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 do Curso de Letras (Bacharelado em Estudos Linguísticos), do Setor de Ciências Humanas, Letras e Artes da Universidade Federal do Paraná.

Orientador: Alessandro Rolim de Moura

CURITIBA

JULHO DE 2012

Sumário

Lista de Figuras

Lista de Tabelas

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

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 similare 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 ferramentas[1], 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 (**??**).

[1] A 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 (**??**). 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 ao 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 **??**) e **??**).

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 **??**), 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 palavras[2] 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 (**??**). 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" (**??**).

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 (**??**). **??**) 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 um verdadeiro cidadão romano.

[2]Aqui 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.

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 língua[3]. 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 palavras[4]. 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 reduzido[5].

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 **??**); 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 **??**) a respeito dos famosos bancos de dados de latim do PHI[6], "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 houve tentativas de se fazer isso para o latim, porém em uma época cujos poderes da

[3] *De Oratore*, livro 3, parágrafo 42.
[4] *De Oratore*, livro 3, parágrafo 49.
[5] *De Oratore*, livro 3, a partir do parágrafo 74.
[6] O mesmo parece se aplicar aos *corpora* de latim clássico do projeto Perseus, da Tufts University.
[7] Tradução 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".

análise linguística feita pelo computador ainda não eram conhecidos (**??**), ou foram tentativas focadas em outras línguas, como o inglês (**??**) e o espanhol (**??**). Outros autores, como **??**), 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 **??**), "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 (**??**).

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 latinistas[8]. 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 latim.

Portanto, não está sendo proposta aqui uma revolução no jeito de trabalhar o

[8]Softwares 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 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 **??**), 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 palavreado 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 (**??**).

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 **??**)[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 **??**), 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 na aquisição de línguas; no caso deste trabalho, o latim.

[9]Tradução 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".

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 algorítimos 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 (**??**), 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 (**??**). 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 contríbuido sobremaneira nos últimos anos para a linguística como um todo (**??**).

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

[10]É 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 fala[11], etiquetamento de anunciados[12], radicalização de palavras[13], gramática categorial, prosódia semântica de textos[14], análises estatísticas, entre outros. Pelo fato de a linguística computacional e PLN serem áreas que atualmete 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 (**??**).

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 automatização de um computador a serviço do linguista (**??**). 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 (**??**), quando o uso da LC e análises estatísticas ainda eram tão primitivos que não eram inteiramente compreendidos,

[11] Speech-to-text e text-to-speech.
[12] Tagging.
[13] Word stemming.
[14] Sentiment analysis.

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 digitalmente[15]. 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 (**??**). 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 traduções[16]. É por essa razão que muitos projetos de LC ainda engatinham e continuam a coletar dados para melhorar suas

[15] Os 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.

[16] Uma 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.

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 (**??**).

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 (**??**). 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 decadas, 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 verdade[17]. 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.

[17]Por 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 irrestrito[18]. 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 NLTK[19].

[18] É 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.

[19] Fez 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 *scrapper*, 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 **??**), 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 simples[20]. Sua principalmente vantagem sobre arquivos de texto puro é a agregação de metadados no meio do texto, mas sem poluí-lo. A seguir, um esqueleto

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

de arquivo de corpus exemplificando seu formato:

```
<?xml version="1.0" encoding="utf-8"?>
<document>
  <meta name="author" value="marcus tullius cicero"/>
  <meta name="title" value="X"/>
  <meta name="source" value="http://latin.packhum.org/dx/text/474/Y/0"/>
  <page id="Z">
    <paragraph>
        [...]
    </paragraph>
  </page>
</document>
```

A primeira linha de todos os corpora indica o formato XML já mencionado. 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 manuscritos[21]. Por exemplo, em *Topica* não se encontrará κρινόμενον 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:

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

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

Os apêndices do trabalho se encarregam de demonstrar com detalhes alguma 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:

```
>>> from CatXMLReader import CategorizedXMLCorpusReader
>>> from nltk.corpus import cicero
>>>
>>> reader = CategorizedXMLCorpusReader(cicero.root, cicero.abspaths(), cat_file='categories.txt')
>>> list = ['cicero_in_verrem.xml']
>>> text = reader.words(list)
>>> print text[:10]
['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 — **??**), **??**) e **??**). 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 **??**), **??**), **??**) e também **??**).

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, **??**) 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*.

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

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

[24]Também aparece como *Cum Senatui Gratias Egit*.

[25]Também 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-

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

[27]Autoria 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.

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

[29]Também 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).

[30]Do 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 **??**).

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

tavianum[32], *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 agrupamentos[33] 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 algorítimos 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 (**??**) mas também dificultar a análise computacional (**??**). 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

[32]Embora 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.

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

foram elaboradas pelo autor a partir das gramáticas latinas de **??**) e **??**), e considerando resultados prévios deste trabalho comparando os filtros com os de outros autores[34].

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

