Universidade do Minho
Escola de Engenharia

Manuel da Silva Alves

Modelos de Língua - Lingua::srilm

Outubro, 2010

Universidade do Minho
Escola de Engenharia

Manuel da Silva Alves

Modelos de Língua - Lingua::srilm

Dissertação de Mestrado
Mestrado em Informática

Trabalho efectuado sob a orientação do
Professor Doutor José João Almeida

Outubro, 2010

Agradecimentos

Em primeiro lugar, quero expressar a minha gratidão aos meus orientadores José João Almeida e Alberto Simões. Sem a sua sabedoria, paciência e conselhos, este trabalho não seria possível.

Também quero agradecer ao director da Escola Básica e Secundária Arga e Lima, Agostinho Gomes, por me ter permitido algum tempo para eu dedicar a este trabalho.

Finalmente, quero agradecer à minha mãe Rosa, ao meu primo José, à minha namorada Celine e ao meu amigo Abel por todo o apoio. Escrever uma tese e realizar um projecto desta magnitude enquanto se tem um trabalho a tempo inteiro requere uma grande vontade e coragem e o seu desenvolvimento passou por dificuldades devido á minha situação profissional.

A todos acima, Eu agradeço do fundo do meu coração.

Se algo sabes na vida, não te precipites a ensinar como quem tiraniza, menosprezando conquistas alheias. Examina as situações características de cada um e procura, primeiramente, entender o irmão de luta. Saber não é tudo. É necessário fazer. E para bem fazer, homem algum dispensará a calma e a serenidade, imprescindíveis ao êxito, nem desdenhará a cooperação, que é a companheira dileta do amor.

Emmanuel , (Livro "Vinha de Luz", pelo médium Francisco Xavier)

Resumo

Título da tese: Modelos de Língua - Lingua::srilm.

Um Modelo de Língua pode ser visto como uma ferramenta que recebe com entrada uma texto T e retorna um valor percental de 0 a 1, sendo esse valor a probabilidade de esse mesmo texto pertencer a essa língua. Actualmente os *Modelos de Língua* mais usados são baseados em estatística. Os Modelos de Língua podem ser usados em várias aplicações como:

- Reconhecimento de Fala.
- Etiquetação Estatística e Segmentação.
- Tradução Automática.

Referir também que tem-se verificado uma elevada utilização desta ferramenta na criacão de Modelos de Tradução.

O facto de o Perl ter surgido como linguagem de scripting modular e reutilizável,e claro, nesse aspecto uma das mais populares linguagens de programação, faz com que seja uma escolha natural para codificar algoritmos de necessários. O objectivo desta tese para além dum profundo estudo do *state-of-art*, ou seja, daquilo que já foi construido e do que está actualmente a ser implementado relativamente a Modelos de Língua é usar o Perl como linguagem de programação para criar o módulo Lingua::srilm em conjunto com outras ferramentas como por exemplo o SRI Toolkit (criação de modelos de língua) e o SWIG (gerador de interfaces). Mais concretamente, interessam as seguintes perspectivas:

- O Lingua::srilm é um módulo Perl que pretende servir de base ao estudo de diferentes abordagens no uso de Modelos de Língua.
- Não pretende ser uma ferramenta eficiente, mas uma ferramenta que possa servir de uso em aplicações futuras.
- Neste sentido pretende ser uma ferramenta que possa ser usada de forma modular.

Neste trabalho pretende-se o desenvolvimento de um módulo, Lingua::srilm, que permita questionar os Modelos de Língua que foram sendo criados.

Áreas de aplicação: Processamento de Linguagem Natural.

Keywords: Perl, C, C++, SRILM, SWIG.

Abstract

A Language Model could be looked as a tool that receives as a input a text T and returns a percentual value of 0 to 1, then this value is used as a probability of text T being a member of that language. In the present the most used *Language Models* are statistic-based. Language Models could be used in several aplications such as:

- Speech Recognition.
- Statistic Labeling and Segmentation.
- Automatic Translation.

