

CAIO ESDRAS DE BRITO BEGOTTI

**O LATIM CLÁSSICO DE CÍCERO À LUZ DA LINGUÍSTICA
DE CORPUS: DESCRIÇÃO E IMPLEMENTAÇÃO DE
MÉTODOS COMPUTACIONAIS**

Monografia apresentada à disciplina Orientação Monográfica II como requisito parcial à obtenção do título de Bacharel em Linguística, do Setor de Ciências Humanas, Letras e Artes, Universidade Federal do Paraná.

Orientador: Alessandro Rolim de Moura

CURITIBA
JULHO DE 2012

Termo de Aprovação

CAIO ESDRAS DE BRITO BEGOTTI

O LATIM CLÁSSICO DE CÍCERO À LUZ DA LINGUÍSTICA DE CORPUS: DESCRIÇÃO E IMPLEMENTAÇÃO DE MÉTODOS COMPUTACIONAIS

Dissertação aprovada como requisito parcial para obtenção do título de Bacharel em Linguística, do Setor de Ciências Humanas, Letras e Artes, Universidade Federal do Paraná, pela seguinte banca examinadora:

Prof. Dr. Alessandro Rolim de Moura
UFPR

Prof. Dr. Rodrigo Tadeu Gonçalves
UFPR

Prof. Dr. José Borges Neto
UFPR

Curitiba, XX de julho de 2012

Agradecimentos

Aos que por anos conseguiram conciliar oito horas diárias de trabalho com mais outras quatro de estudos, pela motivação.
Aos meus amigos e todos que me apoiaram, deram suporte e confiança para que eu pudesse me tornar um linguista.
Aos meus pais, por terem sempre me rodeado com “letras”.

Sumário

Lista de Figuras.....	vi
Lista de Siglas	vii
Resumo.....	viii
Abstract	ix
1 Organização.....	1
2 Introdução.....	2
2.1 Motivação	3
2.2 Por que Cícero?	4
2.3 Diferenciais tecnológicos	6
2.4 Objetivos.....	8
3 Fundamentação teórica.....	11
3.1 Linguística aplicada e computacional	11
3.2 Linguística de corpus.....	12
3.2.1 Histórico	12
3.2.2 Casos de uso	13
4 Metodologia	15
4.1 Corpora	15
4.1.1 Especificação	16
4.1.2 Escopo	18

4.1.2.1	Discursos	19
4.1.2.2	Retórica	20
4.1.2.3	Poesia	20
4.1.2.4	Filosofia	20
4.1.2.5	Cartas	20
4.1.2.6	Outros	20
4.2	Stopwords	21
4.2.1	Conjunções	22
4.2.2	Preposições	23
4.2.3	Pronomes	23
4.3	NLTK	24
4.3.1	Análise de frequência	24
4.3.2	Concordâncias	25
4.4	Python	25
5	Análises	27
5.1	Stopwords	27
5.2	Lei de Zipf	29
5.3	Radicalização	30
5.4	Concordâncias	32
6	Conclusão	34
	Referências	35
	Apêndice A – Sugestão de léxico para estudo	38
	Apêndice B – Códigos	41
B.1	Scraper de dados e montador dos corpora	41

B.2	Filtros dos corpora	43
B.3	Classe de manipulação dos corpora.....	44
B.4	Parser e filtro de pré-nomes latinos.....	48
B.5	Testador de stopwords	48
B.6	Analisador de frequências	49
B.7	Gerador de concordâncias.....	52
B.8	Sugestões de léxico para aprendizagem	54
B.9	Protótipo de algoritmo de radicalização.....	55
B.10	Consulta automatizada ao LemLat	57

Lista de Figuras

Figura 1	Gráfico de frequência (100 termos mais usados, incluindo stopwords)	..	28
Figura 2	Gráfico de frequência (100 termos mais usados)	28
Figura 3	Gráfico atestando Lei de Zipf aos corpora	30
Figura 4	Gráfico de frequência (100 termos radicalizados)	32

Lista de Siglas

NLTK	Natural Language Toolkit
LC	Linguística de Corpus
API	Application Programming Interface
PHI	Packard Humanities Institute
LA	Linguística Aplicada
PLN	Processamento de Linguagem Natural
NLP	Natural Language Processing
BLEU	Bilingual Evaluation Understudy
XML	Extensible Markup Language
UTF-8	8-bit Unicode Transformation Format

Resumo

Escreva aqui o texto de seu resumo...

Palavras-chave: Escreva; Aqui; Suas; Palavras-chave.

Abstract

Write here the English version of your 'Resumo'...

Key-words: Write; Here; Your; Key-words.

1 Organização

TODO: revisar bibliografias por datas de re-edicao, revisao e nova impressao somente

Este trabalho está dividido em seis capítulos principais, seguidos pelas referências bibliográficas e uma série de apêndices técnicos.

O primeiro capítulo *Introdução* explica a motivação por trás do trabalho, diferenças entre pesquisas similares já realizadas e quais são os objetivos em vista. O capítulo *Fundamentação teórica* elenca todas as principais referências linguísticas do trabalho, histórico da área de pesquisa e compara métodos da linguística aplicada, com casos de uso e propostas.

O terceiro capítulo *Metodologia* é o capítulo no qual se encontram todos os detalhes teóricos do trabalho. É nele que está a especificação de escopo, formato dele e as análises a serem feitas, bem como explicações mais técnicas sobre as ferramentas utilizadas. O quarto capítulo *Análises* lista e documenta todas as análises linguísticas deste trabalho, estando em sintonia direta com o capítulo anterior e o seguinte. É neste capítulo que se encontram os dados resultantes do trabalho prático e se discute possíveis utilidades para as análises deles. Há neste capítulo uma breve sugestão de como a linguística aplicada pode ser utilizada em sala de aula e por estudantes autônomos.

O capítulo *Propostas de continuidade* indica caminhos a serem seguidos por pesquisadores e estudantes interessados neste trabalho, afim de que os resultados e métodos possam ser refinados e melhorados. O último capítulo *Conclusão* fecha o trabalho com um resumo geral do que foi desenvolvido.

Por fim, há uma série de apêndices deste trabalho que exemplificam todas as análises realizadas utilizando programação de computadores. Todos os códigos escritos para este trabalho estão nos apêndices e podem ser utilizados livremente.

2 Introdução

O trabalho que aqui se apresenta foca na construção de *corpora* linguísticos de textos do período clássico do latim e em possibilidades de análise computacional desses *corpora*. O propósito por trás dessas duas tarefas é o de sugerir melhoras em métodos de ensino do latim e permitir uma melhor compreensão dessa língua de forma mais empírica, ou seja, uma compreensão baseada no uso que um dia foi dado à língua através da análise de textos reais em latim clássico utilizando conceitos de PLN.

Para tal, este trabalho se baseará no projeto NLTK (Natural Language Toolkit), que consiste em um conjunto de rotinas de programação de computador, em forma de kit de ferramentas¹, para processamento de linguagem natural. Com a ajuda do NLTK e à luz da Linguística de Corpus (LC) procurarei evidenciar os usos mais comuns do latim clássico por Cícero e irei sugerir uma abordagem para uso disso em sala de aula.

Marcus Tullius Cicero, daqui para frente simplesmente Cícero, será o autor escolhido para a montagem dos *corpora*. Cícero é um dos autores que mais representam o período clássico do latim — tanto pela grande produção e variedade de estilos de seus textos quanto pelos seus discursos —, e devido a sua importância após a Renascença nada mais natural do que usá-lo nas análises deste trabalho.

Espera-se ainda, com este trabalho, contribuir para o ainda pequeno grupo de linguistas computacionais no Brasil, especialmente os que apreciam e trabalham com o latim. A exemplo do que já foi feito com o inglês utilizando métodos semelhantes aos que aqui serão apresentados, pode-se ganhar bastante em sala de aula e na pesquisa com tal abordagem: aproximando a linguística de corpus à linguística computacional e às línguas naturais (SARDINHA, 2000b).

¹A descrição pode soar estranha em português e para não iniciados em programação; em bibliografias de computação ambos os termos aparecem simplesmente como API e toolkit, em inglês.

2.1 Motivação

É ponto pacífico que atualmente o ensino de latim se dá de forma bastante limitada em escolas e universidades brasileiras. Todavia, o latim clássico parece estar em uma espécie de retomada em países como Inglaterra (que vale dizer sempre foi um centro de referência em estudos clássicos) e Estados Unidos. Nos Estados Unidos, por exemplo, relatórios mostram o latim entre as oito línguas mais estudadas hoje no país, na frente do russo e do português, com crescimento de mais de 20% em matrículas nos últimos anos (FURMAN; GOLDBERG; LUSIN, 2010). Embora parte dessas matrículas ainda seja para cursos introdutórios, nos quais os alunos pouco aprendem sobre a língua e dependem exclusivamente dos professores, esse número é notável.

Muitas vezes o latim acaba sendo ensinado somente de cima para baixo, ou seja, dos professores aos alunos, pois os primeiros são uma fonte natural de conhecimento e autoridade. Assim, alunos de latim com frequência simplesmente tem que confiar na capacidade de julgamento dos mestres na escolha do melhor material a ser estudado. O que de fato faz sentido, pois os professores sabem avaliar a melhor linha de aprendizado. Mas embora os professores possam fazer boas escolhas quanto aos textos e ao vocabulário a serem estudados, tais escolhas serão sempre parciais ou limitadas, de modo que a criação de materiais e métodos que fundamentem tais escolhas de forma objetiva é sempre bem-vinda.

Por si só isso não representa um grande problema. Entretanto, por se tratar de uma língua sem usuários nativos — logo, poucos podem julgar se o que está sendo estudado de fato representaria a realidade dos usuários de uma língua de dois mil anos atrás —, ninguém na realidade sabe dizer com total certeza se, por exemplo, a seleção de textos e a escolha de determinado vocabulário de estudo são ou não adequados, ou ainda se eles são suficientemente representativos para os alunos ganharem proficiência mais rapidamente.

Isso é até aceitável em línguas estrangeiras modernas, pois o aluno tem mais contato com elas no seu dia-a-dia e pode sair de um beco sem saída com mais facilidade e sem recorrer à ajuda dos professores, mas com línguas clássicas como o latim e o grego, o prejuízo pode se tornar maior para o aluno que se dedicou por tanto tempo e então percebe que tomou um caminho de estudo não muito bom e perdeu uma melhor oportunidade de aprendizado. É claro que se espera dos alunos que eles possam compensar eventuais lacunas desse aprendizado através de estudo e leituras próprias, mas infelizmente não se

pode contar com isso sempre.

Em uma situação limite, um estudante de latim clássico poderia estar aprendendo um vocabulário que outrora pertenceu a um texto menor e que não corresponde à realidade linguística que ele buscava, ou que também não facilita a assimilação de idéias do estilo que procurava entender. Naturalmente, deve-se considerar aqui que o ensino de latim costumava focar em aspectos altamente literários e clássicos. Por exemplo, alunos de latim comumente leem Cícero após um ou dois anos de estudo, mas Cícero representa apenas uma fatia do todo que foi o latim em sua época. Deixa-se então o latim vulgar, falado pelo povo, para aulas de linguística românica, e textos religiosos para aulas sobre latim medieval ou eclesiástico.

Este trabalho, obviamente, não tentará esgotar as análises linguísticas que se pode fazer em *corpora* em latim. Seu escopo será delimitado em torno do latim clássico por razão do tempo limitado para pesquisa, pela complexidade da tarefa de montar *corpora* maiores sem ajuda de terceiros e pela importância do latim clássico de Cícero. Também restarão, de qualquer forma, os métodos e as conclusões deste trabalho para que no futuro outros o levem adiante. Realizando-se as montagens e análises sobre os *corpora* de Cícero, já se terá conseguido avançar um bom caminho.

2.2 Por que Cícero?

Provavelmente se conhece mais a respeito da vida de Cícero do que de qualquer outra pessoa do mundo antigo, ainda assim poucas pessoas hoje saberiam dizer quem foi ele. Dessa maneira, cabe aqui uma breve biografia de Cícero, como narrada em detalhes por Petersson (2005) e Everitt (2001).

Nascido fora da cidade de Roma, Cícero foi levado ainda pequeno para lá. Foi em Roma onde estudou retórica, filosofia e cresceu até entrar no mundo político da cidade, quando começou a acompanhar, ainda bastante jovem, casos públicos no fórum. Aos trinta e poucos anos Cícero já era uma figura importante no meio político e jurídico da cidade. Por volta dos quarenta anos Cícero vira cônsul, o título civil mais alto que alguém poderia ter. Não é exagerado dizer que Cícero foi possivelmente o maior e melhor advogado e orador de Roma.

Embora personagem secundário na história geral do império, Cícero esteve ligado a acontecimentos políticos marcantes como a conspiração de Catilina contra a república e a oposição à César quando este subiu ao poder. Como bem coloca Paratore (1983),

Cícero estava destinado a personificar, com profundidade exemplar, o mal-estar da sua época, e a imprimir, nas letras latinas, uma marca profunda. Foi ele quem trouxe parte do pensamento filosófico grego para o mundo latino, e sua importância para o pensamento ocidental foi tamanha que a igreja católica o declarou um bom pagão, fato decisivo para que tantos dos seus escritos fossem preservados até os dias atuais. Cícero entrou para a história política e jurídica de Roma não por simplesmente vencer casos, mas por ser brilhante em suas exposições e pela sua capacidade de transformar o ato de falar em público em uma arma poderosa.

A influência e a importância de Cícero como pensador jamais serão suficientemente demonstradas, mas é evidente que o latim como língua deve muito a Cícero. Ele popularizou estilos de discurso, cunhou novas palavras² e limou da língua tantas outras, criou parâmetros de expressão em latim. De certa forma, pode-se dizer que Cícero está até os dias atuais impregnado no modo como pessoas públicas se expressam, por exemplo políticos e juristas. Obviamente Cícero não exerce tanto fascínio simplesmente porque foi Cícero, mas pela qualidade da sua expressão e seus estilos (FISHWICK, 2007). Expressão e estilos que foram reconhecidos ainda em vida, vale lembrar. Quintiliano e César o viam como um “supremo manipulador dos corações alheios” e “quase um pioneiro e inventor da eloquência” (KENNEY; CLAUSEN, 1983).

É importante lembrar, entretanto, que Cícero não pode ser tomado como um autor que representa toda a grandeza do latim clássico sozinho. Embora o período literário em que viveu hoje seja chamado de *ciceroniano*, ele estava apenas inserido em todo o período de ouro do latim clássico, rodeado por outros autores. É necessário considerar isso por todo este trabalho. O que será analisado pode muito bem nem mesmo representar toda a produção de Cícero de forma justa, vale dizer.

Neste trabalho serão feitas análises que permitirão a estudiosos de Cícero avaliar uma de suas grandes características, a chamada *elegantia*. *Elegantia* se traduz não simplesmente por “elegância” como seria de se esperar pelo cognato, mas como um refinamento lexical e uma perfeita escolha de palavras para expressar uma idéia. *Elegantia* é saber utilizar a palavra certa, no momento certo, de maneira adequada (PALMER, 1988). Kennedy e Wooten (2001) também lembram elogios do próprio Cícero a César pelo uso exemplar de vocabulário que César demonstra em seu *De Analogia*. Cícero declara que a escolha precisa de palavras ao falar e escrever é um dos vários requisitos para alguém ser

²Aqui cabe uma parte interessante, sobre o esforço de Cícero de tentar introduzir novas idéias no mundo latino sem abusar de palavras e conceitos que só existiam em grego, como notou Lucrécio certa vez.

um verdadeiro cidadão romano. Portanto, nada mais adequado do que tentar enxergar como isso se dava utilizando a LC.

