%
               Falta de Acento           30       0.45%
            */
            palavra.prob = ACCENT_OMISSION_PROBABILITY +
                  UNECESSARY_CHAR_PROBABILITY +
                  WRONG_ACCENT_PROBABILITY;
         } else {
            // Omission
            palavra.prob = CHAR_OMISSION_PROBABILITY;
         }
         // Cedilha considerando erros com cedilha como um erro de acentuação.
         // Ver pagina 23 monografia

         if ((int) c[i] == 231) {
            // should be sum of all accent error probabilities + cedilha omission
            palavra.prob = ACCENT_OMISSION_PROBABILITY +
                  UNECESSARY_CHAR_PROBABILITY +
                  WRONG_ACCENT_PROBABILITY;
         }
         c[i] = (char) j;
         for (int l = 0; l < c.length; l++) {
            palavra.solucao = palavra.solucao + c[l];
         }
         if (dic.suggest(aWord).contains(palavra.solucao)) {
            list.add(palavra);
         }
         aux = palavra.solucao;
         aux = aux.substring(0, 1).toUpperCase()
               + aux.substring(1).toLowerCase();
         if (dic.suggest(aWord).contains(aux)) {
            palavra.solucao = aux;
            list.add(palavra);
         }
         aux = aWord;
         aux.toLowerCase();
      }
      for (int j = 225; j < 251; j++) {
         aux = aWord;
         aux.toLowerCase();
         for (int k = 0; k < aWord.length(); k++) {
            c[k] = aux.charAt(k);
         }
         char jc = (char) j;

         if ((jc == 'á') || (jc == 'â') || (jc == 'ã') || (jc == 'é') || (jc
```

```
                == 'ê') || (jc == 'ç') || (jc == 'í') || (jc == 'ó') || (jc == 'ô') || (jc
                == 'õ') || (jc == 'ú')) {
            Palavra palavra = new Palavra();
            if ((jc != 'ç') && ((c[i] == 'a') || (c[i] == 'e') || (c[i]
                    == 'i') || (c[i] == 'o') || (c[i] == 'u'))) {
                palavra.prob = 0.4715;
            } else if (((char) j == 'ç') && (c[i] == 'c')) {
                palavra.prob = 0.4715;
            } else {
                palavra.prob = 0.1291;
            }
            c[i] = (char) j;
            for (int l = 0; l < c.length; l++) {
                palavra.solucao = palavra.solucao + c[l];
            }
            if (dic.suggest(aWord).contains(palavra.solucao)) {
                list.add(palavra);
            }
            aux = palavra.solucao;
            aux = aux.substring(0, 1).toUpperCase()
                    + aux.substring(1).toLowerCase();
            if (dic.suggest(aWord).contains(aux)) {
                palavra.solucao = aux;
                list.add(palavra);
            }
        }

    }
}
boolean novo = true;
LinkedList<Palavra> listTot = new LinkedList<Palavra>();
int index = 0;
while (index < qtdeSug) {
    int count = 0;
    novo = true;
    while (count < list.size()) {
        if (dic.suggest(aWord).get(index).equals(list.get(count).solucao)) {
            Palavra p = new Palavra();
            p.prob = list.get(count).prob;
            p.solucao = list.get(count).solucao;
            listTot.add(p);
            novo = false;
        }
        count++;
    }
    if (novo) {
        Palavra p = new Palavra();
        p.prob = 0.0369;
        p.solucao = dic.suggest(aWord).get(index);
        listTot.add(p);
    }
```

```
    index++;
}
//trigramas
//encontra palavras anteriores
String palavraAntAnt = "";
String palavraAnt = "";
Set<String> set1 = corpusTeste1.keySet();
String str;
Iterator<String> itr1 = set1.iterator();
while (itr1.hasNext()) {
    str = itr1.next();
    Hashtable h2 = (Hashtable) corpusTeste1.get(str);
    Set<String> set2 = h2.keySet();
    String str2;
    Iterator<String> itr2 = set2.iterator();
    while (itr2.hasNext()) {
        str2 = itr2.next();
        Hashtable h3 = (Hashtable) h2.get(str2);
        Set<String> set3 = h3.keySet();
        String str3;
        Iterator<String> itr3 = set3.iterator();
        while (itr3.hasNext()) {
            str3 = itr3.next();
            if (str3.equals(aWord)) {
                LinkedList<Integer> listH3
                        = (LinkedList<Integer>) h3.get(str3);
                int index2 = 0;
                while (index2 < listH3.size()) {
                    if (listH3.get(index2)
                            == todasPalavras.get(aWord)) {
                        palavraAntAnt = str;
                        palavraAnt = str2;
                    }
                    index2++;
                }
            }
        }
    }
}
calculaVoc();
//encontra probabilidades Totais para todas as correções
int indice = 0;
while (indice < listTot.size()) {
    double qtdeP1P2PC = 0;
    //calculando qtdeP1P2PC
    Set<String> setPC = palavra1.keySet();
    String str1;
    Iterator<String> itrPC = setPC.iterator();
    while (itrPC.hasNext()) {
        str1 = itrPC.next();
        Hashtable h2 = (Hashtable) palavra1.get(str1);
```