This tool it as been massively used in the construction of Translation Models.

The fact of the Perl have appeared as modular and reusable language of scripting, and clearly, in this aspect one of the most popular programming languages, makes it a natural choice to encode the necessary algorithms . The main goal of this thesis beyond a deep study of *state-of-art* and of what currently it is made and being implemented related to Language Models is to use the Perl as programming language to create the module Lingua::srilm using also other auxiliary tools as for example the SRI Toolkit (language models creation) and SWIG (interface generator).

More concretely, we are interested in the following perspectives:

- Lingua::srilm is a Perl module that it intends to serve of base to the study of different approaches in the use of Language Models.
- It does not intend to be very eficient, but a tool that can be used in future aplications.
- In this way it intends to be a tool that can be used in a modular way.

In this work is intended to do the development of the module Lingua::srilm, that allow querying the Language Models that have been created.

Aplication Areas: Natural Language Processing.

Keywords: Perl, C, C++, SRILM, SWIG.

Lista de Siglas e Acrónimos

HMM Hidden Markov Model

SRILM Stanford Research Institute Language Modeling (toolkit)

API Application Programming Interface

SLM Statistical Language Modeling

PERL Pratical Extraction Report Language

CPAN Comprehensive Perl Archive Network

SOAP Simple Object Access Protocol

RPC Remote Procedure Call

HTTP Hyper Text transfer protocol

BASH Born Again SHell

PLN Processamento Linguagem Natural

MLE Maximum Likelihood Estimation

OOV Out Of Vocabulary

SWIG Simplified Wrapper Interface Generator

XML eXtended Markup Language

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Motivações

O projecto de Mestrado em que estamos inseridos pertence à área do Processamento de Linguagem Natural. As ferramentas actuais de criação de Modelos de Língua não disponibilizam um interface claro e limpo que permita a utilização dos mesmos. Tornou-se necessário criar um interface que permitisse o uso das funcionalidades dessas mesmas ferramentas, até porque muitas delas estão desenvolvidas em linguagens (C/C++) diferentes da linguagem Perl que queremos usar. Com a análise das várias ferramentas existentes, tornou-se clara a escolha do SRILM Toolkit para a criação de Modelos de Língua. Para a criação do interface que permitisse usar esses mesmos modelos, foi escolhida a ferramenta SWIG. Foi escolhida a plataforma XML-RPC para criação de uma arquitectura Cliente-Servidor que permitisse a execução das *queries* com uma performance mais elevada. O objectivo é construir um módulo que permita questionar os *Modelos de Língua* usando a linguagem Perl, referido ao longo deste documento como o módulo Lingua::srilm. O Lingua::srilm é um módulo Perl que pretende servir de base ao estudo de diferentes abordagens no uso de *Modelos de Língua*. Não pretende ser uma ferramenta de extrema eficiência, mas uma ferramenta que possa ser usada em aplicações existentes ou que vierem a ser desenvolvidas posteriormente.

1.2 Objectivos e Contribuições

Nesta tese, pretende-se a criação de um prótipo que permita a utilização de *Modelos de Língua*[1] usando a linguagem de programação Perl, a ferramenta SRILM Toolkit e a ferramenta SWIG. Ora, para se efectuar o processo acima descrito serão usados vários recursos:

1. Bases de n-gramas e Corporas Monolíngue;
2. Modelos de Língua.

Embora a verdadeira secção de contribuições apareça no final do documento optou-se por incluir um resumo para ajudar a leitura.

As contribuições deste trabalho podem ser divididas em três categorias: contribuições científicas, contribuições tecnológicas, e recursos linguísticos:

- as contribuições científicas mais relevantes podem ser vistas ao nível da análise de diferentes ferramentas para o treino, teste e uso dos respectivos Modelos de Língua (ngram, ngram-count, SRILM Toolkit, SWIG).
- as contribuições tecnológicas podem ser resumidas pelo módulo: Lingua::srilm.
- os recursos linguísticos são os vários Modelos de Língua criados com o SRILM Toolkit. Estes recursos irão ser disponibilizados online brevemente.