Ajuda entender isto algumas passagens em um de seus textos mais conhecidos, *De Oratore*. Podemos ver a preocupação de Cícero com o modo rústico de se expressar de alguns romanos — tanto no sotaque falado quanto nas palavras grosseiras utilizadas por alguns —, como isso afetava a latinidade deles segundo Cícero e como isso devia ser expurgado da língua³. É particularmente interessante o trecho em que Cícero destaca a importância de ser claro em um discurso, a necessidade de se evitar ambiguidades e conseguir passar o sentido literal do que se tem em mente empregando o uso exato de palavras⁴. É nesse texto que Cícero nos dá embasamento para este trabalho: segundo ele, ao resumir os bons valores de oratória, um orador precisa ter cultura abrangente, que de uma forma ou de outra se mostra em seu texto ou discurso pela riqueza de seu vocabulário, pois não é possível esgotar todos os temas com um universo lexical reduzido⁵.

Embora já se tenha visto *corpora* de latim clássico — alguns, como o do Packard Humanities Institute (PHI), cobrindo todos os textos e discursos de Cícero —, pesquisadores e alunos até hoje não parecem ter desenvolvido pesquisas computacionais práticas em cima deles, como Dee (2000); ou seja, processamento de linguagem natural que conseguisse chegar a conclusões a partir de análises estatísticas dos textos e que fosse útil, por exemplo, em sala de aula. Vale notar ainda que, segundo May e Craig (2002) a respeito dos famosos bancos de dados de latim do PHI⁶, “citar o PHI se tornou banal em qualquer artigo trabalhando com o uso de palavras ciceriano. Talvez a facilidade em obter dados do PHI faça com que conclusões retiradas deles sejam menos valorizadas, e ainda não se viu a publicação de trabalhos sobre o vocabulário ciceroniano, e seu uso, feitos especificamente com a ajuda de computadores”⁷.

2.3 Diferenciais tecnológicos

Parece que um dos motivos pelos quais isso ocorre é a falta de um catálogo linguístico cientificamente montado com as palavras de fato mais usadas da língua. Já

³*De Oratore*, livro 3, parágrafo 42.

⁴*De Oratore*, livro 3, parágrafo 49.

⁵*De Oratore*, livro 3, a partir do parágrafo 74.

⁶O mesmo parece se aplicar aos *corpora* de latim clássico do projeto Perseus, da Tufts University.

⁷Tradução própria. No original, em inglês “[...] citation of this resource has now become commonplace in any article touching upon Ciceronian word usage. Perhaps because the ease of retrieval of these data makes conclusions drawn from them less prized, publication of specifically computer-based work on Ciceronian vocabulary and usage is still in the future”.

houve tentativas de se fazer isso para o latim, porém em uma época cujos poderes da análise linguística feita pelo computador ainda não eram conhecidos (DIEDERICH, 1939), ou foram tentativas focadas em outras línguas, como o inglês (ALMEIDA, 1997) e o espanhol (JACOBI, 2001). Outros autores, como Pellegrino (2009), no máximo se limitaram a criar listas de palavras bastante usadas por Cícero a serem estudadas para exames de latim de escolas americanas, sem uma maior racionalização sobre os *corpora* de Cícero e sem também elaborar em seus métodos. Um trabalho bastante superficial e parecendo-se mais com um manual de prova. Embora os resultados de todas essas experiências tenham sido utilizados de maneira distinta da que aqui se propõe, a motivação é similar: utilizar artefatos reais da língua para o aprendizado dela mesma.

Criando-se hoje *corpora* linguísticos do latim em um formato reconhecido por linguistas computacionais, utilizando-se métodos documentados e comprovadamente eficientes para análise linguística, é possível melhorarmos o ensino de latim em escolas e universidades, além de permitir ao aluno — independente ou aconselhado — que tome o ensino da língua nas suas próprias mãos. Como bem diz Jacobi (2001), “essa forma de trabalho enfatiza o desenvolvimento da habilidade de descoberta nos alunos. A aprendizagem movida a dados posiciona o aluno no papel de descobridor ou de pesquisador, e o professor passa a ter como função primordial propiciar meios para que os alunos desenvolvam estratégias de descoberta”. Para tal, o volume de dados a ser analisado e utilizado nem mesmo precisa ser muito grande para se notar bons ganhos no aprendizado (WILLIS, 1990).

Ainda que existam diversos modelos, bases de dados e programas de computador para a análise de *corpora*, muitos possuem seus códigos fechados ou requerem licenciamentos, ou seja, não permitem o uso irrestrito deles por parte dos usuários e pesquisadores caso estes desejem, por exemplo, aplicar melhorias nesses softwares ou corrigir falhas nas análises por conta própria. Além disso, alguns são bastante caros — o que porém é compreensível dada a dificuldade de catalogar e organizar bases enormes —, e fogem da realidade acadêmica brasileira, ainda carente de recursos para latinistas⁸. Logo, uma solução aberta, sem nenhum tipo de restrição e preferivelmente barata se faz necessária. Essa solução, que será detalhada nas próximas seções, deve ainda aproveitar a possibilidade de utilizar textos sem amarras de copyright como no caso de textos clássicos em

⁸Softwares para análise linguística, por mais simples que sejam suas funcionalidades, não custam menos que algumas dezenas de dólares, quando não centenas. Bancos de dados prontos com *corpora* costumam também cobrar licenças de uso temporário que podem em alguns casos chegar a milhares de euros ou, embora “gratuitas”, acabam restringindo o uso e a pesquisa comercial através desses mesmos bancos de dados.

latim.

Portanto, não está sendo proposta aqui uma revolução no jeito de trabalhar o latim com computação, mas simplesmente uma alternativa viável, barata e eficiente para alunos, pesquisadores e entusiastas brasileiros. Não é incomum encontrar estudantes e pesquisadores de linguística que enxergam PLN como algo de outro mundo, e se limitam a utilizar programas de computador prontos, com rotinas pré-programadas, e não se incomodam em entender como tudo aquilo é feito. Este trabalho vai na contra-mão disso, e se apresenta como um diferencial para reverter essa imagem do PLN.

2.4 Objetivos

O objetivo primordial deste trabalho é permitir uma maior compreensão de textos clássicos em latim pelos estudantes dessa língua, através da criação de vocabulários específicos e ferramentas automatizadas por processamento de linguagem natural, cujas fontes serão os textos de Cícero, do período clássico do latim. De forma análoga a Almeida (2003), esperamos que alunos de latim consigam se beneficiar do mesmo modelo de aprendizado comprovado para o inglês. Dessa maneira, poderemos encontrar respostas para perguntas que não são facilmente comprováveis sem métodos como os propostos neste trabalho, como por exemplo:

1. que vocabulário é mais adequado para determinados alunos ou pesquisadores em um contato inicial com o latim clássico?
2. que tipo de palavrado era mais comum em textos filosóficos e literários, e qual era mais comum em textos informais (como em cartas)?
3. quais as melhorias possíveis no ensino de latim clássico com o auxílio de computadores?

Através de resultados formatados, alunos e professores poderão focar no aprendizado dos textos em si, por exemplo, não desperdiçando tempo em decorar vocabulários sem necessidade, assim como já se mostrou proveitoso para outras línguas. Em última instância, pode-se dizer que o grande objetivo deste trabalho é demonstrar como o ensino do latim clássico pode se tornar melhor com a ajuda da LC.

A busca pelo domínio da leitura é um dos maiores objetivos de qualquer interessado em línguas, e isso não é diferente para o latim. Este trabalho se propõe a ajudar

nisso. A motivação inicial veio através de um projeto similar focado em língua inglesa na UNICAMP. Hoje transformado em livro e guia de estudo, o projeto começou como um mero catálogo das palavras mais comuns do inglês, e desencadeou novas formas de estudar a língua que poderiam beneficiar também o latim (ALMEIDA, 2003).

Teremos também um objetivo secundário que deriva das propostas anteriores: a criação de *corpora* e um catálogo de *stopwords* do latim. Os *corpora* poderão ser utilizados para processamento de linguagem natural por terceiros, da mesma forma que serão utilizados neste trabalho. O catálogo de *stopwords* poderá também ser utilizado por terceiros, e tentará ser abrangente, pois atualmente não se tem um catálogo completo para o latim assim como se tem para outras línguas.

Além de responder às perguntas acima e de chegar a alguns resultados possíveis já citados, será possível gerar uma lista ordenada de palavras mais comuns do latim clássico de Cícero que possam ser utilizadas em sala de aula e eventualmente em concordâncias.

A força que instiga este trabalho ao desenvolvimento de LC é muito bem explicada por Sapir (1949)⁹, que também acreditava que “o vocabulário é um índice bastante sensível da cultura de um povo, e mudanças de significado, perda de palavras, a criação e o empréstimo de novas outras são todos fatos dependentes das histórias de suas próprias culturas. Línguas diferem grandemente pela natureza de seus vocabulários. Distinções que parecem inevitáveis para nós podem ser solenemente ignoradas em línguas que refletem um tipo de cultura totalmente diferente, ao passo que elas podem também insistir em distinções que são incompreensíveis para nós. Tais diferenças de vocabulário vão muito além de nomes de objetos como a ponta de uma flecha, uma armadura ou canhoneira. Elas se aplicam também ao mundo mental”.

Enfim, tal abordagem se mostra, além de eficiente, bastante realista, ao passo que aponta para os alunos somente o que vale a pena ser aprendido e aprendido não através de contextos artificiais ou idealizados da língua. Como lembra Abney (1996), se a aquisição de estruturas sintáticas é trivial para nós — como atualmente se acredita ser —, então o que resta é o aprendizado do léxico da língua alvo do aprendizado. Dados os fatos expostos até aqui, compreende-se mais facilmente porque a LC tem papel fundamental

⁹Tradução própria. No original, em inglês “vocabulary is a very sensitive index of the culture of a people and changes of meaning, loss of old words, the creation and borrowing of new ones are all dependent on the history of culture itself. Languages differ widely in the nature of their vocabularies. Distinctions which seem inevitable to us may be utterly ignored in languages which reflect an entirely different type of culture, while these in turn insist on distinctions which are all but unintelligible to us. Such differences of vocabulary go far beyond the names of cultural objects, such as arrow point, coat of armor or gunboat. They apply just as well to the mental world”.

na aquisição de línguas; no caso deste trabalho, o latim.

3 Fundamentação teórica

3.1 Linguística aplicada e computacional

Todo este trabalho se sustenta sobre os princípios da Linguística Aplicada (LA). A LA por sua vez remete ao uso de ferramentas linguísticas destinado à resolução e análise de problemas reais das línguas, ou seja, é o estudo prático e não teórico delas. Particularmente, o subcampo da LA mais em voga atualmente é o da linguística computacional, que de forma simplificada é um campo de estudo que envolve diversas áreas como: ciência da computação, matemática, neurologia e a própria linguística, para citar os mais populares.

A história da linguística computacional está, naturalmente, ligada ao surgimento dos primeiros computadores logo após a II Guerra Mundial. Muitos problemas linguísticos hoje envolvem resolver algoritmos ou modelar dados — com frequência em grande escala — e foi exatamente para estes tipos de tarefas que os computadores foram criados. Dessa forma, não é surpresa alguma que ambas as áreas estejam hoje tão próximas; no mundo privado ao menos, pois na academia brasileira tenho a impressão de que ainda não alcançamos massa crítica de pesquisa.

Houve trabalhos que tentaram medir essa ainda pequena produção em LC no Brasil, porém em níveis acadêmicos muito distantes dos estudantes de graduação (LA-DEIRA, 2010), e buscas em portais de pesquisa ainda mostram um número reduzido de grupos trabalhando nisso, alguns exclusivos e de acesso bastante difícil para não-doutores (CNPQ, 2012). De fato, parece até mesmo haver um choque cultural entre cientistas da computação e linguistas, sobre quais métodos para se avaliar a língua são melhores ou “mais corretos”, ainda que embora os primeiros tenham contribuído sobremaneira nos últimos anos para a linguística como um todo (ABNEY, 1996).

Do contato da linguística e da computação, então, chegamos ao Processamento de Linguagem Natural (PLN)¹⁰, que basicamente foca em problemas como análise automática

¹⁰É digno de nota que ainda encontra-se maior diversidade bibliográfica procurando-se pelo acrônimo e termo em inglês, NLP, de Natural Language Processing.

de discurso, tradução por máquina, análises morfo-sintáticas, reconhecimento ou geração de fala¹¹, etiquetamento de anunciados¹², radicalização de palavras¹³, gramática categorial, prosódia semântica de textos¹⁴, análises estatísticas, entre outros. Pelo fato de a linguística computacional e PLN serem áreas que atualmente quase sempre andam juntas, poucos conseguem distinguir diferenças entre elas. A cada dia surgem novas aplicações possíveis além das listadas e convencionou-se chamar tudo isso simplesmente “processamento de linguagem natural”.

3.2 Linguística de corpus

3.2.1 Histórico

Dentro da linguística computacional encontramos afinal a LC, campo da linguística que estuda a língua como objeto vivo através de modelos estatísticos e amostragens de dados reais da língua, não representações idealizadas dela. É um campo da linguística totalmente empirista. Entretanto isso não significa que é livre de teorizações, pelo contrário. A única diferença da LC seria que o trabalho teórico é feito *a posteriori*, após se obter dados concretos da língua para análise (SARDINHA, 2004).

Para os aderentes a essa linha teórica, a língua deve se expressar por si mesma, pelos textos e discursos que surgem naturalmente de seus usuários, e daí o linguista pode aferir fatos e analisá-la. A LC, dessa maneira, evita trabalhar com textos criados artificialmente, e seus modelos são modelos de dados e não modelos teóricos de representação da língua; podem ser textos escritos ou falas transcritas, organizados em um dado formato e com categorização padronizada, além de poderem ser manualmente mapeados para um processamento posterior.

Antigamente a LC se encontrava em um estado bastante primitivo, muitas vezes sendo trabalhada por formas manuais. Hoje em dia ela é um campo de estudo que anda junto com a linguística computacional, dado o poder de processamento e automação de um computador a serviço do linguista (SINCLAIR, 1991). Ao se comparar a combinação do linguista com a máquina a um linguista solitário com papel e lápis, as vantagens são evidentes. Contudo, não houve falta de críticas à LC no passado (NORVIG, 2011), quando o uso da LC e análises estatísticas ainda eram tão primitivos que não

¹¹Speech-to-text e text-to-speech.

¹²Tagging.

¹³Word stemming.

¹⁴Sentiment analysis.

eram inteiramente compreendidos, e a coleta de dados e a medição deles era vista quase como irrelevante cientificamente. Nada mais longe da verdade. A linguística não é uma ciência inteiramente teórica. Fazer ciência é desenvolver teorias sobre fatos, e fatos não surgem espontaneamente. A LC tenta ajudar no surgimento desses fatos, ainda que eles se pareçam muitas vezes com amontados de números e estatísticas, ou que modelos da LC não sejam universais para todas as línguas; não são e nunca serão.

3.2.2 Casos de uso

Somente nas últimas décadas a LC tomou o palco das ciências das línguas. Por exemplo, foi somente por volta dos anos 70 que os primeiros *corpora* de línguas faladas foram montados e analisados digitalmente¹⁵. Portanto, não é de surpreender que tão pouco a utilizem ainda e, particularmente no Brasil, exista certa falta de tradição nesse ramo da linguística aplicada.

Hoje, virtualmente todas as ferramentas digitais de tradução, análise linguística e construção de dicionários utilizam a LC como base. Grandes empresas, como o Google e a IBM, utilizam seu poder computacional e infra-estruturas gigantescas para, através de trilhões de fontes de textos disponíveis na Internet — um amontoado de *corpora* de domínio público esperando para ser analisado —, construir conversores de texto para fala, analisadores sintáticos, tradutores automáticos etc.