```
>>> from nltk.corpus import cicero
>>> from nltk.corpus import stopwords
>>> stop = stopwords.words('latin')
>>> words = ['et', 'sum', 'loquor', 'cum', 'res']
>>> filtered = [x for x in words if x not in stop]
>>> print filtered
>>> ['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, quamuis, quanquam, quantumuis, 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.

[34] O 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, cuiusuis, 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, quaenam, quaepiam, quaeque, quaeuis, 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 tempo[35], tornando-o ideal para uso uma vez que se provou estar em constante evolução acompanhando novas descobertas no campo de processamento de linguagem natural (**??**).

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 (**??**). É programado de forma simples e elegante. Chama-se o NLTK de uma plataforma Pythônica (**??**), 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.

Para chegar ao ponto de usar uma análise de frequência, é preciso antes passar

[35]Na 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.

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 (**??**).

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 (**??**). 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" (**??**). 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. 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.

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

5 Análises

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 cerca de 500 *stopwords*, catalogadas nas seções anteriores, representam aproximadamente 33% de todas as ocorrências de palavras dos textos. Parece um número bastante alto, mas compatível com análises de outras línguas.

É comum que termos funcionais (preposições, conjunções etc) sejam maioria no uso real da língua. Todavia, ainda que isso possa indicar que o verdadeiro léxico a ser aprendido por estudantes de latim clássico seja os outros 67%, pode-se entender os números por outro ângulo: por serem tão comuns, as *stopwords* obrigatoriamente precisam ser dominadas pelos estudantes de latim clássico antes de tentarem aumentar seu vocabulário, pois já garantiriam acesso a boa parte dos textos existentes. TODO: e adverbios e afins.

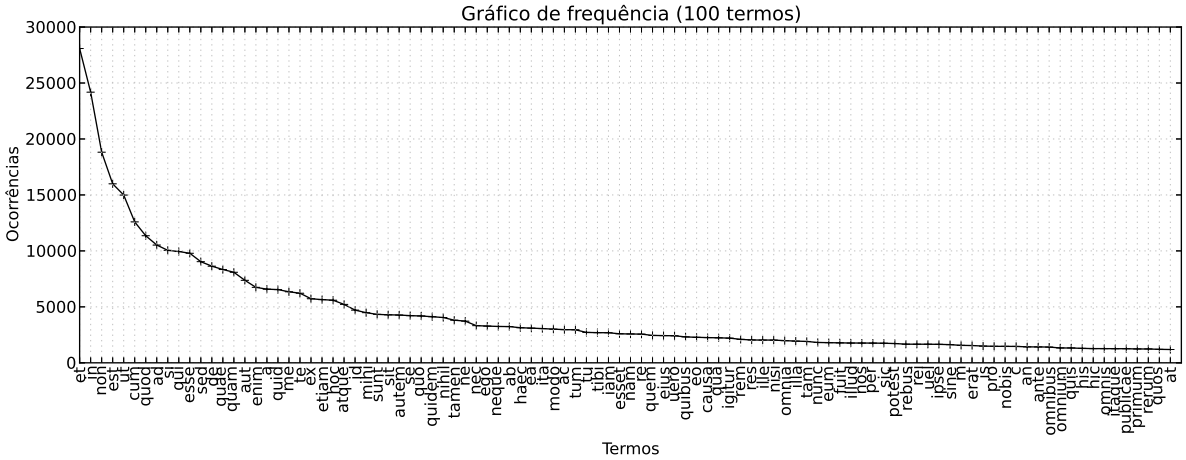

Figura 1: Gráfico de frequência (100 termos mais usados, incluindo stopwords)