1.3 Estrutura do Documento

O resto do documento está estruturado da seguinte maneira: no Capítulo 2, é fornecida uma visão geral dos fundamentos teóricos necessários para este projecto e é apresentado o conceito de Modelo de Língua e investigado o *estado-da-arte*, para selecção de ferramentas e técnicas a serem usadas no projecto. Capítulo 3 é dedicado á apresentação do pacote de software SRILM Toolkit que irá ser utilizado no treino, teste e criação de Modelos de Língua. No Capítulo 4 é proposta a solução para construção do módulo Lingua::srilm. Uma discussão e futura direcção deste trabalho concluem esta tese.

1.4 Especificações Técnicas

Todos os testes apresentados neste documento foram feitos num computador com as seguintes especificações:

- Linux
- 1 GB RAM
- Intel Processor (1,73 GHz)

- C++ compiler: GCC

De referir que este projecto exige uma grande esforço computacional, dado que estamos a tratar recursos de grande dimensões.

2
State-Of-Art

2.1 Introdução

É importante referir que a primeira fase do projecto foi dedicada completamente à investigaçao. Foram analisados vários recursos, como por exemplo, os algorítmos descritos em (Simões, A. 2008). Foi analisada também a ferramenta NATools, desenvolvida por Alberto Simões no âmbito do seu projecto de doutoramento, ferramenta essa que voltará a ser referida mais à frente. Dentro desta analise, foram adquiridos conhecimentos sobre conceitos como:

1. corpus
2. corpora seja monolíngue ou multilingue(comparável ou paralelo).

Por corpus deve entender-se como sendo uma colecção finita de textos, relativos a determinado assunto. Corpora é o plural de Corpus.

Por recurso bilingue os mais desatentos devem entender como "sendo informação que existe relativa a duas línguas diferentes e que está relacionada entre si,seja por tradução ou por semelhança". Neste projecto em particular vamos apenas usar recursos monolíngue.

É habitual a disponibilização de Corpora para consulta na Internet.

Exemplos de Referências

1. AC/DC — Acesso a Corpora / Disponibilização de Corpora [1]

2. COMPARA — Corpus Paralelo de Obras Literárias [2]

Entre as várias aplicações de Corpora está a criação de Modelos Estatísticos de Língua. Neste projecto pretende-se também usar os seguintes recursos:

	Recursos Usados
R1	Corpora Monolíngue
R2	N-gramas

Tabela 2.1: Recursos Usados

2.2 NATools

O NATools[2] foi uma ferramenta desenvolvida por Alberto Simões. O NATools (Natura Alingnment Tools) é um pacote de software para processamento de corpora paralelos mas que surgiu como ferramenta de extracção de dicionários probabilísticos de tradução. É uma ferramenta *open-source* que inclui várias ferramentas:

- um alinhador á frase;
- um extractor de PTD;
- ferramentas para consulta de recursos na Web;
- uma linguagem para a especificação de padrões de alinhamento;
- dois extractores de exemplos de tradução;
- ferramentas para generalização de exemplos;
- um servidor/biblioteca para disponibilização eficiente de recursos;
- uma API C e Perl para manuseamento dos objectos criados;

[1] Disponiveis em: http://www.linguateca.pt/acesso
[2] Disponivel em: http://www.linguateca.pt/COMPARA

2.3 Modelos de Língua

Modelo de Língua (de uma Língua L) pode ser representado como uma função f que recebe um texto T e devolve um valor percentual de 0 a 1. Este valor é perto de 1 se o texto T se parece com texto na Língua L. O valor é perto de 0 se o texto T não se parece com texto na Língua L.

O que se pretende, não é olhar para as palavras e ver se elas pertencem à língua L, mas ver se a ordem é a esperada.

Considere-se estas duas frases:

- A) O gato comeu duas sardinhas frescas.
- B) O comeu duas gato frescas sardinhas.