```
            Set<String> set2 = h2.keySet();
            String str2;
            Iterator<String> itr2 = set2.iterator();
            while (itr2.hasNext()) {
                str2 = itr2.next();
                Hashtable h3 = (Hashtable) h2.get(str2);
                Set<String> set3 = h3.keySet();
                String str3;
                Iterator<String> itr3 = set3.iterator();
                while (itr3.hasNext()) {
                    str3 = itr3.next();
                    if (str3.equals(listTot.get(indice).solucao)) {
                        LinkedList listH3 = (LinkedList) h3.get(str3);
                        if (str1.equals(palavraAntAnt) && str2.equals(palavraAnt)) {
                            qtdeP1P2PC = qtdeP1P2PC + listH3.size();
                        }
                    }
                }
            }
        }
        //calculando probabilidades e aplicando Add-one
        listTot.get(indice).prob = listTot.get(indice).prob;
        listTot.get(indice).probP1P2PC = (qtdeP1P2PC + 1) / (V + Voc);
        listTot.get(indice).probTotal = listTot.get(indice).probP1P2PC
                * listTot.get(indice).prob;
        indice++;
    }
    //ordena lista
    OrdenaLista ol = new OrdenaLista();
    Collections.sort(listTot, ol); //listTot
    //retira palavras repetidas
    LinkedHashSet<Palavra> set = new LinkedHashSet<Palavra>();
    for (int i = 0; i < listTot.size(); i++) { //listTot
        set.add(listTot.get(i)); //listTot
    }
    //passa sugestões para aProposals
    int cont = 0;
    Iterator it = set.iterator();
    for (int i = 0; i < set.size(); i++) {
        if (it.hasNext()) {
            if (aProposals.length < cont) {
                Palavra p = (Palavra) it.next();
                aProposals[cont] = p.solucao;
                cont++;
            }
        }
    }
    //System.out.println(aWord);
    //for (int i = 0; i < qtdeSug; i++) {
      //  System.out.println(aProposals[i]);
    //}
```

```
            //System.out.println();

            return new com.example.XSpellAlternatives_impl(aWord, aLocale,
                  SpellFailure.SPELLING_ERROR, aProposals);
         } else {
            aProposals[0] = dic.suggest(aWord).get(0);
            System.out.println(aWord + "\n" + aProposals[0] + "\n");
            return new com.example.XSpellAlternatives_impl(aWord, aLocale,
                  SpellFailure.SPELLING_ERROR, aProposals);
         }
      }

      private boolean IsEqual(Locale aLoc1, Locale aLoc2) {
         return aLoc1.Language.equals(aLoc2.Language)
               && aLoc1.Country.equals(aLoc2.Country)
               && aLoc1.Variant.equals(aLoc2.Variant);
      }

      public boolean
removeLinguServiceEventListener(com.sun.star.linguistic2.XLinguServiceEventListener xLstnr)
      {
         // NOTE: Default initialized polymorphic structs can cause problems
         // because of missing default initialization of primitive types of
         // some C++ compilers or different Any initialization in Java and C++
         // polymorphic structs.
         return false;
      }


      public com.sun.star.lang.Locale[] getLocales()
      {
         Locale aLocales[] = {
            new Locale("it", "IT", "")
         };
         return aLocales;
      }

};
```

APÊNDICE K - Código fonte *LevenstheinDistance.java*

```
public class LevenstheinDistance {

   public static int distance(String a, String b) {
      a = a.toLowerCase();
      b = b.toLowerCase();
      // i == 0
      int [] costs = new int [b.length() + 1];
```

```
        for (int j = 0; j < costs.length; j++)
            costs[j] = j;
        for (int i = 1; i <= a.length(); i++) {
            // j == 0; nw = lev(i - 1, j)
            costs[0] = i;
            int nw = i - 1;
            for (int j = 1; j <= b.length(); j++) {
                int cj = Math.min(1 + Math.min(costs[j], costs[j - 1]), a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw
+ 1);
                nw = costs[j];
                costs[j] = cj;
            }
        }
        return costs[b.length()];
    }
}
```

APÊNDICE L - Código fonte

DamerauLevensthein.java

```
public class DamerauLevenshtein {

    public static int distance(String source, String target) {
        if (source == null || target == null) {
            throw new IllegalArgumentException("Parameter must not be null");
        }
        int sourceLength = source.length();
        int targetLength = target.length();
        if (sourceLength == 0) return targetLength;
        if (targetLength == 0) return sourceLength;
        int[][] dist = new int[sourceLength + 1][targetLength + 1];
        for (int i = 0; i < sourceLength + 1; i++) {
            dist[i][0] = i;
        }
        for (int j = 0; j < targetLength + 1; j++) {
            dist[0][j] = j;
        }
        for (int i = 1; i < sourceLength + 1; i++) {
            for (int j = 1; j < targetLength + 1; j++) {
                int cost = source.charAt(i - 1) == target.charAt(j - 1) ? 0 : 1;
                dist[i][j] = Math.min(Math.min(dist[i - 1][j] + 1, dist[i][j - 1] + 1), dist[i - 1][j - 1] + cost);
                if (i > 1 &&
                        j > 1 &&
                        source.charAt(i - 1) == target.charAt(j - 2) &&
                        source.charAt(i - 2) == target.charAt(j - 1)) {
                    dist[i][j] = Math.min(dist[i][j], dist[i - 2][j - 2] + cost);
                }
            }
```

```
        }
        return dist[sourceLength][targetLength];
    }
}
```