Explicar FreqDist

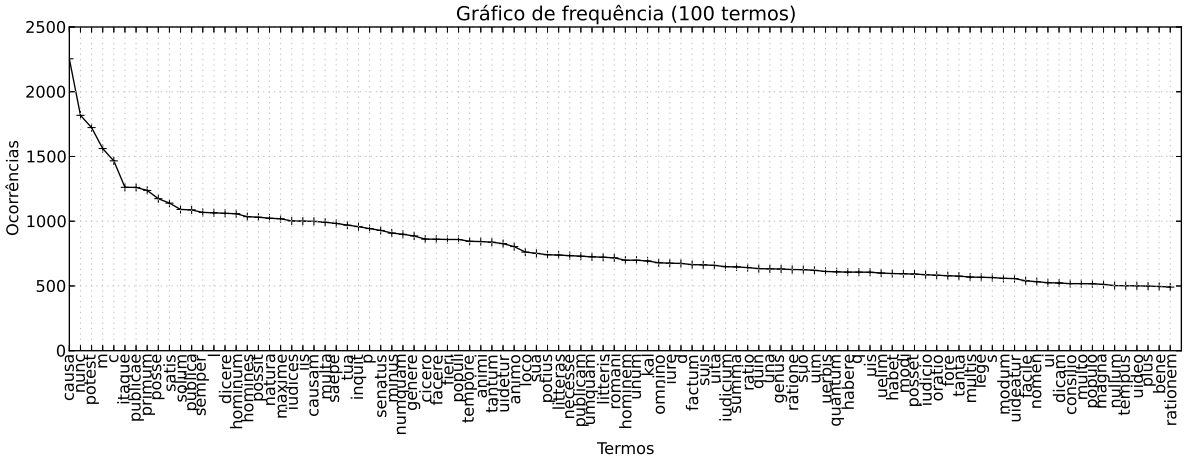

Figura 2: Gráfico de frequência (100 termos mais usados)

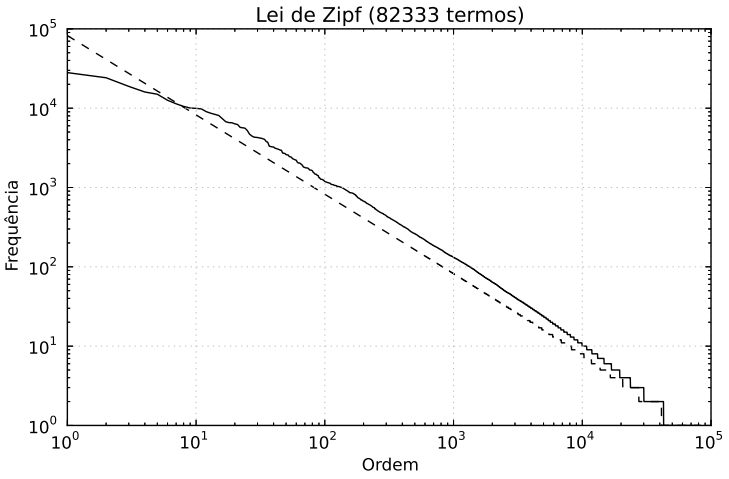

Figura 3: Gráfico atestando Lei de Zipf aos corpora

Daria pra usar FreqDist e verbos "modais"ou "reflexivos"pra analisar a estilistica dos textos?

Graficos dist.plot() com e sem stopwords

Lista de 500 palavras mais comuns, apos remover declinacoes e casos

Concordancias como uso pras frequencias em sala de aula

TODO: explicacao dos codigos escritos com python e nltk

Falar que listas a seguir pra sala de aulas sao bem diversas, comentar corpus de cada uma

de oratore

in verrem

epistulae ad familiares

epistulae ad atticum

tusculanae disputationes

TODO 42% de termos unicos em cicero, nos 5 maiores textos acima

Tabela 1: Sugestão de léxico para aprendizado de Cícero

Termo	Tradução
1	2
5	6
9	0

Termo	Tradução
1	2
5	6
9	0

Termo	Tradução
1	2
5	6
9	0

TODO: corpora nao sao perfeitos, pode haver sujeira nao esperada ainda

Propostas de uso em ambiente acadêmico e para aprendizagem do latim.

Gerar puzzles com NLTK pra sala de aula?

Lembrar que sao resultados iniciais, precisa melhorar os corpora, pensar em problemas que a LC pode ajudar etc

Embora seja uma lingua altamente flexional, o FreqDist bota o latim classico de cicero no mesmo patamar estatistico de linguas modernas, no qual stopwords dominam totalmente as listas de termos mais frequentes da lingua. A ideia de que seria possivel se expressar em latim com um numero bastante reduzido de conjuncoes e preposicoes, por exemplo, nao somente eh uma ilusao comum que se tem, mas estatisticamente esta

bastante longe da verdade.