No caso dos tradutores automáticos, por exemplo, é graças à LC que recentemente o serviço Google Translate passou a suportar a tradução de textos em diversas línguas do e para o latim, já que seria quase impossível implementar de forma fixa todas as regras gramaticais do latim em um tradutor automático. Embora os resultados de ferramentas como o Google Translate ainda não pareçam muito profissionais, o fato é que traduções baseadas em estatísticas de *corpora* tendem a apresentar melhores resultados que as baseadas em regras linguísticas fixas (OCH, 2005). A qualidade dos resultados está diretamente relacionada ao volume de dados a serem processados. É por isso que muitas vezes textos de serviços como o do Google ainda mostram ter uma sintaxe estranha. Quanto maiores e mais abrangentes forem os *corpora*, melhores serão as traduções¹⁶. É por essa razão que muitos projetos de LC ainda engatinham e continuam a coletar dados

¹⁵Os dicionários *Collins COBUILD English Language Dictionary* e *The American Heritage Dictionary of the English Language* foram os primeiros a utilizar a LC em sua construção algumas décadas atrás.

¹⁶Uma das formas mais populares para avaliação de traduções automáticas baseadas em estatísticas de *corpora* é a pontuação BLEU, proposta pela IBM, que indica o grau de inteligibilidade de um texto traduzido por um computador comparando-o a um traduzido por um ser humano, atestando sua qualidade.

para melhorar suas análises. Somente recentemente passamos a ver e utilizar corpus com trilhões de palavras e que são úteis para treinar modelos estatísticos, seja para a tradução automática de textos entre línguas ou simplesmente correções ortográficas em um trabalho escolar (FRANZ; BRANTS, 2006).

O uso da LC se mostrou também bastante útil para filólogos e para a linguística histórica. Devido principalmente ao poder de processamento dos computadores atuais, e a criação de novos algoritmos para análise de LC, linguistas trabalhando na construção de árvores de famílias linguísticas conseguem testar hipóteses a respeito da origem comuns das línguas, do indo-europeu e finalmente solidificar uma base científica à filogenia (CLACKSON, 2007). A filogenia, portanto, hoje depende sobremaneira da pesquisa em LC e análise matemática de dados linguísticos.

Entretanto, os maiores casos de uso da LC ainda são em língua inglesa. Ocorre que avanços computacionais foram feitos primariamente em inglês nas últimas décadas, e por várias razões políticas e econômicas, a língua inglesa ainda é considerada a língua *de facto* da LC. À princípio isso não é um problema, porém é comum ler e confundir “LC do inglês” simplesmente como sinônimo de “LC”, o que não é verdade¹⁷. Apesar de ser extremamente difícil encontrar pesquisa em LC bem feita com, por exemplo, línguas africanas, orientais, e até mesmo com o latim, isso não significa que é impossível ou que não deveria ser encorajado. O inglês não pode ser um sinônimo exclusivo de LC como é hoje. É justamente essa abordagem que se apresentará neste trabalho.

¹⁷Por exemplo, mesmo para quem não é iniciado em LC, uma rápida busca no *LRE Resources Map* em <http://www.resourcebook.eu/LreMap/faces/views/resourceMap.xhtml> demonstra como isso não é verdade.

4 Metodologia

A metodologia deste trabalho se sustenta sobre um pressuposto bastante simples, porém muitas vezes renegado em LC: a reprodutibilidade do método para se chegar aos mesmos dados apresentados. Infelizmente ainda se encontra trabalhos de LC e linguística computacional que simplesmente explicam dados sem detalhar como eles foram obtidos, como foram processados e com que critérios foram trabalhados.

Não é fato raro, também, encontrar pesquisas feitas utilizando programas de computador desenvolvidos por terceiros, aos quais nem os próprios pesquisadores dessas pesquisas têm acesso total (*i.e.* ao código fonte do programa). É um ponto central deste trabalho a crença de que ciência com dados verdadeiramente reproduzíveis é ciência feita com códigos fontes e dados abertos, de uso irrestrito¹⁸. Por isso, todo o código desenvolvido para este trabalho será de domínio público, para que outros estudantes e pesquisadores possam aprender com o que foi pesquisado e que possam escrutinar o método proposto aqui.

4.1 Corpora

Os *corpora* de Cícero serão construídos de acordo com as regras de formatação e organização de outros *corpora* encontrados no projeto NLTK, que atualmente possui cerca de quarenta *corpora* prontos para uso. Isso facilitará bastante o processamento dos textos e também contribuirá com o projeto, para aumentar — criar, na realidade — sua base de textos em latim. Até o momento, não se conhece corpora de latim clássico compatível com o NLTK¹⁹.

¹⁸É digno de nota um excelente editorial sobre esse tema, originalmente publicado pela revista Nature, reproduzido em <http://arstechnica.com/science/news/2012/02/science-code-should-be-open-source-according-to-editorial.ars> pelo portal Ars Technica. No editorial é possível encontrar uma forte argumentação para o uso de plataformas *open source* de computação para o trabalho científico.

¹⁹Fez parte do protótipo deste trabalho, porém, a construção de corpora para PLN do latim clássico reconstruído pelo Nuntii Latini, disponível em <http://github.com/caio1982/Nuntii-Latini> para uso.

O apêndice A.1 demonstra uma ferramenta *scraper*, ou seja, para extração seletiva de textos online, que foi utilizada para coletar os textos do trabalho. Projetos como o PHI e Perseus disponibilizam suas bases apenas para visualização, logo foi necessário desenvolver um método para automatizar o processo que culmina nos corpora prontos para uso. A licença de uso justo para fins de pesquisa de ambos projetos acomoda o uso dessa ferramenta.

O escopo dos textos escolhido para compor os *corpora* deste trabalho abrange o período clássico da literatura latina, da qual Cícero é um dos maiores expoentes, senão o maior. A escolha de Cícero se deu por sua produção ter sido bastante variada, tendo ele escrito desde tratados até cartas pessoais, de textos jurídicos até filosofia. Cícero teve uma grande influência no pensamento moderno, e é ainda um dos autores mais estudados em cursos de latim, o que o torna ideal para uma análise linguística do latim.

Quanto ao tamanho dos *corpora*, que indiretamente determina a utilidade deles, em um teste preliminar chegou-se à marca de 80 mil termos. No teste, apenas alguns textos de Cícero facilmente encontráveis publicamente foram processados, sem muito método. De acordo com estimativas de Sardinha (2000a), isso já garante a este trabalho *corpora* de tamanhos razoáveis. O tamanho final dos *corpora* deverá ser muito maior que esse número, porém. Quase 1.4 milhão de tokens, sendo que destes aproximadamente 900 mil são termos relevantes (desconsiderando-se *stopwords* e pontuações). Finalmente, deste número se obtêm cerca de 82 mil termos únicos, ignorando-se repetições nos corpora.

Todavia, é importante salientar que os textos dos corpora não são originais. Naturalmente não era de se esperar que textos tão antigos sobrevivessem até hoje, logo é preciso considerar o trabalho de copistas ao lê-los. Isso significa que podem haver erros de grafia ou até edições nas construções das frases. Não é parte deste trabalho entrar nesses detalhes, mas vale a nota a respeito disto.

4.1.1 Especificação

A especificação dos arquivos dos corpora é bastante simples. O motivo para isso é permitir um manuseio mais direto dos dados, sem incluir complicações desnecessárias. O formato escolhido para criar os arquivos é o XML, um formato de documento extremamente flexível e simples²⁰. Sua principal vantagem sobre arquivos de texto puro é a agregação de metadados no meio do texto, mas sem poluí-lo. A seguir, um esqueleto

²⁰Ver <http://www.w3.org/XML/> para mais detalhes, a discussão sobre formatos de arquivos não está no escopo deste trabalho.

de arquivo de corpus exemplificando seu formato:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <document>
3   <meta name="author" value="marcus tullius cicero"/>
4   <meta name="title" value="X"/>
5   <meta name="source" value="http://latin.packhum.org/dx/text/474/Y/0"/>
6   <page id="Z">
7     <paragraph>
8       [...]
9     </paragraph>
10  </page>
11 </document>

```

A primeira linha de todos os corpora indica o formato XML, já mencionado, a codificação dos caracteres em UTF-8. A segunda indica o início do documento, mas não necessariamente o início do texto. As três linhas seguintes indicam metadados sobre o corpus, ou seja, informações de autoria, título e fonte. Cada seção do corpus está dividida em grupos *page* e cada parágrafo do texto em um grupo *paragraph*; podem haver múltiplos *paragraph* dentro de múltiplos *page*. As marcações *X*, *Y* e *Z* indicam valores que variam entre corpora, e a marcação *[...]* indica onde aparece o conteúdo textual.

Alguns textos possuem códigos misturados às palavras. Isso é uma forma encontrada para evitar problemas com símbolos gregos nos textos de Cícero (que são bastante presentes), ou para preservar sempre que possível as marcações de manuscritos²¹. Por exemplo, em *Topica* não se encontrará $\chi\rho\iota\nu\acute{o}\mu\epsilon\nu\omicron\nu$ como seria de se esperar, mas os códigos representando os símbolos. Embora ilegíveis para pessoas, qualquer programa de computador que suporte o formato XML conseguirá exibir os símbolos corretamente.

Para, a partir do NLTK, manipular os corpora montados neste trabalho, foi decidido re-escrever parte do carregador de corpora do NLTK. Foi criada uma classe de programação *CategorizedXMLCorpusReader* que deve ser utilizada. Os códigos nos apêndices do trabalho contêm exemplos de uso dela. Com a *CategorizedXMLCorpusReader* é possível utilizar corpora no formato XML separados por categorias e ao mesmo tempo ter as mesmas flexibilidades de corpora em texto puro.

O escopo está organizado em categorias que podem também ser manipuladas diretamente pelo NLTK, como no exemplo a seguir, permitindo um manuseio mais específico dos textos via programação. Naturalmente, a partir desse método também é possível fazer uma seleção particular de quais textos e quais categorias se deseja utilizar dos corpora:

²¹Ver <http://sites.google.com/site/homertheveryidea/resources/editorial-sigla-and-abbreviations> para exemplos.

```

1 >>> from nltk.corpus import cicero
2 >>> print cicero.categories()
3 ['letter', 'other', 'philosophy', 'poetry', 'rhetoric', 'speech', 'spurious']

```

Os apêndices do trabalho se encarregam de demonstrar com detalhes algumas formas possíveis de manuseio desses corpora. Entretanto, vale salientar que eles não precisam ser trabalhados sempre como um todo. Os corpora podem ser manuseados isoladamente também, como no exemplo a seguir, permitindo análises mais profundas em um ou outro texto de Cícero:

```

1 >>> from CatXMLReader import CategorizedXMLCorpusReader
2 >>> from nltk.corpus import cicero
3 >>>
4 >>> reader = CategorizedXMLCorpusReader(cicero.root, cicero.abspaths(), cat_file='categories.txt')
5 >>> list = ['cicero_in_verrem.xml']
6 >>> text = reader.words(list)
7 >>> print text[:10]
8 ['in', 'c', '.', 'uerrem', 'actio', 'prima', 'quod', 'erat', 'optandum', 'maxime']

```

4.1.2 Escopo

As listas a seguir foram construídas a partir de compilações populares de obras de Cícero — como a *Loeb Classical Library*, publicada pela Harvard University Press e o próprio PHI —, e consultando-se obras que tratam de Cícero ou da história da literatura latina de uma maneira geral — Conte e Solodow (1999), Kytzler (1989) e Paratore (1983). Todavia, ainda se notará a ausência de alguns textos. Entre estes textos está *Pro Titinia*, *Pro Acilio*, *Pro Gaio Antonio*, *De Temporibus Suis*, *De Consulatu Suo* e alguns outros que é desnecessário citar no momento. Tais textos são atribuídos a Cícero, estão completamente perdidos e pode-se somente encontrar referências a eles mas não ao seu conteúdo integral. É comum encontrar estas obras perdidas sob títulos como *Ciceronis Orationum Deperditarum*, e para mais detalhes recomenda-se ver Nisard (1869), Müller (1879), Crawford (1984) e também Harrison (1990).

É necessário também considerar textos dos quais temos somente fragmentos, como é o caso de *Hortensius* e algumas orações. O fato de serem apenas fragmentos influencia pouco nas análises deste trabalho, uma vez que caso surjam novas partes dos textos elas poderiam ser facilmente incorporadas aos outros corpora. Por isso, há poucas indicações explícitas sobre os fragmentos nas listas a seguir.

Entende-se ainda que não é objetivo deste trabalho realizar uma curadoria extensa de todos os textos de Cícero. Tal tarefa demandaria anos e especialização além do possível no momento. De qualquer maneira, é necessário considerar alguns problemas eventuais do escopo abaixo. Por exemplo, Kytzler (1989) menciona 58 discursos, mas não atesta todos eles. Ele também diz haver 20 textos filosóficos e que somente 13 sobreviveram. Entretanto, se considerarmos divisões de volumes de alguns livros, chega-se ao número mencionado, mas ele não explica isso em detalhes.

Acreditamos, por fim, que a lista de obras abaixo representa com orgulho um verdadeiro conjunto de corpora de Cícero para análise de LC e PLN, pesquisada e organizada com cuidado.

4.1.2.1 Discursos

Os discursos a seguir incluem tanto discursos de caráter jurídico quanto discursos políticos e de interesse público. Alguns, como *De Domo Sua* (que trata de uma petição pessoal de Cícero a um colegiado), são de difícil categorização pois podem ser interpretados tanto como discursos jurídicos quanto de outra natureza, ao mesmo tempo. Por essa razão, procurou-se evitar divisões excessivas entre os textos.

*De Domo Sua, De Haruspicum Responsis, De Lege Agraria Contra Rullum, De Provinciis Consularibus, Divinatio In Caecilium*²², *In Catilinam, In Pisonem, In Vatinium, In Verrem, Orationum Deperditarum Fragmenta, Orationum Incertarum Fragmenta, Philippicae, Post Reditum In Quirites*²³, *Post Reditum In Senatu*²⁴, *Pro Aemilio Scauro, Pro Archia Poeta, Pro Aulo Caecina, Pro Aulo Cluentio Habito, Pro Cornelio Balbo, Pro Gnaeo Plancio, Pro Lege Manilia*²⁵, *Pro Ligario, Pro Lucio Flacco, Pro Lucio Murena, Pro Marcello, Pro Marco Caelio, Pro Marco Fonteio, Pro Marco Tullio, Pro Milone, Pro Publio Quinctio, Pro Publio Cornelio Sulla, Pro Quinto Roscio Comoedo, Pro Rabirio Perduellionis Reo, Pro Rabirio Postumo, Pro Rege Deiotaro, Pro Sestio, Pro Sexto Roscio Amerino.*

²²Embora seja um texto de caráter claramente jurídico, no qual há um embate para se decidir entre quem será o acusador no caso contra Verres, Craig (1985) acredita que *Divinatio In Caecilium* pode ser compreendido também como um texto de retórica, principalmente pelo seu gênero de *divinatio* explicitado já no título.

²³Também aparece como *Post Reditum Ad Populum* ou *Cum Populo Gratias Egit*.

²⁴Também aparece como *Cum Senatui Gratias Egit*.

²⁵Também aparece como *De Imperio Gnaei Pompei*.

4.1.2.2 Retórica

*Ad Marcum Brutum Orator, Brutus, De Inventione, De Optimo Genere Oratorum, De Oratore, De Partitione Oratoria*²⁶, *Rhetorica Ad Herennium*²⁷, *Topica*.

4.1.2.3 Poesia

Arati Phaenomena, Arati Prognostica, Carmina.

4.1.2.4 Filosofia

*Academica Priora*²⁸, *Academica Posteriora*²⁹, *Cato Maior De Senectute, De Amicitia, De Divinatione, De Fato, De Finibus Bonorum Et Malorum, De Legibus, De Natura Deorum, De Officiis, De Re Publica, Hortensius, Paradoxa Stoicorum, Philosophicorum Librorum Fragmenta, Timaeus, Tusculanae Disputationes*³⁰.

4.1.2.5 Cartas

Epistulae Ad Atticum, Epistulae Ad Brutum, Epistulae Ad Familiares, Epistulae Ad Quintum Fratrem.

4.1.2.6 Outros

Nesta seção estão textos que não foram ainda categorizados por ninguém, ou que não se assemelham aos outros ou ainda devido ao seu tamanho por alguns serem fragmentos.

*Commentarii Causarum*³¹, *De Iure Civ. In Artem Redig.*³¹, *Epistula ad Oc-*

²⁶Aparece com frequência também como *Partitiones Oratoriae*, no plural.

²⁷Autoria disputada há muito tempo mas que certamente não é de Cícero, embora tanto o PHI quanto a Loeb Classical Library possuam o texto em seus catálogos junto com os de Cícero. Por completude, e devido ao seu estilo e vocabulário muito próximo ao de Cícero, o texto será considerado em algumas análises; o código que exibe concordâncias lexicais, por exemplo, permite selecionar esse texto. Para mais detalhes, ver a introdução de Harry Caplan à edição da própria Loeb.

²⁸Também citado como *Catulus*, são fragmentos da segunda parte do primeiro livro que compõe os *Academica* de Cícero.

²⁹Também citado como *Lucullus*, é a segunda parte dos *Academica*, aproximadamente quatro vezes maior do que restou da primeira parte. Para mais detalhes ver a introdução à tradução de Charles Duke Yonge (1875).

³⁰Do gênero *consolatio*, aparece com frequência também como *Tusculanae Quaestiones*. Não confundir o nome do gênero com o texto falsamente atribuído a Cícero. Para mais detalhes ver Sage (1910).

³¹Existem apenas pequenos fragmentos, atestados em *S. Hieronymi Presbyteri Opera* de Lardet e em *Iurisprudentiae Antejustinianae Reliquiae* de Huschke, Seckel e Kübler, respectivamente.

tavianum³², *Epistulae Fragmenta*³², *Facete Dicta*, *In Sallustium Crispum*³², *Incertorum Librorum Fragmenta*.

4.2 Stopwords

O catálogo de *stopwords* é uma dependência natural dos *corpora* pois é ele que filtra resultados indesejados ou que poderiam eventualmente poluir os dados. *Stopwords*, do ponto de vista de um computador, nada mais são que palavras ou agrupamentos³³ de palavras que interrompem o processamento da linguagem e modificam esse processamento de acordo com alguma regra pré-estabelecida. Toda língua possui uma lista de *stopwords* para processamento, porém o latim ainda não conta com uma verdadeiramente utilizável.

Embora pudessem ter sua criação automatizada por algoritmos de textos, catálogos de *stopwords* geralmente são construídos manualmente por linguistas. Logo, não existe um catálogo definitivo ou verdadeiramente completo para uma determinada língua, somente mais abrangentes e menos abrangentes, a depender do uso que será dado a eles.

Em computação, *stopwords* não são utilizadas somente para filtrar resultados indesejados ou que não são o foco da pesquisa linguística. Muitas vezes são termos repetitivos que levam a uma pior performance de um sistema e que se não forem ignorados diminuem a legibilidade dos resultados. Por exemplo, é bastante comum filtrar-se preposições, conjunções e pronomes, uma vez que eles pouco influenciam na maioria das análises computacionais de um texto e representam um conjunto de termos funcionais — de pouca relevância na maioria das análises — que podem não só poluir os dados do ponto de vista linguístico (SARDINHA, 2000a) mas também dificultar a análise computacional (MAKREHCHI; KAMEL, 2008). Todavia, é claro que tais filtros podem ser desligados se tais termos funcionais da língua forem de fato o alvo da pesquisa.

Para o latim clássico de Cícero foi criado um catálogo geral de *stopwords* (no modelo tradicional, uma lista finita de termos), e catálogos menores onde se encontrarão somente preposições, ou somente pronomes e assim por diante, para que os alunos e pesquisadores possam refinar melhor o uso de *stopwords* em latim. As listas a seguir foram

³²Embora façam parte de coleções de corpora como a do PHI, são textos espúrios, que ainda constam nestas coleções por serem de estilo muito próximo ao de Cícero ou por motivos específicos de cada projeto. Em *The Text of the Pseudo-Ciceronian Epistula Ad Octavianum*, W. S. Watt — responsável inclusive pelo texto que está no PHI — comenta sobre a relação entre os manuscritos falsos que levam ao texto que conhecemos hoje. Ver nota de *Rhetorica Ad Herennium* para detalhes de análise.

³³A linguística computacional também se refere a isso como *n-grams*, uma sequência de *n* termos textuais a serem processados conjuntamente e não individualmente.

elaboradas pelo autor a partir das gramáticas latinas de Leumann, Hofmann e Szantyr (1977) e Kühner (1912), e considerando resultados prévios deste trabalho comparando os filtros com os de outros autores³⁴.

As stopwords podem ser carregadas e filtradas, por exemplo, com um código similar ao abaixo:

```

1 >>> from nltk.corpus import cicero
2 >>> from nltk.corpus import stopwords
3 >>> stop = stopwords.words('latin')
4 >>> words = ['et', 'sum', 'loquor', 'cum', 'res']
5 >>> filtered = [x for x in words if x not in stop]
6 >>> print filtered
7 >>> ['loquor']

```

Todo o verbo *sum* está incluído nas *stopwords* (mas não listado abaixo) por sua enorme e óbvia frequência. Apenas 3 de suas 70 declinações possíveis não aparece nos corpora. Foram incluídas nas *stopwords* também todas as formas do adjetivo *omnis* e do substantivo *res*, por serem igualmente comuns (ao menos nos *corpora* de Cícero).

O apêndice A.5 contém a implementação de um testador para as *stopwords* abaixo. Ele foi desenvolvido para que se tivesse certeza que a) as *stopwords* fossem aplicáveis ao latim clássico, e não ao medieval, por exemplo e que b) cada uma tivesse ao menos uma ocorrência nos corpora, para evitar *stopwords* sem uso real.

4.2.1 Conjunções

Conjunções a serem filtradas: ac, alii, an, anne, antequam, as, ast, at, atque, atqui, aut, autem, certe, cum, deinde, denique, donec, dum, dummodo, enim, equidem, ergo, et, etenim, etiam, etiamsi, etsi, idcirco, ideo, igitur, insuper, item, licet, modo, nam, namque, ne, nec, necdum, necne, nempe, neque, neu, neue, ni, nimirum, nisi, num, postquam, priusquam, proinde, prout, quam, quamquam, quamvis, quanquam, quantumvis, quapropter, quasi, que, quia, quid, quidem, quippe, quo, quod, quominus, quomodo, quoniam, quoque, quum, saltem, sed, seu, si, sicut, sin, siquidem, siue, tamen, tametsi, tum, ue, uel, uerum, uerumtamen, uidelicet, ut, uti, utrum.

³⁴O projeto Perseus possui um catálogo de stopwords do Latim com cerca de 90 termos, porém os critérios para as escolhas feitas pelos responsáveis pelo projeto são desconhecidos, além de o próprio catálogo parecer incompleto.

4.2.2 Preposições

Preposições a serem filtradas: a, ab, abs, ad, aduersum, aduersus, ante, apud, circa, circiter, circum, cis, citra, clam, contra, coram, cum, de, dextra, e, erga, ex, extra, in, infra, inter, intra, iuxta, laude, ob, palam, penes, per, pone, post, prae, praeter, pro, prope, propter, quando, re, secundum, secus, sine, sub, subter, super, supra, tenus, trans, ultra.

4.2.3 Pronomes

Pronomes a serem filtrados: alia, aliae, aliam, aliarum, alias, alicui, alicuius, aliis, alio, aliorum, alios, aliqua, aliquae, aliquam, aliquarum, aliquas, aliquem, aliqui, aliquibus, aliquid, aliquis, aliquo, aliquorum, aliquos, aliud, alium, alius, cui, cuidam, cuinam, cuiquam, cuius, cuiusdam, cuiusnam, cuiusquam, cuiusque, cuiusvis, ea, eadem, eae, eam, eandem, earum, eas, ecquis, ego, egomet, ei, eidem, eis, eisdem, eius, eiusdem, eo, eodem, eorum, eorundem, eos, eosdem, eum, eundem, hac, hae, haec, hanc, harum, has, hi, hic, his, hoc, horum, hos, huic, huius, hunc, id, ii, iidem, iisdem, illa, illae, illam, illarum, illas, ille, illi, illic, illis, illius, illo, illorum, illos, illud, illum, ipsa, ipsae, ipsam, ipsarum, ipsas, ipse, ipsi, ipsis, ipsius, ipso, ipsorum, ipsos, ipsum, is, ista, istae, istam, istarum, istas, iste, isti, istis, istius, isto, istorum, istos, istud, istum, me, mea, meae, mearum, meas, mei, meis, meme, memet, meo, meorum, meos, meum, meus, mi, mihi, mihimet, nemine, neminem, nemini, nemo, nihil, nihili, nihilo, nobis, nobismet, nos, nosmet, noster, nostra, nostrae, nostram, nostrarum, nostras, nostri, nostris, nostro, nostrorum, nostros, nostrum, nulla, nullius, nullo, qua, quadam, quae, quaecumque, quaedam, quaelibet, quaeenam, quaequam, quaeque, quaevis, quale, qualem, quales, quali, qualia, qualibus, qualis, quam, quamnam, quanam, quandam, quarum, quarumnam, quarundam, quas, quasdam, quasnam, quem, quemcumque, quemnam, quempiam, quemquam, quendam, qui, quibus, quibusdam, quibusnam, quicquam, quicquid, quicumque, quid, quidam, quiddam, quidnam, quidpiam, quidquam, quidquid, quinam, quippiam, quis, quisnam, quispiam, quisquam, quisque, quisquis, quiuis, quo, quod, quodam, quodcumque, quoddam, quodnam, quodpiam, quonam, quoquam, quoquo, quorum, quorundam, quos, quoscumque, quosdam, se, sese, sibi, suarum, sui, suos, suus, te, tete, tibi, tu, tuae, tuam, tuarum, tuas, tui, tuis, tuo, tuorum, tuos, tute, tuus, uester, uestra, uestrae, uestram, uestrarum, uestras, uestri, uestris, uestro, uestrorum, uestros, uestrum, uniuscuiusque, unumquemque, uobis, uobismet, uos, uosmet, uostrae, uostri, uterque, utraque, utrique, utrisque, utrumque.

4.3 NLTK

Neste trabalho será utilizado o NLTK, um conjunto de ferramentas de programação voltado ao processamento de linguagem natural através de computação. Atualmente o NLTK é um dos projetos de maior sucesso para esse fim, pois, além de ser livre de licenças e custos, é bastante abrangente, cobre diversas línguas oficialmente e é disponível já com diversos *corpora* para testes. O projeto possui livros e publicações sobre seu funcionamento, tem mais de dez anos de desenvolvimento e é bastante utilizado em universidades há algum tempo³⁵, tornando-o ideal para uso uma vez que se provou estar em constante evolução acompanhando novas descobertas no campo de processamento de linguagem natural (BIRD; KLEIN; LOPER, 2009).

Uma característica do NLTK que levou à sua escolha como ferramenta deste trabalho é que ele foi desenvolvido pensando na interação entre professores, pesquisadores e alunos (BIRD; LOPER, 2004). É programado de forma simples e elegante. Chama-se o NLTK de uma plataforma Pythônica (MONTALENTI, 2011), isto é, que segue à risca os princípios da linguagem de programação Python, que será detalhada a seguir: os programadores do projeto NLTK não tentaram reinventar a roda recriando métodos de programação se a linguagem na qual ele foi criado já os possuía prontos. Além disso, o projeto é muito bem documentado e conta com comunidade aberta e disposta a ajudar quem se aventura com o NLTK. Tecnicamente falando, estas são vantagens consideráveis sobre quaisquer concorrentes.

TODO: introduzir os metodos a seguir e mencionar o que sao metodos de programacao e classes, e mencionar alguns legais do nltk para uso ou estudo futuro

4.3.1 Análise de frequência

Será usado o método de análise de frequência estatística como forma básica de obter os resultados de vocabulários dos *corpora*. Análise de frequência é uma forma de, através de pequenos experimentos automáticos (utilizando-se programação e tentando-se obter combinações de um termo em um texto, por exemplo), descobrir a proporção de uso de uma palavra qualquer em relação ao todo de um corpus específico. Essa é uma das atividades primordiais da LC, muitas vezes chamada simplesmente de estatística lexical por aderentes da linguística quantitativa.

³⁵Na última contagem, cerca de 2.000 trabalhos haviam sido publicados sobre o NLTK. Ver <http://scholar.google.com.au/scholar?q=NLTK> para uma lista completa.

Para chegar ao ponto de usar uma análise de frequência, é preciso antes passar pela montagem dos *corpora*, da criação de filtros e estipular outros parâmetros do processamento. Porém, uma vez que isso é feito, a análise de frequência se mostra relativamente simples e bastante proveitosa. É através da análise de frequência, por exemplo, que se pode tentar confirmar as Leis de Zipf para o latim clássico de Cícero, que estabelecem relações diretas entre a ordem de palavras em uma lista estatisticamente ordenada e a frequência de suas ocorrências em um corpus suficientemente grande. Em outras palavras, as leis de Zipf indicam que um número baixo de termos da língua corresponde a maior porção do uso efetivo da língua (TESITELOVA, 1992).

4.3.2 Concordâncias

Chama-se de concordâncias os extratos de um ou mais textos, geralmente de uma única linha, contendo uma palavra procurada alinhada verticalmente com todas as suas ocorrências nos *corpora*. O uso disso permite ajudar não só em um ganho de vocabulário por estudantes do latim, mas também para os pesquisadores que precisam de uma boa e imediata visualização do uso de determinadas palavras em textos. Embora existam livros impressos de concordâncias para os grandes autores (não só da antiguidade), estes são de difícil acesso por estudantes comuns, quando não são também caros.

Através do NLTK será demonstrado como pode-se visualizar e realizar buscas nos *corpora* de Cícero, de forma bastante simples e rápida, que pode ainda ser utilizada em sala de aula no ensino do latim clássico. A forma como concordâncias serão montadas é bastante particular a este trabalho, logo servirão também como exemplos de LA.

4.4 Python

Tanto o NLTK quanto ferramentas auxiliares deste trabalho serão feitos utilizando a linguagem de programação Python. A escolha dessa linguagem de programação se dá, entre outros motivos, pelo fato de o NLTK utilizá-la internamente para processar línguas naturais. Também, a facilidade de leitura dos código-fonte da linguagem por pessoas que não são da área de computação é um fator chave (PERKINS, 2010). Pelo fator de Python ser uma linguagem de alto nível, ou seja, de alta abstração ao se programar com ela, o desenvolvimento de programas é mais rápido com ela e permite que ele seja menos críptico para terceiros que precisem analisá-lo.

Criada por volta de 1989 por Guido van Rossum, a linguagem de programação

Python sempre teve como objetivo facilitar a criação de soluções rápidas e práticas em computação. Todo o seu desenvolvimento esteve centrado na máxima que diz que “programação de computadores pode ser feita por qualquer pessoa” (ROSSUM, 1999). Uma de outras suas grandes vantagens frente à outras linguagens de programação é sua extensibilidade, ou seja, sua capacidade de crescimento através de códigos de terceiros, que enriquecem-na. É precisamente esse o caso do NLTK.

Utilizar a linguagem de programação Python junto com o NLTK vai permitir uma autonomia muito grande para se criar, analisar e trabalhar de forma geral os *corpora* de latim clássico. Algumas alternativas para análise linguística de *corpora*³⁶ incluem os programas WordSmith Tools, Perseus Hopper (a versão web do projeto Perseus), Diogenes, Wmatrix, LanguageWare da IBM e LIWC. Alguns pesquisadores também optam por utilizar ferramentas de sistemas Unix e Linux, para criar o que se chama de *shell scripting*, para manipular textos e construir corpora. Algumas dessas ferramentas mais conhecidas são Sed, Grep, Awk, Sort, Wc e Deroff.

Infelizmente, quase todos esses programas possuem limitações tecnológicas, quando não de preço também. Utilizando a linguagem Python junto com o NLTK se tem uma possibilidade de desenvolvimento, customização e aprendizado muito grande. Digamos que se precise corrigir algum componente interno de um desses softwares, seja porque o erro é aparente, seja porque o pesquisador suspeita do seu comportamento. Isso não é possível com eles, mas é com a linguagem Python e o NLTK, tornando estes até mesmo mais cientificamente confiáveis.

³⁶Em <http://linguistlist.org/sp/GetWRListings.cfm?WRAbbrev=Software> há uma lista mais completa.

5 Análises

5.1 Stopwords

Uma vez que se tenha construído e organizado os *corpora*, é preciso remover os ruídos dos dados. No caso de um corpus linguístico esses ruídos são as *stopwords*. A necessidade de se filtrá-las dos *corpora* de Cícero se comprova através de alguns números. As *stopwords*, catalogadas nas seções anteriores, representam aproximadamente 33% de todas as ocorrências de palavras dos corpora de Cícero. Parece um número bastante alto, mas é até razoável se lembrarmos que no nosso catálogo elas são cerca de 500, um número expressivo. A porcentagem delas em um texto obviamente depende da sua abrangência, que nesse trabalho foi exagerada propositadamente pois *stopwords* não são o foco das análises. Para outras línguas ou pesquisas mais pontuais o número de *stopwords* irá variar bastante.

É comum que termos funcionais (preposições, conjunções etc) ocorram em grande número, muitas línguas dependem gramaticalmente desses termos. Todavia, ainda que isso possa indicar que o verdadeiro léxico a ser aprendido por estudantes de latim clássico seja os outros 67% livres de *stopwords*, é interessante interpretar os números por outro ângulo: por serem incrivelmente comuns, as *stopwords* obrigatoriamente precisam ser dominadas pelos estudantes de latim clássico antes de tentarem aumentar seu vocabulário, pois já garantiriam alguma familiaridade com boa parte dos textos existentes.

A partir do código do apêndice B.6 foi possível gerar um gráfico de frequência dos 100 termos mais utilizados por Cícero, incluindo *stopwords*. O código se baseia primariamente na classe de programação FreqDist, parte do módulo de probabilidade do NLTK. Basicamente, a FreqDist permite gerar distribuições de frequência a partir do resultado de um determinado evento, neste caso traduzível para “o termo X é encontrado N vezes no texto Z”. Tendo gerado uma distribuição de frequência dos termos de todos os corpora combinados, o resultado na figura abaixo comprova o nível de ruído que as *stopwords* criam.

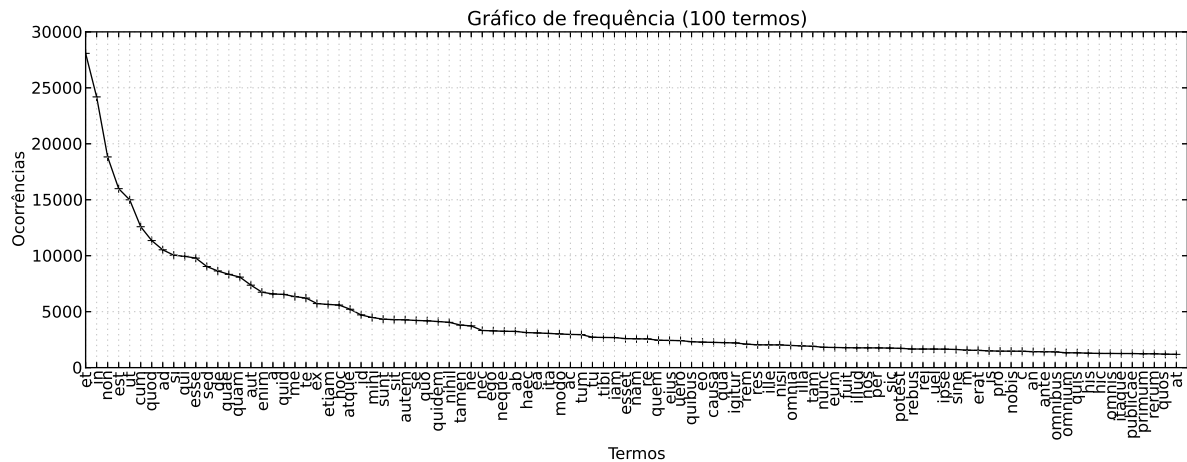


Figura 1: Gráfico de frequência (100 termos mais usados, incluindo stopwords)

O pico inicial de termos é formado principalmente por *stopwords*, o que significa muito pouco para uma análise lexical que queremos fazer. Entretanto, já podemos perceber, mesmo com *stopwords* presentes, uma curva acentuada que mostra um número cada vez menor de ocorrências para a maioria dos termos. Isso será analisado em breve. É possível reparar também termos de uma letra só. Isso não é um erro, mas uma anomalia dos corpora pelo uso de pré-nomes romanos abreviados. Assim, onde se vê *c* deve-se entender *gaius*, onde se vê *m* deve-se entender *marcus* e assim por diante, seguindo as convenções romanas.

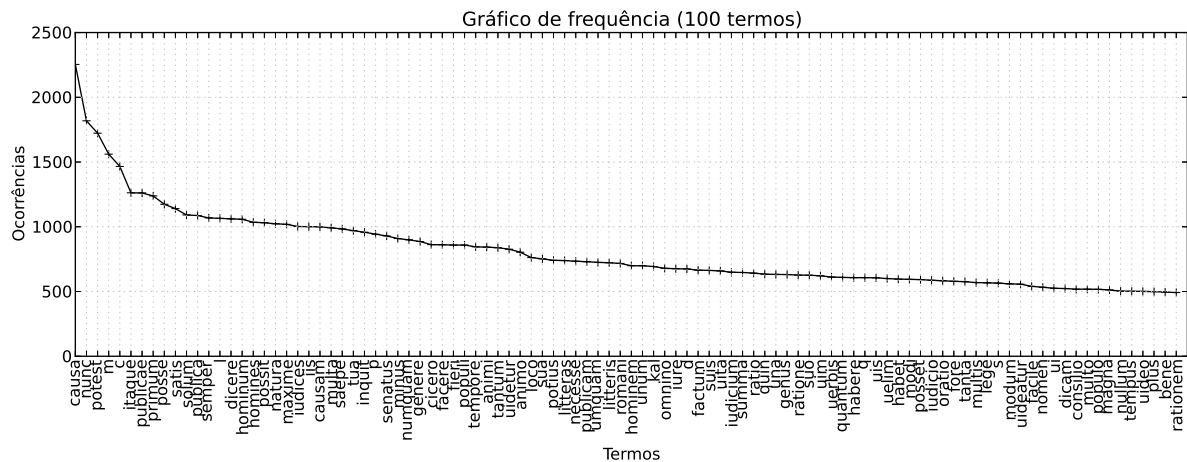


Figura 2: Gráfico de frequência (100 termos mais usados)

Excluindo-se as *stopwords*, enfim, chegamos a um gráfico de distribuição de frequência mais realista. A figura acima agora nos permite ver, por exemplo, que o termo mais usado nos corpora de Cícero seria na realidade um termo que antes apa-

recia no meio do gráfico. Nota-se ainda a diferença enorme no volume das primeiras frequências, demonstrando o predomínio das *stopwords* até então e a baixa frequência de termos “úteis”.

Entretanto, aqui temos um novo problema particular ao latim: o caráter flexional da língua faz com que declinações de um verbo ou casos de um nome tenham suas frequências calculadas separadamente. Isso não é de todo ruim, caso se esteja analisando as declinações e os casos. Neste trabalho, porém, podemos entender isso como um outro tipo de ruído nos dados, e que também será tratado em breve.

5.2 Lei de Zipf

Retomando o “fenômeno” da curva do gráfico na primeira figura, pode-se perceber nela uma semelhança dos corpora de Cícero com outros de outras línguas. Essa curva linear pode ser explicada pela Lei de Zipf. Por ser de ordem matemática, além do escopo desse trabalho, somente avaliaremos os resultados da lei, e não seu métodos. Para mais detalhes, incluindo uma explicação detalhada do funcionamento da Lei de Zipf e variações, recomenda-se o trabalho de Tesitelova (1992).

Dado um corpus suficientemente grande, alguns poucos termos serão utilizados em uma ordem de grandeza bem maior que a maioria dos outros termos deste corpus. Estatisticamente, quanto mais usado for um termo, maior será também a sua posição em uma lista ordenada de todos os termos do corpus, sendo que, em geral, o número de ocorrências de um termo cai pela metade a cada posição decrescente nessa mesma lista ordenada.

Pode-se dizer que isso leva à uma distribuição desproporcional do uso lexical. De fato leva, mas essa desproporção é sempre a mesma, em todas as línguas naturais. Logo, é possível dizer se um corpus é mal formado, filtrado demais, ou não representativo, se ele não bater com a Lei de Zipf. Li (2002) elenca uma série de ocorrências da Lei de Zipf, não só em línguas naturais, e demonstra como ela se faz válida independentemente da família linguística e até mesmo em corpora artificiais.

A figura a seguir é a aplicação da Lei de Zipf a todos os corpora deste trabalho. O gráfico está em escalas logarítmicas, e a linha tracejada aponta os valores ideais dos corpora segundo os cálculos da lei. A implementação da lei em código está no final do apêndice B.6. A parte específica dos cálculos segue abaixo, onde *ranks* e *freqs* são a ordem dos termos e sua ocorrências, respectivamente.