Embora as duas frases tenham as mesmas palavras, é de esperar que a função f aplicada à frase A dê um valor perto de 1 e que a aplicada à B dê um valor perto de 0. O problema é descobrir como se implementa uma função destas. Não existe um algorítmo ”certo”, existem é diferentes abordagens. A abordagem mais comum é usar uma base de n-gramas: sequências de palavras, do género de:

- O gato gordo é: 50 ocorrências
- Gato gordo é mau: 20 ocorrências

E assim sucessivamente. Assim será fácil entender que se só existissem estas duas frases no nosso vocabulário, a probabilidade da frase *”O gato gordo é”* seria $\frac{20}{70}$.

2.4 Recursos Utilizados

Fica aqui uma breve nota sobre os recursos usados no treino e teste de um Modelo de Língua.

2.4.1 Corpora Monolíngue

Corpora Monolíngue como a palavra sugere, são textos numa só língua. Exemplo de Corpora Monolíngue é o corpus CETEMPublico[3].

2.4.2 N-gramas

As bases de N-gramas representam um tipo particular de corpora monolíngue muito usados para, por exemplo, o treino e criação de modelos de língua. Existem várias bases dependendo do *N* usado.

2.5 Problemas na Criação de Modelos de Língua

Como já foi referido, as bases de N-gramas, um tipo particular de corpora monolíngue são usadas para a criação de modelos de língua. O maior problema subjacente a isto está associado ao facto de os modelos gerados serem em termos de tamanho muito superiores as bases de N-gramas usadas para os gerar. Este factor faz com que sejam necessárias máquinas com maior capacidade de memória para obter melhores desempenhos.

2.5.1 Modelos de Língua Estatísticos

Vamos agora fazer uma análise ao estado da arte relativamente a Modelos de Língua. A qualidade dos modelos n-gramas depende da ordem dos n-gramas que foi escolhida quando o modelo foi treinado. Como a tradução por computador se tornou muito importante investigaram-se extensões para melhorar a qualidade do Modelo de Língua. A tradução baseada em estatística provou ser um bom método para criar traduções boas dado um bom corpus. Um sistema de tradução baseada em estatística está dividido em dois módulos:

- o Modelo de Tradução;
- o Modelo de Língua.

O Modelo de Tradução recebe numa dada sequência e cria várias hipóteses de tradução juntamente com um valor que descreve a probabilidade da hipótese de tradução. Depois, as hipóteses de tradução são enviadas para um Modelo de Língua Estatístico que escolhe as hipóteses com maior probabilidade e atribui um novo valor que descreve a correcção das hipóteses num contexto de linguagem natural. A combinação do peso destes valores é então usada para determinar qual a melhor tradução. A expressão "Modelo de Língua Estatístico" descreve uma família de Modelos de Língua que modelam linguagens naturais usando as propriedades estatísticas dos n-gramas. Para estes modelos assume-se que cada palavra apenas depende do contexto das (n-1) palavras em vez de depender do corpus todo. Esta caracteristica do modelo Markov[4] simplifica muito o problema do treino de Modelos de Língua e ainda nos permite usar modelação de língua para a criação de sistemas de tradução baseados em estatística. Experiências empíricas mostraram que os Modelos de Língua estatísticos podem ser usados para criar traduções usáveis. O SRILM permite criar e aplicar Modelos de Língua Estatísticos para usar num sistema de tradução estatístico, reconhecimento de fala, etiquetação estatística e segmentação. Vejamos por tópicos:

- O objectivo da modelação estatística de língua é construir um Modelo Estatístico da Língua que possa estimar a distribuição da linguagem natural o mais exacto possível.
- Um Modelo Estatístico de Língua (SLM em inglês) é uma distribuição de probabilidade P(s) sobre uma frase S que tenta mostrar a frequência com que essa sequência de caracteres ocorre.