Mencionar que os resultados serao mais uteis se forem expandidos os corpora

Dizer algo como: Although there are many advantages to having all these data stored electronically, the real power of this format comes not from the mere storage of data, but from the "relational"aspect, viz. the database program's ability to perform various operations on the material, creating new sets of data at lightning speed without disturbingtheoriginalset. Ittakesonlyafewsimplecommandsto create, for example, frequency lists for the whole database or any specifiable subset thereof.

6 Propostas de continuidade

6.1 Radicalização

Chama-se de radicalização[37] o processo de se obter, através de algoritmos, a raíz de uma palavra sem nenhum de seus afixos relevantes. É um processo conceitualmente diferente e mais simples do que encontrar o lema de uma palavra, entretanto bastante útil em LC pois permite análises mais profundas. Por exemplo, uma análise estatística de ocorrências de palavras poderia considerar não as palavras propriamente, mas seus *stems*, fazendo a análise sobre termos do mesmo grupo semântico e não gramatical.

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. **??**) propôs regras para radicalização em latim, mas não as implementou. Coube a Martin Porter implementar o algoritmo utilizando a plataforma Snowball[38].

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 LemLat, um serviço online para lematização de palavras em latim do Istituto di Linguistica

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

[38] A 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 **??**), que é excepcionalmente raro online.

Computazionale, do CNR italiano. Os dois códigos apresentados nesses apêndices podem ser combinados para automatizar e melhorar o processo de radicalização de palavras do latim no futuro.

6.2 Performance dos métodos

Limpar stopwords e pontuação dos corpora todos de uma vez é bem demorado:

6.3 Formato dos corpora

Leitor de corpus NLTK pro PHI5?

7 Conclusão

Pro fim de maio...

Referências

APÊNDICE A – Códigos

A.1 Scrapper de dados e montador dos corpora

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# caio begotti <caio1982@gmail.com>
# this is under public domain

# to add some sleep time between fetches
# and not hammer down the server
import time

# to avoid charsetting mess with UTF-8 strings
import codecs
import string

# to check if a text was already fetched
from os.path import exists

# document parser
from lxml import etree

# output generation
from elementtree.SimpleXMLWriter import XMLWriter

refs = []
urls = []

# the root directory of all latin texts in PHI
base = 'http://latin.packhum.org/'

# marcus tullius cicero entry in PHI
browse = base + 'author/474'

try:
    page = etree.parse(browse, etree.HTMLParser(encoding='utf-8'))
except Exception, err:
    print 'Browse Error: ' + str(err)

# gets the list of texts by cicero currently in PHI
matches = page.xpath("//span[@class='wnam']//text()")

```

```
counter = 1
for entry in matches:
    # creates a reference list with download addresses for every text
    refs.append((base + 'dx/text/474/%s/' % str(counter), entry.lower()))
    counter += 1

for param in refs:
    source = param[0]
    title = param[1]

    # debug
    print '[%s]' % title

    filename = title.replace(' ', '_')
    for p in string.punctuation.replace('_',''):
        filename = filename.replace(p, '')

    w = XMLWriter('cicero_' + filename + '.xml', encoding='utf-8')
    xml = w.start("document")

    # metadata entries of the output files
    w.element("meta", name="author", value="marcus tullius cicero")
    w.element("meta", name="title", value=title)
    w.element("meta", name="source", value=source + '0')

    # upon checking it no text in PHI attributed to cicero
    # has more than 500 pages, so this is a safe download limit
    for x in range(0, 500):
        lines = []
        entry = []
        section = source + str(x)
        reference = base + 'loc/474/' + str(x) + '/0'

        # debug
        print '\t<%s>' % section

        # output filename
        path = 'ready/' + filename + '-' + str(x) + '.txt'

        if not exists(path):
            # fetches the current page
            try:
                page = etree.parse(section, etree.HTMLParser(encoding='utf-8'))
            except Exception, err:
                print 'Text Error: ' + str(err)

            # parses the page paragraphs
            try:
                entry = page.xpath("//tr/td[1]//text() | //h3//text()")
            except Exception, err:
                print 'Match Error: ' + str(err)
                # a priori this is not needed but it is helpful for debugging
                f = codecs.open("log.txt", "a", "utf8")
                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)
```

A.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
```

```
        # fixes the linebreaking of corpora
        text = text.replace("- ", "")

        # fixes unused letters for their real latin ones
        text = text.replace("v","u")
        text = text.replace("j","i")

        with file(loop.replace('raw/', 'ready/'), 'w') as content:
            content.write(text)

if __name__ == "__main__":
    normalizer()
```