```

1 >>> ax.loglog(ranks, freqs, 'k-')
2 >>> ax.loglog(range(1, len(freqs)+1), [len(freqs)/x for x in range(1, len(freqs)+1)], 'k--')

```

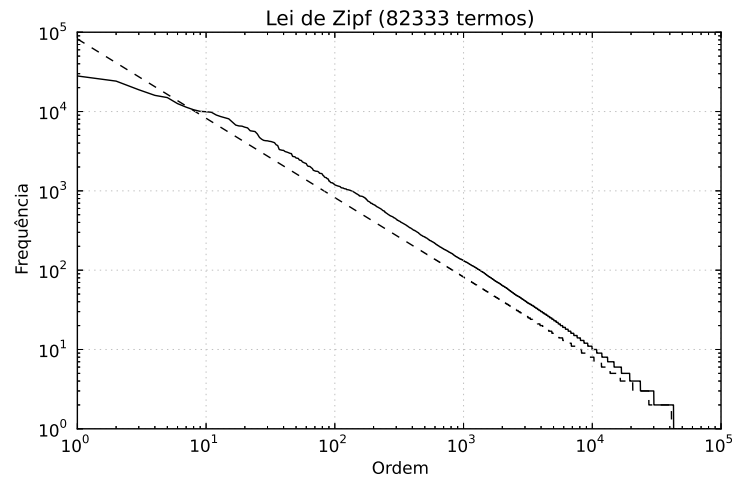


Figura 3: Gráfico atestando Lei de Zipf aos corpora

Pode-se ver claramente que os corpora do trabalho praticamente batem com os valores esperados deles. O resultado é bastante satisfatório, pois, embora a Lei de Zipf não resolva nenhum problema linguístico, ela é uma forma matemática eficiente para testar empiricamente um corpus de linguagem natural.

Se virtualmente toda língua natural, com um corpus suficientemente grande para amostragens estatísticas, irá bater com a Lei de Zipf, isso significa dizer que os corpora desse trabalho são plenamente utilizáveis em LC e linguística computacional dados os números e gráficos apresentados.

Um corpus contendo problemas de transcrição, muito ruído de *stopwords* etc dificilmente bateria com a Lei de Zipf dessa maneira. De fato, se analisarmos os corpora de Cícero individualmente, veríamos que alguns ficariam bastante “fora da curva”, como é o caso do pequeno *Academica*. Outros, como *Epistulae Ad Atticum*, batem perfeitamente por terem tamanhos consideráveis. Este gráfico apresenta a média geral dos corpora.

5.3 Radicalização

Chama-se de radicalização³⁷ o processo de se obter, através de algoritmos, a raiz de uma palavra. Isso pode ser feito morfológicamente, removendo seus afixos, ou semanti-

³⁷Stemming, em inglês, é um termo mais comum na literatura.

camente, reduzindo o termo ao seu sentido dicionarizado mais comum. Por exemplo, uma análise estatística de ocorrências de palavras poderia considerar não somente os termos como eles aparecem, mas seus *stems* morfológicos ou sentidos mais utilizados.

Com o latim isso seria particularmente interessante devido ao caráter bastante flexional da língua. Ao invés de se analisar palavras isoladamente, e manualmente trabalhando declinações e casos da língua, utilizando-se radicalização nas análises reduziria bastante a complexidade da LC com latim. Houve quem propusesse regras para radicalização em latim, mas elas nunca foram implementadas de fato (SCHINKE et al., 1996). Coube a Martin Porter implementar um algoritmo funcional utilizando a plataforma Snowball³⁸.

O apêndice A.7 demonstra uma outra proposta, que pode ser melhor desenvolvida no futuro. É uma re-implementação do algoritmo sugerido por Porter porém escrito na linguagem Python. Comparando os resultados do código aqui proposto com os de Porter a margem de erro não ultrapassa 2% em um corpus de aproximadamente 30 mil palavras, números consideráveis mas que podem ser melhorados no futuro. Para tal, seria necessário um tempo maior analisando resultados falso-positivos e refinando o algoritmo com referências gramaticais mais precisas do latim.

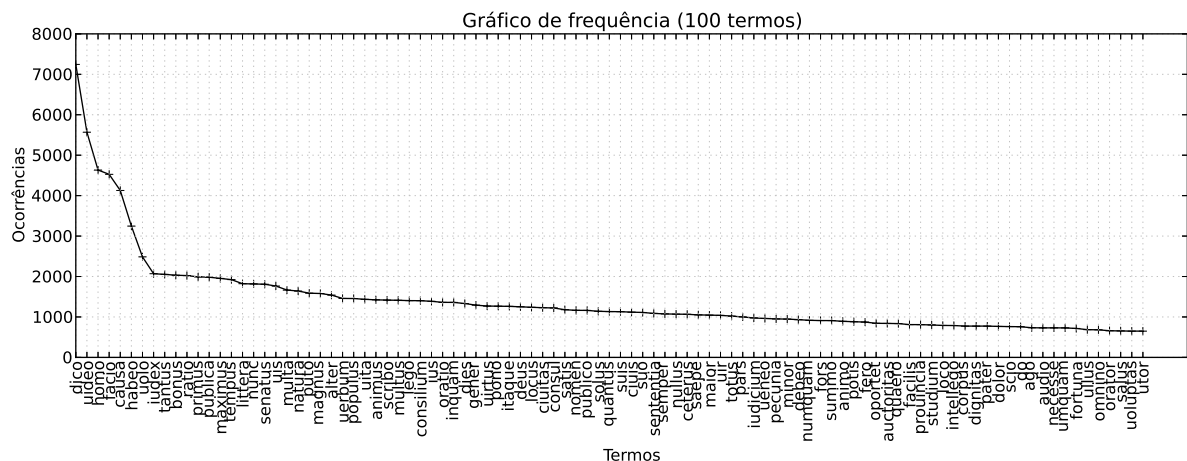
O apêndice A.8 também apresenta uma ferramenta simples de consulta ao Lem-Lat, um serviço online para lematização de palavras em latim do Istituto di Linguistica Computazionale, do CNR italiano. A diferença nesse caso é a preservação semântica de cada termo, reduzido a um lema original. Não fica-se restrito à operações morfológicas de remoção de afixos.

Os dois códigos apresentados nesses apêndices podem ser combinados para automatizar e melhorar o processo de radicalização de palavras do latim no futuro. Entende-se que estão longe do ideal em questão de performance e acuidade dos resultados, porém são um bom começo para quem desejar melhorá-los.

Embora ainda imperfeitos, os códigos desses dois apêndices nos permitem, finalmente, tentar resolver o problema particular ao latim clássico que interfere nas análises estatísticas: ser flexional demais. Finalmente podemos tentar gerar um gráfico a partir dos verdadeiros radicais dos termos mais usados.

Para a geração do gráfico com termos radicalizados, ocorrências de *video* e *wide-*

³⁸A versão do algoritmo de radicalização para latim de Porter jamais foi lançada oficialmente e permanece não testada. Ver <http://snowball.tartarus.org/otherapps/schinke/intro.html> para detalhes. Cabe também um agradecimento especial a Porter, por ter enviado por correio uma cópia impressa do artigo original de Schinke et al. (1996), que é excepcionalmente raro online.



analisado. O apêndice B.7 permite não só buscas simples mas filtradas também, levando em consideração textos espúrios de Cícero ou não. Além disso, ele permite indicar a profundidade da busca e exibição do termo, não só pelo número de ocorrências, mas também pelo tamanho do contexto lateral ao termo buscado. Todas as buscas, naturalmente, retornam o nome do corpus pesquisado ao lado da ocorrência.

Um exemplo simples de concordância, para o termo *animalis*, seria o seguinte:

```

1 ./concordances.py -w 90 -l animalis
2 >>> [Fin] [non enim iam stirpis bonum quaeret , sed animalis . quid , si non sensus modo ei sit datus]
3 >>> [Inv] [es alii genus , alii pars est . nam homo animalis pars est , thebani aut troiani genus . h]
4 >>> [ND] [dilatantur , ut frequenter ducatur cibus animalis , quo maxime aluntur animantes . ex inte]
5 >>> [Luc] [nes terram mare deos esse , quod quaedam animalis intellegentia per omnia ea permanet et t]
6 >>> [Tusc] [ reliquae duae partes , una ignea altera animalis , ut illae superiores in medium locum mu]

```

Nesse caso, a interpretação é simples pelo fato de *animalis* ter poucas ocorrências: o tamanho lateral das buscas (ou seja, contexto) foi limitado à 90 caracteres e somente 5 corpora possuem o termo, alinhado no centro em relação à todas as ocorrências.

Utilizadas como auxiliares para o aprendizado, concordâncias como as sugeridas aqui podem ser bastante eficientes em ensinar formas de uso reais dos termos de uma língua. Dúvidas como “o termo XYZ era utilizado de que forma, exatamente?” podem ser esclarecidas ao se ler Cícero. Concordâncias podem ser bastante úteis também para eliminar dúvidas quanto à ambiguidade de um termo. No exemplo, é possível saber quais *animalis* são genitivo e quais são nominativo, dado o mesmo caso do nome.

Existem dicionários publicados com concordâncias de autores e livros famosos, como Shakespeare e a bíblia católica, mas até o fim desse trabalho não encontrou-se um para Cícero. O código do apêndice B.7 poderia, em tese, ser utilizado para se criar um.

6 Conclusão

Este trabalho teve como objetivos criar possibilidades de LC e PLN com textos em latim clássico através da construção e catalogação de cerca de 75 corpora de Cícero, a fim de permitir novas análises e usos dos textos deste autor com auxílio de linguagens de programação. Além disso, procurou-se também a integração de mundos tão distintos como o da LA e o latim em sala de aula com a ajuda de ferramentas de acesso a esses corpora — cujo resultado direto é o léxico sugerido nos apêndices.

Qualquer trabalho sobre LC ou PLN jamais chegará a um resultado imutável e perfeito. O tempo e avanços em computação impedem que se indique um fim em uma pesquisa como essa. Embora os resultados dos métodos propostos aqui tenham clara utilidade em sala de aula, ainda há muito que melhorar. Foram criadas aproximadamente 1.000 linhas de códigos de programação que sustentam as análises do trabalho. Algumas tiveram uso direto, algumas são de referência.

Todos os códigos, corpora e arquivos auxiliares deste trabalho estão disponíveis em domínio público online, através do endereço <http://caio.ueberalles.net/cicero>. Iniciados em programação notarão diversos pontos para melhorias nos códigos. De fato, contribuições de terceiros são mais do que bem-vindas, pois o trabalho de PLN nos corpora de Cícero não termina com aqui e certamente não é uma tarefa para uma pessoa só.

Este trabalho também serviu para apresentar o NLTK e demonstrar o quão poderoso ele é em LC. Apenas uma parte mínima do projeto foi utilizada aqui. O NLTK pode ser muito mais utilizado em sala de aula, por exemplo. Pode-se, à partir do que foi apresentado neste trabalho, por exemplo, construir palavras cruzadas com vocabulário de latim clássico. Pode-se, também, criar ferramentas para testar gramáticas e exercícios de sala de aula, ou ainda investigar mudanças históricas do latim com corpora de períodos distintos. Os possíveis usos do NLTK a partir daqui são enormes.

Espera-se que com este trabalho o mundo clássico do latim — e por que não, de outras línguas clássicas também —, se aproxime mais dos avanços em LC e PLN.

Referências

- ABNEY, S. *Statistical Methods and Linguistics*. Cambridge: MIT, 1996.
- ALMEIDA, R. *Palavras mais comuns em inglês*. [S.l.]: Dicas-L, 1997. http://www.dicas-l.com.br/arquivo/palavras_mais_comuns_em_ingles.php. Acessado em 5 de novembro de 2011.
- ALMEIDA, R. *As Palavras Mais Comuns da Língua Inglesa*. Campinas: UNICAMP, 2003.
- BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Processing with Python*. Beijing: O'Reilly Media, 2009.
- BIRD, S.; LOPER, E. *NLTK: The Natural Language Toolkit*. Barcelona: Association for Computational Linguistics, 2004.
- CLACKSON, J. *Indo-European Linguistics: An Introduction*. Cambridge: Cambridge University Press, 2007.
- CNPQ. *Diretório dos Grupos de Pesquisa no Brasil*. [S.l.]: Conselho Nacional de Desenvolvimento Científico Tecnológico, 2012. <http://dgp.cnpq.br/buscaoperacional/>. Acessado em 14 de fevereiro de 2012.
- CONTE, G. B.; SOLODOW, J. *Latin Literature: A History*. Baltimore: Johns Hopkins University Press, 1999.
- CRAIG, C. P. *Dilemma in Cicero's Divinatio in Caecilium*. Baltimore: The Johns Hopkins University Press, 1985.
- CRAWFORD, J. *Marcus Tullius Cicero: The lost and unpublished orations*. Göttingen: Vandenhoeck & Ruprecht, 1984.
- DEE, J. *A Dual-Source Database of Word Frequencies in Latin*. [S.l.]: University of Illinois, 2000. http://www.uic.edu/las/clas/LF_database.html. Acessado em 19 de abril de 2012.
- DIEDERICH, P. *The Frequency of Latin Words and Their Endings*. Chicago: University of Chicago Press, 1939.
- EVERITT, A. *Cicero: the life and times of Rome's greatest politician*. New York: Random House, 2001.
- FISHWICK, M. W. *Cicero, classicism, and popular culture*. Binghamton: The Haworth Press, 2007.

- FRANZ, A.; BRANTS, T. *All Our N-gram are Belong to You*. [S.l.]: Google Machine Translation Team, 2006. <http://googleresearch.blogspot.com.br/2006/08/all-our-n-gram-are-belong-to-you.html>. Acessado em 17 de janeiro de 2012.
- FURMAN, N.; GOLDBERG, D.; LUSIN, N. *Enrollments in Languages Other Than English in United States Institutions of Higher Education, Fall 2009*. New York: Modern Language Association of America, 2010. http://www.mla.org/pdf/2009_enrollment_survey.pdf. Acessado em 22 de novembro de 2011.
- HARRISON, S. J. *Cicero's De Temporibus Suis: The Evidence Reconsidered*. Wiesbaden: Hermes, 1990.
- JACOBI, C. Dissertação de mestrado. *Linguística de Corpus e ensino de espanhol a brasileiros: Descrição de padrões e preparação de atividades didáticas*. São Paulo: PUC/SP, 2001.
- KENNEDY, G.; WOOTEN, C. *The orator in action and theory in Greece and Rome*. Leiden: Brill, 2001. (225).
- KENNEY, E. J.; CLAUSEN, W. V. *The Cambridge History of Classical Literature: Latin Literature*. Cambridge: Cambridge University Press, 1983.
- KÜHNER, R. *Ausführliche Grammatik der lateinischen Sprache*. Hannover: Hahnsche Buchhandlung, 1912.
- KYTZLER, B. *Breve diccionario de autores griegos y latinos*. Madrid: Gredos, 1989.
- LADEIRA, A. P. *Processamento de linguagem natural: caracterização da produção científica dos pesquisadores brasileiros*. Belo Horizonte: UFMG, 2010. Tese de doutorado.
- LEUMANN, M.; HOFMANN, J.; SZANTYR, A. *Lateinische Grammatik*. Munich: Beck, 1977.
- LI, W. *Zipf's Law Everywhere*. Lüdenscheid: Glottometrics, Volume 5, 2002.
- MAKREHCHI, M.; KAMEL, M. *Automatic Extraction of Domain-Specific Stopwords from Labeled Documents*. Proceedings of ECIR - 30: Advances in information retrieval, 2008.
- MAY, J.; CRAIG, C. *Brill's companion to Cicero, oratory and rhetoric*. Leiden: Brill, 2002.
- MONTALENTI, A. *Just Enough NLP with Python*. [S.l.]: Pixel Monkey, 2011. <http://pixelmonkey.org/pub/nlp-training>. Acessado em 20 de março de 2012.
- MÜLLER, C. F. W. M. *Tulli Ciceronis scripta quae manserunt omnia*. Leipzig: B.G. Teubner, 1879.
- NISARD, D. *Œuvres complètes de Cicéron, avec la traduction en français*. Paris: Institut de France, 1869.
- NORVIG, P. *On Chomsky and the Two Cultures of Statistical Learning*. [S.l.: s.n.], 2011. <http://norvig.com/chomsky.html>. Acessado em 7 de janeiro de 2012.

- OCH, F. J. *Statistical Machine Translation: Foundations and Recent Advances*. Phuket: Google, 2005.
- PALMER, L. *The Latin Language*. Norman: University of Oklahoma Press, 1988.
- PARATORE, E. *História da Literatura Latina*. Lisboa: Fundação Calouste Gulbenkian, 1983.
- PELLEGRINO, D. *Cicero and Horace Vocabulary Frequency Lists for AP Selections*. Wauconda: Bolchazy-Carducci, 2009.
- PERKINS, J. *Python Text Processing with NLTK Cookbook*. Birmingham: Packt Publishing, 2010.
- PETERSSON, T. *Cicero: A Biography*. Berkeley: Kessinger Publishing, 2005.
- ROSSUM, G. van. *Computer Programming for Everybody*. Reston: Corporation for National Research Initiatives, 1999. <http://www.python.org/doc/essays/cp4e.html>.
- SAGE, E. T. *The pseudo-Ciceronian Consolatio*. Chicago: University of Chicago Press, 1910.
- SAPIR, E. *Culture, language and personality*. Berkeley: University of California Press, 1949.
- SARDINHA, T. B. *Comparing corpora with WordSmith Tools: How large must the reference corpus be?* Stroudsburg: WCC, 2000.
- SARDINHA, T. B. *Computador, corpus e concordância no ensino da léxico-gramática de língua estrangeira*. Pelotas: Educat/ALAB, 2000.
- SARDINHA, T. B. *Linguística de corpus*. Barueri: Manole, 2004.
- SCHINKE, R. et al. *A stemming algorithm for Latin text databases*. Sheffield: Journal of Documentation, Volume 52, 1996.
- SINCLAIR, J. M. *The automatic analysis of corpora*. Estocolmo: Directions in corpus linguistics, 1991.
- TESITELOVA, M. *Quantitative linguistics*. Amsterda: John Benjamins, 1992.
- WILLIS, D. *The Lexical Syllabus, a new approach to language teaching*. London: Collins, 1990.

APÊNDICE A – Sugestão de léxico para estudo

dico	uerbum	quantus	oportet	lex	anima
possum	populus	suis	auctoritas	modius	liber
uideo	uita	ciuis	quaero	ars	exercitus
homo	animus	suo	facilis	arbitror	reliquus
facio	tuus	suus	prouincia	quin	praetor
causa	scribo	sententia	studium	genus	honor
habeo	multus	semper	loco	uirus	diligo
uolo	lego	nullus	intellego	plus	bene
iudex	consilium	ceterus	corpus	periculum	bellum
tantus	ius	saepe	dignitas	optimus	sentio
bonus	oratio	maior	pater	soleo	domus
ratio	inquam	uir	dolor	scriba	sapio
primus	dies	totus	scio	opus	rego
publica	gener	pars	do	dica	diu
maximus	uirtus	iudicium	ago	modus	imperium
tempus	pono	ueneo	necesse	proficiscor	plures
littera	itaque	pecunia	audio	mors	accipio
nunc	deus	minor	umquam	spes	praesertim
senatus	locus	debeo	fortuna	urbs	uno
uis	ciuitas	numquam	ullus	pario	malis
multa	consul	fors	omnino	duo	uoluntas
natura	satis	summo	orator	gratia	amicus
puto	nomen	animo	salus	memoria	moueo
magnus	publico	potis	utor	credo	mens
alter	solus	fero	uoluptas	populor	postea

fides	antea	paene	amico	prior	contineo
crimen	terra	statim	exter	laudo	iter
paullus	testis	crassus	manus	exspecto	consto
sermo	bello	peto	constituo	nolo	frumentum
brutus	melior	nosco	officio	usque	finis
patria	aio	plurimus	socius	conscribo	cumque
iniuria	nego	pertineo	uer	contrarius	dicis
sal	loquor	uerres	sol	philosophia	quaeso
gloria	mорий	tandem	numero	brutes	absum
potestas	honestus	dedo	accusator	spero	quirito
epistula	defendo	consulo	accedo	summa	concedo
sanus	mitto	publicus	commodus	foro	accuso
aetas	ualde	uix	puteo	praesidium	audacia
ager	aliquando	domio	intersum	eloquentia	mare
malo	existimo	communio	magistratus	castrum	disciplina
filius	immortalis	breuis	sex	contio	diuino
fere	annus	libertas	amplus	arator	alienus
cognosco	uiuo	iudico	nonne	libet	sicilis
amicitia	bellis	familiaris	iubeo	adulesco	legio
frater	minimus	casus	falsum	mundus	inuenio
efficio	summus	liberi	tribunus	ostendo	quaestio
grauis	italia	affero	praeclarus	pridie	miles
appello	quintus	unde	reus	uenio	consequor
uehemens	placeo	censeo	caput	cottidie	laus
planus	princeps	genero	paco	mehercule	defensio
ingenius	scelerus	sequor	consulatus	tabula	turpis
licet	gero	cupiditas	ciuilis	scientia	dubito
negotium	beneficium	testimonium	opinor	diligentia	decimus
beo	adhuc	dolabella	uera	caelum	communis
tamquam	metus	gratus	filia	nimis	praeceptum
opera	rex	orbis	legatus	asia	clarus
ualeo	consuetudo	conuenio	miser	similis	multitudo
hostis	certus	accido	auctor	labor	magnitudo
arma	praeterea	talis	amor	dignus	conficio
nescio	religio	philosophus	plebs	odeum	praesto
nascor	longus	ordo	imperator	prudencia	duco
simul	tot	agero	opinio	uaco	metellus
cura	uox	inimicus	cogito	adsum	fortis
officium	oculus	utilitas	sapientia	actio	honestas
copia	partim	facia	doceo	poena	aegritudo

exemplum	numerus	dato	quod-si	puer	occido
suspicio	ordino	edico	teneo	consularis	uictoria
ulciscor	contentio	adeo	superior	demonstro	cognitio
argumentum	inuidia	relinqo	constitutio	catulus	singula
probo	exemplo	libido	rogus	argumentatio	senectus
ibi	par	pauci	caedes	instituo	fructus
forma	acta	acte	insum	hortens	hortus
lentulus	similitudo	nauis	subeo	platon	nouo
facultas	familia	otion	adhibeo	curro	singularis
inde	nimius	scilicet	considero	mos	datus
aequus	culpa	necessarius	gens	redeo	reddo
domo	societas	utinam	uerso	adno	cerno
aduersarius	scipio	simile	hinc	apio	grauitas
dea	sisto	nuper	uia	diuinatio	contendo
controuersia	exeo	hir	perspicuus	species	humanitas
tribus	plerusque	quoad	percipio	templum	popularis
utilis	uitio	uereor	amo	aedes	necessitudo
cupio	auspicium	marcus	dubius	armatus	quattuor
assentio	difficilis	dis	balbus	collega	mater
totus	iustitia	aperio	signo	careo	pando
nondum	mulier	postulo	iuro	cado	seruus
beneuolentia	coepio	istic	calamitas	auris	coniungo
sumo	graecor	improbis	fides	absoluo	ferrum
pompeum	doctrina	fama	molestus	condemno	forum
cio	parum	cato	cipio	eques	facinus

APÊNDICE B – Códigos

B.1 Scraper de dados e montador dos corpora

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  # to add some sleep time between fetches
7  # and not hammer down the server
8  import time
9
10 # to avoid charsetting mess with UTF-8 strings
11 import codecs
12 import string
13
14 # to check if a text was already fetched
15 from os.path import exists
16
17 # document parser
18 from lxml import etree
19
20 # output generation
21 from elementtree.SimpleXMLWriter import XMLWriter
22
23 refs = []
24 urls = []
25
26 # the root directory of all latin texts in PHI
27 base = 'http://latin.packhum.org/'
28
29 # marcus tullius cicero entry in PHI
30 browse = base + 'author/474'
31
32 try:
33     page = etree.parse(browse, etree.HTMLParser(encoding='utf-8'))
34 except Exception, err:
35     print 'Browse Error: ' + str(err)
36
37 # gets the list of texts by cicero currently in PHI
38 matches = page.xpath("//span[@class='unam']/text()")
39