- Através da amostra de vários fenómenos da linguagem em termos de simples parâmetros num Modelo Estatístico, os SLM's fornecem um metódo fácil para lidar com a complexidade da linguagem natural no computador.
- A aplicação original (e ainda a mais importante) dos SLM's é o reconhecimento de fala, mas também desempenham papéis importantes noutras aplicações da linguagem natural, tão diversas como tradução por computador, anotação de discurso, método de entrada inteligente e sistemas de conversão de texto para voz (por exemplo, as chamadas dos aeroportos).

Para tradução por computador, os Modelos de Língua criados pelo SRILM são actualmente o standard preferido[5].

2.6 Ferramentas para a Geração de Modelos de Língua

Como já foi mencionado, neste tipo de projectos relacionadas com PLN, existe uma grande componente de leitura, investigação e muitos testes, não existem algorítmos ou estratégias fixas, mas sim diferentes abordagens. Foram analisadas as seguintes ferramentas:

- As ferramentas ngram (variantes test e build)[6] que permitem o treino e teste de Modelos de Língua;
- O pacote de software SRILM - The SRI Language Modeling Toolkit[7] usado também para treino e teste de Modelos de Língua, que vai ser analisado ao pormenor no capítulo seguinte;
- O pacote de software NLTK[8], que representa um conjunto de recursos (criados na linguagem de scripting Python) para serem usados na pesquisa e desenvolvimento na área do Processamento de Linguagem Natural. Esta ferramenta ainda não permite o treino e teste de Modelos de Língua mas permite várias operações com Modelos de Língua que já estejam criados.

Por fim optou-se por usar no projecto a ferramenta SRILM dado que se apresentou como a mais bem documentada, fácil e simples de trabalhar.

2.6.1 Teste de Performance

A qualidade dos Modelos de Língua que usam n-gramas pode ser melhorada através do treino desses modelos usando corpora de grandes dimensões. Isto é uma propriedade da natureza estatística da aproximação que usamos. Como já foi dito, ao longo dos anos, conforme foi aumentando o volume de recursos extraídos disponíveis, esses recursos podem ser usados para melhores Modelos Estatísticos de Língua. Como já foi discutido anteriormente, tornou-se mais fácil dispor de grandes quantidades de recursos monolíngue para criar Modelos de Língua. Isso permite usar tetra-gramas ou penta-gramas em vez de apenas tri-gramas. Como os Modelos de Língua actuais são gerados a partir de uma quantidade enorme de recursos que são extraídos, vão ser necessárias grandes

quantidades de memória para serem usados nos sistemas actuais. O problema da implementação de Modelos de Língua baseia-se no facto de que os recursos que estão nos n-gramas têm que ser carregados em memória ao mesmo tempo, mesmo que só seja precisa uma pequena fracção destes recursos para o processamento. A memória do computador tem vindo a ficar mais barata embora continue a ser um recurso caro, especialmente se comparado com o preço dos discos rígidos. Ou seja, será um bom caminho investigar novos métodos para implementação de Modelos de Língua que reduzam o uso da memória do computador. Em vez de usar apenas uma máquina com muita memória para executar uma tarefa, distribui-se essa tarefas por várias máquinas que executam partes do processamento requerido. Dentro do Modelo de Língua SRI, todas estas máquinas têm que ter uma quantidade de memória elevada para tratar um Modelo de Língua grande. Por isso não é de estranhar que há não muito tempo que o software usado para tratamento de Modelos de Língua por computador usando estatística era caro, fechado e inflexível.

2.7 Conclusões

Os Modelos Estatísticos de Língua são utilizados em várias áreas desde o reconhecimento de fala (a mais importante), mas também desempenham papéis importantes noutras aplicações da linguagem natural, tão diversas como tradução por computador, anotação de discurso, método de entrada inteligente e sistemas de conversão de texto para voz. As ferramentas actuais de criação modelos de língua usam bases de n-gramas que são ficheiros em que cada linha é uma sequência de N-palavras, com o N=3 a ser actualmente, a base mais utilizada. O estado-da-arte bases de n-gramas é actualmente representado por N=5. Embora se possa usar uma base de valor N superior, por *default*, o SRILM usa N=3. A ferramenta SRILM apresenta-se como sendo a simples, robusta e fácil de manusear, sendo a escolhida para o treino e teste de Modelos de Língua Estatísticos.