A.3 Classe de manipulação dos corpora

```
# http://stackoverflow.com/questions/6849600/does-anyone-have-a-categorized-xml-corpus-reader-for-nltk

# standard nltk classes
from nltk.corpus.reader import CategorizedCorpusReader
from nltk.corpus.reader import XMLCorpusReader

# stopwords (i.e. latin ones)
from nltk.corpus import stopwords

# for CategorizedCorpusReader's init
from nltk.compat import defaultdict

# punctuations load
import string

class MyCategorizedCorpusReader(CategorizedCorpusReader):
    def _init(self):
        self._f2c = defaultdict(set)
        self._c2f = defaultdict(set)

        if self._pattern is not None:
            for file_id in self._fileids:
                category = re.match(self._pattern, file_id).group(1)
                self._add(file_id, category)

        elif self._map is not None:
            for (file_id, categories) in self._map.items():
                for category in categories:
                    self._add(file_id, category)

        elif self._file is not None:
            for line in self.open(self._file).readlines():
                line = line.strip()
                file_id, categories = line.split(self._delimiter, 1)
                # https://github.com/nltk/nltk/issues/250
                #if file_id not in self.fileids():
```

```
                #    raise ValueError('In category mapping file %s: %s '
                #                     'not found' % (self._file, file_id))
                for category in categories.split(self._delimiter):
                    self._add(file_id, category)


class CategorizedXMLCorpusReader(MyCategorizedCorpusReader, XMLCorpusReader):
    def __init__(self, *args, **kwargs):
        MyCategorizedCorpusReader.__init__(self, kwargs)
        XMLCorpusReader.__init__(self, *args, **kwargs)

    def _resolve(self, fileids, categories):
        if fileids is not None and categories is not None:
            raise ValueError('Specify fileids or categories, not both')
        if categories is not None:
            return self.fileids(categories)
        else:
            return fileids

    def raw(self, fileids=None, categories=None):
        return XMLCorpusReader.raw(self, self._resolve(fileids, categories))

    def words(self, fileids=None, categories=None):
        words = []
        fileids = self._resolve(fileids, categories)
        for fileid in fileids:
            words += XMLCorpusReader.words(self, fileid)
        return words

    def text(self, fileids=None, categories=None):
        fileids = self._resolve(fileids, categories)
        text = ""
        for fileid in fileids:
            for i in self.xml(fileid).getiterator():
                if i.text:
                    text += i.text
        return text

    def sents(self, fileids=None, categories=None):
        text = self.words(fileids, categories)
        sents = nltk.PunktSentenceTokenizer().tokenize(text)
        return sents

    def paras(self, fileids=None, categories=None):
        return CategorizedCorpusReader.paras(self, self._resolve(fileids, categories))


def stopless(wordslist):
    stop = stopwords.words('latin')
    filtered = [x for x in wordslist if x not in stop]
    return filtered

def punctless(wordslist):
    punct = string.punctuation
```

```
91
92      punct += u'\u00a7' # SECTION SIGN
93      punct += u'\u00b3' # SUPERSCRIPT THREE
94      punct += u'\u00b2' # SUPERSCRIPT TWO
95      punct += u'\u00b7' # MIDDLE DOT
96      punct += u'\u00b9' # SUPERSCRIPT ONE
97      punct += u'\u2014' # EM DASH
98      punct += u'\u2019' # RIGHT SINGLE QUOTATION MARK
99      punct += u'\u2020' # DAGGER
100     punct += u'\u2184' # LATIN SMALL LETTER REVERSED C
101     punct += u'\u221e' # INFINITY
102     punct += u'\u23d1' # METRICAL BREVE
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',
```

```
        'cicero_facete_dicta.xml': 'Facet',
        'cicero_hortensius.xml': 'Hort',
        'cicero_in_catilinam.xml': 'Catil',
        'cicero_in_pisonem.xml': 'Pis',
        'cicero_in_q_caecilium.xml': 'DivCaec',
        'cicero_in_sallustium_sp.xml': 'Sal',
        'cicero_in_vatinium.xml': 'Vat',
        'cicero_in_verrem.xml': 'Ver',
        'cicero_incertorum_librorum_fragmenta.xml': 'LibFrr',
        'cicero_laelius_de_amicitia.xml': 'Amic',
        'cicero_lucullus.xml': 'Luc',
        'cicero_orationum_deperditarum_frr.xml': 'DepFrr',
        'cicero_orationum_incertarum_frr.xml': 'IncFrr',
        'cicero_orator.xml': 'Orat',
        'cicero_paradoxa_stoicorum.xml': 'Parad',
        'cicero_philippicae.xml': 'Phil',
        'cicero_philosophicorum_librorum_frr.xml': 'PhilFrr',
        'cicero_post_reditum_ad_populum.xml': 'RedPop',
        'cicero_post_reditum_in_senatu.xml': 'RedSen',
        'cicero_pro_archia.xml': 'Arch',
        'cicero_pro_balbo.xml': 'Balb',
        'cicero_pro_caecina.xml': 'Caec',
        'cicero_pro_caelio.xml': 'Cael',
        'cicero_pro_cluentio.xml': 'Clu',
        'cicero_pro_flacco.xml': 'Flac',
        'cicero_pro_fonteio.xml': 'Font',
        'cicero_pro_lege_manilia.xml': 'Man',
        'cicero_pro_ligario.xml': 'Lig',
        'cicero_pro_marcello.xml': 'Marc',
        'cicero_pro_milone.xml': 'Mil',
        'cicero_pro_murena.xml': 'Mur',
        'cicero_pro_plancio.xml': 'Planc',
        'cicero_pro_q_roscio_comoedo.xml': 'QRosc',
        'cicero_pro_quinctio.xml': 'Quinct',
        'cicero_pro_rabirio_perduellionis_reo.xml': 'RabPerd',
        'cicero_pro_rabirio_postumo.xml': 'RabPost',
        'cicero_pro_rege_deiotaro.xml': 'Deiot',
        'cicero_pro_s_roscio_amerino.xml': 'SRosc',
        'cicero_pro_scauro.xml': 'Scaur',
        'cicero_pro_sestio.xml': 'Sest',
        'cicero_pro_sulla.xml': 'Sul',
        'cicero_pro_tullio.xml': 'Tul',
        'cicero_rhetorica_ad_herennium_sp.xml': 'RhetHer',
        'cicero_timaeus.xml': 'Tim',
        'cicero_topica.xml': 'Top',
        'cicero_tusculanae_disputationes.xml': 'Tusc',
    }.get(filename, 'Cic')
```