```

```

40 counter = 1
41 for entry in matches:
42     # creates a reference list with download addresses for every text
43     refs.append((base + 'dx/text/474/%s/' % str(counter), entry.lower()))
44     counter += 1
45
46 for param in refs:
47     source = param[0]
48     title = param[1]
49
50     # debug
51     print '[%s]' % title
52
53     filename = title.replace(' ', '_')
54     for p in string.punctuation.replace('_', ''):
55         filename = filename.replace(p, '')
56
57     w = XMLWriter('cicero_' + filename + '.xml', encoding='utf-8')
58     xml = w.start("document")
59
60     # metadata entries of the output files
61     w.element("meta", name="author", value="marcus tullius cicero")
62     w.element("meta", name="title", value=title)
63     w.element("meta", name="source", value=source + '0')
64
65     # upon checking it no text in PHI attributed to cicero
66     # has more than 500 pages, so this is a safe download limit
67     for x in range(0, 500):
68         lines = []
69         entry = []
70         section = source + str(x)
71         reference = base + 'loc/474/' + str(x) + '/0'
72
73         # debug
74         print '\t<%s>' % section
75
76         # output filename
77         path = 'ready/' + filename + '-' + str(x) + '.txt'
78
79         if not exists(path):
80             # fetches the current page
81             try:
82                 page = etree.parse(section, etree.HTMLParser(encoding='utf-8'))
83             except Exception, err:
84                 print 'Text Error: ' + str(err)
85
86             # parses the page paragraphs
87             try:
88                 entry = page.xpath("//tr/td[1]//text() | //h3//text()")
89             except Exception, err:
90                 print 'Match Error: ' + str(err)
91                 # a priori this is not needed but it is helpful for debugging
92                 f = codecs.open("log.txt", "a", "utf8")
93                 f.write('\nMatch Error: ' + str(err) + ' [missing] ' + section)

```

```

94         f.close()
95
96         # checks if the end of text has been reached
97         if 'No text' in entry:
98             print 'EOF: ' + str(x)
99             break
100
101         empty = u'\xa0\xa0'
102         if len(entry) > 0:
103             for e in entry:
104                 if e.startswith(empty):
105                     # apparently PHI texts have double blank spaces indicating new paragraphs
106                     lines.append(''.join(e.replace(empty, '')))
107                 else:
108                     lines.append(''.join(e))
109
110         paragraph = ' '.join(lines)
111         y = codecs.open(path, "w", "utf8")
112         y.write(paragraph)
113         y.write
114     else:
115         # if text has been fetched ok, process it
116         paragraph = codecs.open(path, "r", "utf8")
117         strings = paragraph.read()
118
119         # finally writes the new content to the corpus file
120         w.start("page", id=str(x))
121         w.element("paragraph", strings)
122         w.end("page")
123
124         # give the PHI server some time until the next fetch
125         # time.sleep(5)
126     # generates the output file
127     w.close(xml)

```

B.2 Filtros dos corpora

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  import codecs
7  import glob
8
9  def normalizer():
10     for loop in glob.glob('raw/*.txt'):
11         text = ''
12         with file(loop, 'r') as content:
13             text = content.read().lower()
14

```

```

15     # fixes the linebreaking of corpora
16     text = text.replace("- ", "")
17
18     # fixes unused letters for their real latin ones
19     text = text.replace("v", "u")
20     text = text.replace("j", "i")
21
22     with file(loop.replace('raw/', 'ready/'), 'w') as content:
23         content.write(text)
24
25 if __name__ == "__main__":
26     normalizer()

```

B.3 Classe de manipulação dos corpora

```

1  # http://stackoverflow.com/questions/6849600/does-anyone-have-a-categorized-xml-corpus-reader-for-nltk
2
3  # standard nltk classes
4  from nltk.corpus.reader import CategorizedCorpusReader
5  from nltk.corpus.reader import XMLCorpusReader
6
7  # stopwords (i.e. latin ones)
8  from nltk.corpus import stopwords
9
10 # for CategorizedCorpusReader's init
11 from nltk.compat import defaultdict
12
13 # punctuations load
14 import string
15
16 class MyCategorizedCorpusReader(CategorizedCorpusReader):
17     def _init(self):
18         self._f2c = defaultdict(set)
19         self._c2f = defaultdict(set)
20
21         if self._pattern is not None:
22             for file_id in self._fileids:
23                 category = re.match(self._pattern, file_id).group(1)
24                 self._add(file_id, category)
25
26         elif self._map is not None:
27             for (file_id, categories) in self._map.items():
28                 for category in categories:
29                     self._add(file_id, category)
30
31         elif self._file is not None:
32             for line in self.open(self._file).readlines():
33                 line = line.strip()
34                 file_id, categories = line.split(self._delimiter, 1)
35                 # https://github.com/nltk/nltk/issues/250
36                 #if file_id not in self.fileids():

```

```

37         # raise ValueError('In category mapping file %s: %s '
38         #                     'not found' % (self._file, file_id))
39         for category in categories.split(self._delimiter):
40             self._add(file_id, category)
41
42
43 class CategorizedXMLCorpusReader(MyCategorizedCorpusReader, XMLCorpusReader):
44     def __init__(self, *args, **kwargs):
45         MyCategorizedCorpusReader.__init__(self, kwargs)
46         XMLCorpusReader.__init__(self, *args, **kwargs)
47
48     def _resolve(self, fileids, categories):
49         if fileids is not None and categories is not None:
50             raise ValueError('Specify fileids or categories, not both')
51         if categories is not None:
52             return self.fileids(categories)
53         else:
54             return fileids
55
56     def raw(self, fileids=None, categories=None):
57         return XMLCorpusReader.raw(self, self._resolve(fileids, categories))
58
59     def words(self, fileids=None, categories=None):
60         words = []
61         fileids = self._resolve(fileids, categories)
62         for fileid in fileids:
63             words += XMLCorpusReader.words(self, fileid)
64         return words
65
66     def text(self, fileids=None, categories=None):
67         fileids = self._resolve(fileids, categories)
68         text = ""
69         for fileid in fileids:
70             for i in self.xml(fileid).getiterator():
71                 if i.text:
72                     text += i.text
73         return text
74
75     def sents(self, fileids=None, categories=None):
76         text = self.words(fileids, categories)
77         sents = nltk.PunktSentenceTokenizer().tokenize(text)
78         return sents
79
80     def paras(self, fileids=None, categories=None):
81         return CategorizedCorpusReader.paras(self, self._resolve(fileids, categories))
82
83
84 def stopless(wordslist):
85     stop = stopwords.words('latin')
86     filtered = [x for x in wordslist if x not in stop]
87     return filtered
88
89 def punctless(wordslist):
90     punct = string.punctuation

```

```

91 punct += u'\u00a7' # SECTION SIGN
92 punct += u'\u00b3' # SUPERSCRIFT THREE
93 punct += u'\u00b2' # SUPERSCRIFT TWO
94 punct += u'\u00b7' # MIDDLE DOT
95 punct += u'\u00b9' # SUPERSCRIFT ONE
96 punct += u'\u2014' # EM DASH
97 punct += u'\u2019' # RIGHT SINGLE QUOTATION MARK
98 punct += u'\u2020' # DAGGER
99 punct += u'\u2184' # LATIN SMALL LETTER REVERSED C
100 punct += u'\u221e' # INFINITY
101 punct += u'\u23d1' # METRICAL BREVE
102
103
104 punctuation = list(punct)
105
106 words = []
107 for w in wordslist:
108     if w.isalpha():
109         words.append(w)
110
111 filtered = [x.encode('utf-8', 'replace') for x in words if x not in punctuation]
112 return filtered
113
114 def ciceroabbr(filename):
115     return {
116         'cicero_academica.xml': 'Ac',
117         'cicero_arati_phaenomena.xml': 'AratPhaen',
118         'cicero_arati_prognostica.xml': 'AratProgn',
119         'cicero_brutus.xml': 'Brut',
120         'cicero_carmina_fragmenta.xml': 'CarFrr',
121         'cicero_cato_maior_de_senectute.xml': 'Sen',
122         'cicero_commentarii_causarum.xml': 'CommCaus',
123         'cicero_de_divinatione.xml': 'Div',
124         'cicero_de_domo_sua.xml': 'Dom',
125         'cicero_de_fato.xml': 'Fat',
126         'cicero_de_finibus.xml': 'Fin',
127         'cicero_de_haruspicum_responso.xml': 'Har',
128         'cicero_de_inventione.xml': 'Inv',
129         'cicero_de_iure_civ_in_artem_redig.xml': 'IurCiv',
130         'cicero_de_lege_agraria.xml': 'Agr',
131         'cicero_de_legibus.xml': 'Leg',
132         'cicero_de_natura_deorum.xml': 'ND',
133         'cicero_de_officiis.xml': 'Off',
134         'cicero_de_optimo_genere_oratorum.xml': 'OptGen',
135         'cicero_de_oratore.xml': 'DeOrat',
136         'cicero_de_partitione_oratoria.xml': 'Part',
137         'cicero_de_provinciis_consularibus.xml': 'Prov',
138         'cicero_de_republica.xml': 'Rep',
139         'cicero_epistula_ad_octavianum_sp.xml': 'EpOct',
140         'cicero_epistulae_ad_atticum.xml': 'Att',
141         'cicero_epistulae_ad_brutum.xml': 'AdBrut',
142         'cicero_epistulae_ad_familiares.xml': 'Fam',
143         'cicero_epistulae_ad_quintum_fratrem.xml': 'Qfr',
144         'cicero_epistulae_fragmenta.xml': 'EpFrr',

```



```

145      'cicero_facete_dicta.xml': 'Facet',
146      'cicero_hortensius.xml': 'Hort',
147      'cicero_in_catilinam.xml': 'Catil',
148      'cicero_in_pisonem.xml': 'Pis',
149      'cicero_in_q_caecilium.xml': 'DivCaec',
150      'cicero_in_sallustium_sp.xml': 'Sal',
151      'cicero_in_vatinium.xml': 'Vat',
152      'cicero_in_verrem.xml': 'Ver',
153      'cicero_incertorum_librorum_fragmenta.xml': 'LibFrr',
154      'cicero_laelius_de_amicitia.xml': 'Amic',
155      'cicero_lucullus.xml': 'Luc',
156      'cicero_orationum_deperditarum_frr.xml': 'DepFrr',
157      'cicero_orationum_incertarum_frr.xml': 'IncFrr',
158      'cicero_orator.xml': 'Orat',
159      'cicero_paradoxa_stoicorum.xml': 'Parad',
160      'cicero_philippicae.xml': 'Phil',
161      'cicero_philosophicorum_librorum_frr.xml': 'PhilFrr',
162      'cicero_post_reditum_ad_populum.xml': 'RedPop',
163      'cicero_post_reditum_in_senatu.xml': 'RedSen',
164      'cicero_pro_archia.xml': 'Arch',
165      'cicero_pro_balbo.xml': 'Balb',
166      'cicero_pro_caecina.xml': 'Caec',
167      'cicero_pro_caelio.xml': 'Cael',
168      'cicero_pro_cluentio.xml': 'Clu',
169      'cicero_pro_flacco.xml': 'Flac',
170      'cicero_pro_fonteio.xml': 'Font',
171      'cicero_pro_lege_manilia.xml': 'Man',
172      'cicero_pro_ligarario.xml': 'Lig',
173      'cicero_pro_marcello.xml': 'Marc',
174      'cicero_pro_milone.xml': 'Mil',
175      'cicero_pro_murena.xml': 'Mur',
176      'cicero_pro_plancio.xml': 'Planc',
177      'cicero_pro_q_rosco_comoedo.xml': 'QRosc',
178      'cicero_pro_quinctio.xml': 'Quinct',
179      'cicero_pro_rabirio_perduellionis_reo.xml': 'RabPerd',
180      'cicero_pro_rabirio_postumo.xml': 'RabPost',
181      'cicero_pro_rege_deiotaro.xml': 'Deiot',
182      'cicero_pro_s_rosco_amerino.xml': 'SRosc',
183      'cicero_pro_scauro.xml': 'Scaur',
184      'cicero_pro_sestio.xml': 'Sest',
185      'cicero_pro_sulla.xml': 'Sul',
186      'cicero_pro_tullio.xml': 'Tul',
187      'cicero_rhetorica_ad_herennium_sp.xml': 'RhetHer',
188      'cicero_timaeus.xml': 'Tim',
189      'cicero_topica.xml': 'Top',
190      'cicero_tusculanae_disputationes.xml': 'Tusc',
191      }.get(filename, 'Cic')

```

B.4 Parser e filtro de pré-nomes latinos

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  # reference: https://gist.github.com/2307114
7  # double-check: http://en.wiktionary.org/wiki/Appendix:Roman_praenomina
8
9  import codecs
10 import glob
11 import re
12
13 def parser():
14     regex = re.compile("(?:A|Ap|D|C|Cn|K|L|M|Mam|N|O|P|Q|Qu|S|Sp|Ser|Sex|Sec|Seq|Sept|T|Ti|Tit|Vel|Vo)'\?\. [A-Z]{0,}\w{0,
15     praeenomina = []
16
17     for loop in glob.glob('raw/*.txt'):
18         with file(loop, 'r') as content:
19             text = content.read()
20             for entry in regex.findall(text):
21                 praeenomina.append(entry)
22
23     return sorted(set(praeenomina))
24
25 def replacer():
26     list = parser()
27     regex = re.compile("~(.*?)\. ")
28     for loop in glob.glob('ready/*.txt'):
29         with file(loop, 'r') as content:
30             text = content.read()
31             replaced = ''
32             for entry in list:
33                 r = regex.search(entry)
34                 match = r.group(1)
35                 name = re.sub('~' + match, '(' + match + ')', entry)
36                 name = name.replace('.', ',')
37                 replaced = re.sub(entry, name, text)
38                 #with file(loop, 'w') as content:
39                     # content.write(replaced)
40                 print entry + ' -> ' + name
41
42 if __name__ == "__main__":
43     replacer()