A.4 Parser e filtro de pré-nomes latinos

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# caio begotti <caio1982@gmail.com>
# this is under public domain

# reference: https://gist.github.com/2307114
# double-check: http://en.wiktionary.org/wiki/Appendix:Roman_praenomina

import codecs
import glob
import re

def parser():
    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,
    praenomina = []

    for loop in glob.glob('raw/*.txt'):
        with file(loop, 'r') as content:
            text = content.read()
            for entry in regex.findall(text):
                praenomina.append(entry)

    return sorted(set(praenomina))

def replacer():
    list = parser()
    regex = re.compile("^(.*)\. ")
    for loop in glob.glob('ready/*.txt'):
        with file(loop, 'r') as content:
            text = content.read()
            replaced = ''
            for entry in list:
                r = regex.search(entry)
                match = r.group(1)
                name = re.sub('^' + match, '(' + match + ')', entry)
                name = name.replace('.', '')
                replaced = re.sub(entry, name, text)
                #with file(loop, 'w') as content:
                #    content.write(replaced)
                print entry + ' -> ' + name

if __name__ == "__main__":
    replacer()
```

A.5 Testador de stopwords

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

```
# caio begotti <caio1982@gmail.com>
# this is under public domain

import glob

from nltk import Text
from nltk.tokenize import word_tokenize
from nltk.corpus import cicero

from CatXMLReader import CategorizedXMLCorpusReader

def stopwords():
    stopwords = []
    with file('../../stopwords/latin', 'r') as content:
        for line in content.readlines():
            stopwords.append(line.replace('\n',''))
    return stopwords

def tokenizer():
    fileids = cicero.abspaths()
    reader = CategorizedXMLCorpusReader('/', fileids, cat_file='categories.txt')
    tokens = Text(reader.words(fileids))
    return tokens

matches = []
tokens = tokenizer()

for s in stopwords():
    counter = tokens.count(s)
    matches.append(counter)
    percentage = (float(counter)/float(len(tokens)))*100
    print "%d\t%f\t%s" % (counter, percentage, s)

total_stat = (float(sum(matches))/float(len(tokens)))*100
print "stopwords: %d (%f percent)" % (sum(matches), total_stat)
```

A.6 Analisador de frequências

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# caio begotti <caio1982@gmail.com>
# this is under public domain

import string
import optparse
import pylab

from itertools import islice

from CatXMLReader import CategorizedXMLCorpusReader

```

```
from CatXMLReader import stopless
from CatXMLReader import punctless

from nltk.corpus import stopwords
from nltk.corpus import cicero

from nltk import FreqDist
from nltk import Text

parser = optparse.OptionParser("Usage: %prog [options]")
parser.add_option("-s", "--stopwords", action="store_true", dest="stopwords",
                  default=False, help="include stopwords in the calculations")
parser.add_option("-p", "--plot", action="store_true", dest="plot",
                  default=False, help="plot the frequency distribution of terms")
parser.add_option("-z", "--zipf", action="store_true", dest="zipf",
                  default=False, help="plots a zipf's law log.log graph")
parser.add_option("-l", "--limit", type="int", dest="limit",
                  default=100, help="prints calculation of first (default: 100) terms")
parser.add_option("-c", "--count", type="int", dest="count",
                  default=100, help="shows only counts higher than (default: 100)")

(options, args) = parser.parse_args()
#if options is None:
#    parser.print_help()
#    exit(-1)

class MyFreqDist(FreqDist):
    def plot(self, *args, **kwargs):
        if len(args) == 0:
            args = [len(self)]
        samples = list(islice(self, *args))

        cumulative = _get_kwarg(kwargs, 'cumulative', False)
        if cumulative:
            freqs = list(self._cumulative_frequencies(samples))
        else:
            freqs = [self[sample] for sample in samples]

        fig = pylab.figure(figsize=(12.5, 5))
        ax = fig.add_subplot(1, 1, 1)

        if "title" in kwargs:
            ax.set_title(kwargs["title"])
            del kwargs["title"]
        if "xlabel" in kwargs:
            ax.set_xlabel(kwargs["xlabel"])
            del kwargs["xlabel"]
        if "ylabel" in kwargs:
            ax.set_ylabel(kwargs["ylabel"])
            del kwargs["ylabel"]

        ax.plot(freqs, 'k+-', **kwargs)
        ax.grid(True, color="silver")
        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'GrÃ¡fico de frequÃªncia (' + str(options.limit) + ' termos)',
98                ylabel=u'OcorrÃªncias',
99                xlabel=u'Termos')
100 else:
101     print 'Data lenght: ' + 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
123     fig = pylab.figure(figsize=(7.5, 5))
124     ax = fig.add_subplot(1, 1, 1)
125     ax.loglog(ranks, freqs, 'k-')
126     ax.loglog(range(1, len(freqs)+1), [len(freqs)/x for x in range(1, len(freqs)+1)], 'k--')
127     ax.grid(True, color="silver")
128     ax.set_title(u'Lei de Zipf (' + str(len(dist.items())) + ' termos)')
129     ax.set_ylabel(u'FrequÃªncia')
130     ax.set_xlabel(u'Ordem')
131     pylab.tight_layout()
132     pylab.savefig('word_frequency_zipf.eps', dpi=300)
133     pylab.show()
```

A.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'
```

```
green = '\033[1;32m'
yellow = '\033[1;33m'
blue = '\033[1;34m'

class MyText(Text):
    def search(self, corpus, word, width, lines):
        res = self.concordance(word, width, lines, corpus)
        if res is not None:
            print res

    def concordance(self, corpus, word, width=150, lines=1):
        if '_concordance_index' not in self.__dict__:
            if options.verbose is True:
                print "\nBuilding index..."
            self._concordance_index = MyConcordanceIndex(self.tokens, key=lambda s:s.lower())
        self._concordance_index.print_concordance(width, lines, corpus, word)

class MyConcordanceIndex(ConcordanceIndex):
    def print_concordance(self, corpus, word, width=150, lines=1):
        half_width = (width - len(word) - 2) / 2
        context = width/4

        offsets = self.offsets(word)
        if offsets:
            lines = min(lines, len(offsets))
            if options.verbose is True:
                print "Displaying %s of %s matches:" % (lines, len(offsets))
            for i in offsets:
                if lines <= 0:
                    break
                left = (' ' * half_width +
                        ' '.join(self._tokens[i-context:i]))
                right = ' '.join(self._tokens[i+1:i+context])
                left = left[-half_width:]
                right = right[:half_width]
                abbr = ciceroabbr(corpus)
                abbrinfo = '[' + abbr + ']'
                abbrinfo = abbrinfo.center(12, ' ').replace(abbr, green + abbr + reset)
                print abbrinfo + '[' + left, yellow + self._tokens[i] + reset, right + ']'
                lines -= 1
        else:
            if options.verbose is True:
                print "No matches found for " + word + " in " + corpus
            #exit(-1)