```

B.5 Testador de stopwords

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-

```

```

3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  import glob
7
8  from nltk import Text
9  from nltk.tokenize import word_tokenize
10 from nltk.corpus import cicero
11
12 from CatXMLReader import CategorizedXMLCorpusReader
13
14 def stopwords():
15     stopwords = []
16     with file('.././stopwords/latin', 'r') as content:
17         for line in content.readlines():
18             stopwords.append(line.replace('\n', ''))
19     return stopwords
20
21 def tokenizer():
22     fileids = cicero.abspaths()
23     reader = CategorizedXMLCorpusReader('/', fileids, cat_file='categories.txt')
24     tokens = Text(reader.words(fileids))
25     return tokens
26
27 matches = []
28 tokens = tokenizer()
29
30 for s in stopwords():
31     counter = tokens.count(s)
32     matches.append(counter)
33     percentage = (float(counter)/float(len(tokens)))*100
34     print "%d\t%f\t%s" % (counter, percentage, s)
35
36 total_stat = (float(sum(matches))/float(len(tokens)))*100
37 print "stopwords: %d (%f percent)" % (sum(matches), total_stat)

```

B.6 Analisador de frequências

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  import string
7  import optparse
8  import pylab
9
10 from itertools import islice
11
12 from CatXMLReader import CategorizedXMLCorpusReader
13

```

```

14 from CatXMLReader import stopless
15 from CatXMLReader import punctless
16
17 from nltk.corpus import stopwords
18 from nltk.corpus import cicero
19
20 from nltk import FreqDist
21 from nltk import Text
22
23 parser = optparse.OptionParser("Usage: %prog [options]")
24 parser.add_option("-s", "--stopwords", action="store_true", dest="stopwords",
25                  default=False, help="include stopwords in the calculations")
26 parser.add_option("-p", "--plot", action="store_true", dest="plot",
27                  default=False, help="plot the frequency distribution of terms")
28 parser.add_option("-z", "--zipf", action="store_true", dest="zipf",
29                  default=False, help="plots a zipf's law log.log graph")
30 parser.add_option("-l", "--limit", type="int", dest="limit",
31                  default=100, help="prints calculation of first (default: 100) terms")
32 parser.add_option("-c", "--count", type="int", dest="count",
33                  default=100, help="shows only counts higher than (default: 100)")
34
35 (options, args) = parser.parse_args()
36 #if options is None:
37 #    parser.print_help()
38 #    exit(-1)
39
40 class MyFreqDist(FreqDist):
41     def plot(self, *args, **kwargs):
42         if len(args) == 0:
43             args = [len(self)]
44             samples = list(islice(self, *args))
45
46             cumulative = _get_kwarg(kwargs, 'cumulative', False)
47             if cumulative:
48                 freqs = list(self._cumulative_frequencies(samples))
49             else:
50                 freqs = [self[sample] for sample in samples]
51
52             fig = pylab.figure(figsize=(12.5, 5))
53             ax = fig.add_subplot(1, 1, 1)
54
55             if "title" in kwargs:
56                 ax.set_title(kwargs["title"])
57                 del kwargs["title"]
58             if "xlabel" in kwargs:
59                 ax.set_xlabel(kwargs["xlabel"])
60                 del kwargs["xlabel"]
61             if "ylabel" in kwargs:
62                 ax.set_ylabel(kwargs["ylabel"])
63                 del kwargs["ylabel"]
64
65             ax.plot(freqs, 'k+-', **kwargs)
66             ax.grid(True, color="silver")
67             pylab.xticks(range(len(samples)), [str(s) for s in samples], rotation=90)

```

```

68     pylab.tight_layout()
69     pylab.savefig('word_frequency.eps', dpi=300)
70     pylab.show()
71
72 def _get_kwarg(kwargs, key, default):
73     if key in kwargs:
74         arg = kwargs[key]
75         del kwargs[key]
76     else:
77         arg = default
78     return arg
79
80 categories = 'categories.txt'
81 reader = CategorizedXMLCorpusReader(cicero.root,
82                                     cicero.abspaths(),
83                                     cat_file=categories)
84
85 data = reader.words(cicero.fileids())
86
87 if options.stopwords is True:
88     filtered = punctless(data)
89 else:
90     filtered = punctless(stopless(data))
91
92 dist = MyFreqDist(Text(filtered))
93
94 if options.plot is True:
95     dist.plot(options.limit,
96               cumulative=False,
97               title=u'Gráfico de frequência (' + str(options.limit) + ' termos)',
98               ylabel=u'Ocorrências',
99               xlabel=u'Termos')
100 else:
101     print 'Data length: ' + str(len(data))
102     print 'Filtered data: ' + str(len(filtered))
103     print 'Distribution of: ' + str(len(dist))
104     print '\nCOUNT\tp(%)\tTERM'
105
106     total = len(dist.items())
107     limit = options.limit
108     if limit == 0:
109         limit = total
110     for item in dist.items()[:limit]:
111         if len(item[0]) >= 1 and item[1] >= options.count:
112             percentage = dist.freq(item[0]) * 100
113             percentage = '{0:.3}'.format(percentage)
114             print '%d\t%s\t%s' % (item[1], percentage + '%', item[0])
115
116 if options.zipf is True:
117     ranks = []
118     freqs = []
119     for rank, word in enumerate(dist):
120         ranks.append(rank+1)
121         freqs.append(dist[word])

```

```

122     fig = pylab.figure(figsize=(7.5, 5))
123     ax = fig.add_subplot(1, 1, 1)
124     ax.loglog(ranks, freqs, 'k-')
125     ax.loglog(range(1, len(freqs)+1), [len(freqs)/x for x in range(1, len(freqs)+1)], 'k--')
126     ax.grid(True, color="silver")
127     ax.set_title(u'Lei de Zipf (' + str(len(dist.items())) + ' termos)')
128     ax.set_ylabel(u'Frequência')
129     ax.set_xlabel(u'Ordem')
130     pylab.tight_layout()
131     pylab.savefig('word_frequency_zipf.eps', dpi=300)
132     pylab.show()
133

```

B.7 Gerador de concordâncias

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  import glob
7  import optparse
8  import string
9
10 from CatXMLReader import CategorizedXMLCorpusReader
11 from CatXMLReader import ciceroabbr
12
13 from nltk.corpus import cicero
14
15 from nltk import ConcordanceIndex
16 from nltk import Text
17
18 parser = optparse.OptionParser("Usage: %prog [options]")
19 parser.add_option("-l", "--lookup", type="string", dest="term",
20                  help="look up concordances for a word")
21 parser.add_option("-f", "--fake", action="store_true", dest="fake",
22                  default=False, help="considers non-ciceronian texts")
23 parser.add_option("-w", "--width", type="int", dest="width",
24                  default=150, help="width of the context data")
25 parser.add_option("-c", "--count", type="int", dest="count",
26                  default=1, help="how many matches to display")
27 parser.add_option("-v", "--verbose", action="store_true", dest="verbose",
28                  default=False, help="print headers or stats")
29
30 (options, args) = parser.parse_args()
31 if options.term is None:
32     parser.print_help()
33     exit(-1)
34
35 reset = '\033[1;m'
36 red = '\033[1;31m'

```

```

37 green = '\033[1;32m'
38 yellow = '\033[1;33m'
39 blue = '\033[1;34m'
40
41 class MyText(Text):
42     def search(self, corpus, word, width, lines):
43         res = self.concordance(word, width, lines, corpus)
44         if res is not None:
45             print res
46
47     def concordance(self, corpus, word, width=150, lines=1):
48         if '_concordance_index' not in self.__dict__:
49             if options.verbose is True:
50                 print "\nBuilding index..."
51                 self._concordance_index = MyConcordanceIndex(self.tokens, key=lambda s:s.lower())
52                 self._concordance_index.print_concordance(width, lines, corpus, word)
53
54 class MyConcordanceIndex(ConcordanceIndex):
55     def print_concordance(self, corpus, word, width=150, lines=1):
56         half_width = (width - len(word) - 2) / 2
57         context = width/4
58
59         offsets = self.offsets(word)
60         if offsets:
61             lines = min(lines, len(offsets))
62             if options.verbose is True:
63                 print "Displaying %s of %s matches:" % (lines, len(offsets))
64             for i in offsets:
65                 if lines <= 0:
66                     break
67                 left = ( ' ' * half_width +
68                        ' '.join(self._tokens[i-context:i]))
69                 right = ' '.join(self._tokens[i+1:i+context])
70                 left = left[-half_width:]
71                 right = right[:half_width]
72                 abbr = ciceroabbr(corpus)
73                 abbrinfo = '[' + abbr + ']'
74                 abbrinfo = abbrinfo.center(12, ' ').replace(abbr, green + abbr + reset)
75                 print abbrinfo + '[' + left, yellow + self._tokens[i] + reset, right + ']'
76                 lines -= 1
77         else:
78             if options.verbose is True:
79                 print "No matches found for " + word + " in " + corpus
80                 #exit(-1)
81
82 def corpora_loader(corpus, fake):
83     reader = CategorizedXMLCorpusReader(cicero.root,
84                                         cicero.abspaths(),
85                                         cat_file='categories.txt')
86     data = Text(reader.words([corpus]))
87     return data
88
89 if __name__ == "__main__":
90     for corpus in cicero.fileids():

```

```

91         if corpus in cicero.fileids(['spurious']) and options.fake is False:
92             continue
93         content = corpora_loader(corpus, fake=options.fake)
94         text = MyText(content)
95         res = text.search(options.term,
96                           options.width,
97                           options.count,
98                           corpus)
99         if res is not None:
100             print res

```

B.8 Sugestões de léxico para aprendizagem

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  from CatXMLReader import CategorizedXMLCorpusReader
7  from CatXMLReader import stopless
8  from CatXMLReader import punctless
9
10 from nltk.corpus import cicero
11
12 from nltk import FreqDist
13 from nltk import Text
14
15 # experimental
16 from latin_lemmatizer import lemmatize
17
18 # fancy dictionary
19 from collections import defaultdict
20
21 total = defaultdict(int)
22 for corpus in cicero.fileids():
23     print corpus
24     reader = CategorizedXMLCorpusReader(cicero.root,
25                                       cicero.abspaths(),
26                                       cat_file='categories.txt')
27     try:
28         dist = FreqDist(Text(punctless(stopless(reader.words([corpus])))))
29     except UnicodeEncodeError as e:
30         print str(e)
31         break
32
33     definitions = {}
34     stat = reader.words([corpus])
35     for item in dist.items()[:1000]:
36         entry = item[0]
37         if len(entry) >= 2:
38             lemma = lemmatize(item[0])

```



```

39         if lemma is not None:
40             if lemma not in total:
41                 definitions[entry] = lemma
42                 num = dist[entry]
43                 total[lemma] += num
44             #print sum(total.values()), len(stat)
45             #print corpus + ': ' + ', '.join(sorted(definitions.values()))
46
47 res = sorted(total.items(), key=lambda x: x[1], reverse=True)
48 for r in res:
49     print str(r[1]) + ':' + r[0]
50
51 print sum(total.values())

```

B.9 Protótipo de algoritmo de radicalização

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  # reference: the schinke latin stemming algorithm in python
7  # http://snowball.tartarus.org/otherapps/schinke/intro.html
8
9  import sys
10
11 que = ['atque', 'quoque', 'neque', 'itaque', 'absque', 'apsque', 'abusque',
12        'adaeque', 'adusque', 'deniquep', 'deque', 'susque', 'oblique', 'peraeque',
13        'plenisque', 'quandoque', 'quisque', 'quaequep', 'cuiusque', 'cuique',
14        'quemque', 'quamque', 'quaque', 'quique', 'quorumque', 'quarumque',
15        'quibusque', 'quosque', 'quasque', 'quotusquisque', 'quousque', 'ubique',
16        'undique', 'usque', 'uterque', 'utique', 'utroque', 'utribique', 'torque',
17        'coque', 'concoque', 'contorque', 'detorque', 'decoque', 'excoque',
18        'extorque', 'obtorque', 'optorque', 'retorque', 'recoque', 'attorque',
19        'incoque', 'intorque', 'praetorque']
20
21 noun_suffix = ['ibus', 'ius', 'ae', 'am', 'as', 'em', 'es', 'ia', 'is',
22               'nt', 'os', 'ud', 'um', 'us', 'a', 'e', 'i', 'o', 'u']
23
24 verb_suffix = ['iuntur', 'beris', 'erunt', 'untur', 'iunt', 'mini', 'ntur',
25               'stis', 'bor', 'ero', 'mur', 'mus', 'ris', 'sti', 'tis', 'tur', 'unt',
26               'bo', 'ns', 'nt', 'ri', 'm', 'r', 's', 't']
27
28 orig = []
29 nouns = []
30 verbs = []
31
32 # http://stackoverflow.com/questions/3411006/fastest-implementation-to-do-multiple-string-substitutions-in-python
33 # this is the multiple replacing algorithm proposed by matt anderson at stackoverflow in 2010
34 # it should perform faster than python's native replace method on huge corpora
35 def multi_replace(pairs, text):

```

```

36     stack = list(pairs)
37     stack.reverse()
38     def replace(stack, parts):
39         if not stack:
40             return parts
41         stack = list(stack)
42         from_, to = stack.pop()
43         # debug
44         # print 'split (%r=>%r)' % (from_, to), parts
45         split_parts = [replace(stack, part.split(from_)) for part in parts]
46         parts = [to.join(split_subparts) for split_subparts in split_parts]
47         # debug
48         # print 'join (%r=>%r)' % (from_, to), parts
49         return parts
50     return replace(stack, [text])[0]
51
52 def stemmer():
53     for entry in sys.stdin.readlines():
54         # step 2
55         entry = multi_replace([('j', 'i'), ('v', 'u')], entry.replace('\n', ''))
56
57         # hackish buffer
58         buffer = entry
59         orig.append(buffer)
60
61         # step 3
62         if entry not in que:
63             if entry.endswith('que'):
64                 entry = entry[:-3]
65             else:
66                 nouns.append(entry)
67                 verbs.append(entry)
68
69         # step 4
70         for s in noun_suffix:
71             if entry.endswith(s):
72                 entry = entry[:-len(s)]
73                 break
74
75         # step 5
76         if len(entry) >= 2:
77             nouns.append(entry)
78
79         # step 6
80         i = ['iuntur', 'erunt', 'untur', 'iunt', 'unt', 'i']
81         bi = ['beris', 'bor', 'bo', 'bi']
82         eri = ['ero', 'eri']
83
84         # repeat removal of que for verbs
85         if buffer not in que:
86             if buffer.endswith('que'):
87                 buffer = buffer[:-3]
88             else:
89                 nouns.append(buffer)

```

```

90         verbs.append(buffer)
91
92     endings = [i, bi, eri]
93     for list in endings:
94         for item in list[:-1]:
95             if buffer.endswith(item):
96                 buffer = buffer.replace(item, list[-1])
97                 break
98             else:
99                 for v in verb_suffix:
100                     if buffer.endswith(v):
101                         buffer = buffer[:-len(v)]
102                         break
103
104     # step 7
105     if len(buffer) >= 2:
106         verbs.append(buffer)
107
108     return zip(orig, nouns, verbs)
109
110 if __name__ == "__main__":
111     # step 1
112     res = stemmer()
113     if res is not None:
114         for r in res:
115             print "%s:%s:%s" % (r[0], r[1], r[2])

```

B.10 Consulta automatizada ao LemLat

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # caio begotti <caio1982@gmail.com>
4  # this is under public domain
5
6  # this is a python query script to interface with
7  # CHLT LEMLAT's web lemmatizer for latin:
8  # http://www.ilc.cnr.it/lemlat/lemlat/index.html
9
10 from sys import argv
11 from sys import exit
12
13 from lxml import etree
14
15 def lemmatize(term):
16     term = term.lower().strip()
17     parser = etree.HTMLParser()
18
19     tree = etree.parse('http://www.ilc.cnr.it/lemlat/cgi-bin/LemLat.cgi?World+Form=' + term, parser)
20     element = tree.xpath('/u/text()')
21     if element and element[0] is not None:
22         return element[0]

```

```
23
24 if __name__ == "__main__":
25     if not len(argv) == 2:
26         exit('Usage: ' + argv[0] + " 'latin word to lemmatize'")
27     else:
28         res = lemmatize(argv[1])
29         if res is not None:
30             print res
```