def corpora_loader(corpus, fake):
    reader = CategorizedXMLCorpusReader(cicero.root,
                                        cicero.abspaths(),
                                        cat_file='categories.txt')
    data = Text(reader.words([corpus]))
    return data

if __name__ == "__main__":
    for corpus in cicero.fileids():
```

```
        if corpus in cicero.fileids(['spurious']) and options.fake is False:
            continue
        content = corpora_loader(corpus, fake=options.fake)
        text = MyText(content)
        res = text.search(options.term,
                          options.width,
                          options.count,
                          corpus)
        if res is not None:
            print res
```

A.8 Sugestões de léxico para aprendizagem

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# caio begotti <caio1982@gmail.com>
# this is under public domain

from CatXMLReader import CategorizedXMLCorpusReader
from CatXMLReader import stopless
from CatXMLReader import punctless

from nltk.corpus import cicero

from nltk import FreqDist
from nltk import Text

# experimental
from latin_lemmatizer import lemmatize

# fancy dictionary
from collections import defaultdict

list = [
#'cicero_de_republica.xml',
#'cicero_brutus.xml',
#'cicero_de_divinatione.xml',
#'cicero_rhetorica_ad_herennium_sp.xml',
#'cicero_de_inventione.xml',
#'cicero_de_natura_deorum.xml',
#'cicero_de_officiis.xml',
#'cicero_de_finibus.xml',
#'cicero_philippicae.xml',
'cicero_de_oratore.xml',
'cicero_in_verrem.xml',
'cicero_epistulae_ad_familiares.xml',
'cicero_epistulae_ad_atticum.xml',
'cicero_tusculanae_disputationes.xml'
]

total = defaultdict(int)
```

```
for corpus in list:
    reader = CategorizedXMLCorpusReader(cicero.root,
                                        cicero.abspaths(),
                                        cat_file='categories.txt')
    try:
        dist = FreqDist(Text(punctless(stopless(reader.words([corpus])))))
    except UnicodeEncodeError as e:
        print str(e)
        break

    definitions = {}
    stat = reader.words([corpus])
    for item in dist.items()[:100]:
        entry = item[0]
        if len(entry) >= 2:
            lemma = lemmatize(item[0])
            if lemma is not None:
                if lemma not in total:
                    definitions[entry] = lemma
                num = dist[entry]
                total[lemma] += num
    #print sum(total.values()), len(stat)
    #print corpus + ': ' + ', '.join(sorted(definitions.values()))

res = sorted(total.items(), key=lambda x: x[1], reverse=True)
for r in res:
    print str(r[1]) + ':' +  r[0]

print sum(total.values())
```

A.9 Protótipo de algoritmo de radicalização

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# caio begotti <caio1982@gmail.com>
# this is under public domain

# reference: the schinke latin stemming algorithm in python
# http://snowball.tartarus.org/otherapps/schinke/intro.html

import sys

que = ['atque', 'quoque', 'neque', 'itaque', 'absque', 'apsque', 'abusque',
'adaeque', 'adusque', 'deniquep', 'deque', 'susque', 'oblique', 'peraeque',
'plenisque', 'quandoque', 'quisque', 'quaequep', 'cuiusque', 'cuique',
'quemque', 'quamque', 'quaque', 'quique', 'quorumque', 'quarumque',
'quibusque', 'quosque', 'quasque', 'quotusquisque', 'quousque', 'ubique',
'undique', 'usque', 'uterque', 'utique', 'utroque', 'utribique', 'torque',
'coque', 'concoque', 'contorque', 'detorque', 'decoque', 'excoque',
'extorque', 'obtorque', 'optorque', 'retorque', 'recoque', 'attorque',
'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])
```

A.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
```

```
# CHLT LEMLAT's web lemmatizer for latin:
# http://www.ilc.cnr.it/lemlat/lemlat/index.html

from sys import argv
from sys import exit

from lxml import etree

def lemmatize(term):
    term = term.lower().strip()
    parser = etree.HTMLParser()

    tree = etree.parse('http://www.ilc.cnr.it/lemlat/cgi-bin/LemLat_cgi.cgi?World+Form=' + term, parser)
    element = tree.xpath('//u//text()')
    if element and element[0] is not None:
        return element[0]

if __name__ == "__main__":
    if not len(argv) == 2:
        exit('Usage: ' + argv[0] + " 'latin word to lemmatize'")
    else:
        res = lemmatize(argv[1])
        if res is not None:
            print res
```