3

SRILM - SRI Language Modeling Toolkit

3.1 Descrição

Como se disse anteriormente, o SRILM é um pacote de software para criação e aplicação de Modelos de Língua Estatisticos, muito usado no reconhecimento de fala, etiquetação estatística, segmentação e tradução por computador. Está, desde 1995, em desenvolvimento no SRI Speech Technology and Research Laboratory. O SRILM usa um Modelo de Língua baseado em n-gramas ou construído a partir dum corpus.

A Figura 1 mostra o esquema de execução do SRILM.

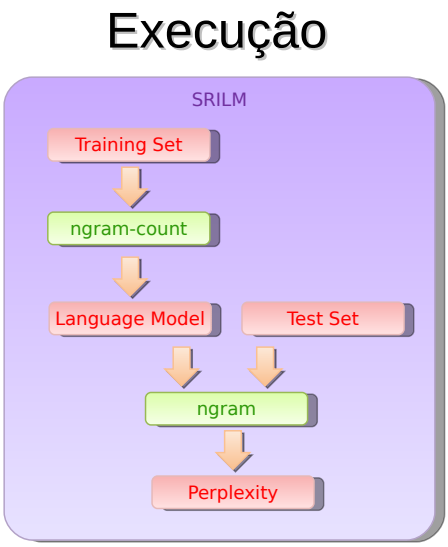

Figura 3.1: Esquema de execução do SRILM.

3.1.1 Características

SRILM é composto pelos seguintes componentes:

- um conjunto de bibliotecas de classes C++ que implementam Modelos de Língua, suportam estruturas de dados e funções de utilidade geral.
- um conjunto de programas desenvolvidos em cima destas bibliotecas para executar tarefas como treinar Modelos de Língua e testá-los com dados, etiquetação e segmentação.
- ngram-count: cria bases de n-gramas e estima Modelos de Língua.
- ngram-merge: faz a junção de várias bases de n-gramas (apenas necessário para corpora de grandes dimensões).
- ngram: calcula probablidades e perplexidades dado um Modelo de Língua (também permite o trabalho conjunto de vários modelos).
- um conjunto de scripts gerais que facilitam tarefas menores.

3.2 Smoothing/Discount

Consideremos que questionamos um Modelo de Língua com a seguinte frase:

- Olá World.

E como resposta ele nos diz que a frase não existe, isto é, tem probabilidade igual a zero ou muito baixa. A palavra *World* não existe na língua portuguesa ou pelo menos não existe no corpus que foi usado para treino do Modelo de Língua. O SRILM resolve este problema usando técnicas de *smooth* e *discount*. Estas técnicas permitem, como a tradução nos faz entender *suavizar* o modelo, ou seja, torná-lo menos rígido. Basicamente, são técnicas que permitem que mesmo que uma palavra que não existe no corpus de treino (e pode vir a existir um dia...), o Modelo de Língua não atribua probabilidade zero. Senão vejamos, a palavra *Olá* que aparece, por exemplo, mil vezes num corpus. A palavra *World* não aparece nenhuma. Então atribui-se um valor de contagem, por exemplo, 998 à palavra *Olá*. A palavra *World* teria um valor de contagem igual a 2. Para terminar assim se percebe que a frase *Olá World* já não teria valor zero de probabilidade (ou próximo disso) e o modelo ficaria assim suavizado. Estas técnicas estão descritas na bibliografia aconselhada para o SRILM[7].

3.3 Modelos N-gram

Como foi referido os Modelos de Língua são criados tendo como base de treino, bases de n-gramas (uni-gramas, bi-gramas, tri-gramas, tetra-gramas) ou corpus monolíngue. O SRILM não trabalha com classes de palavras[1], isto é, não permite a criação de *clusters* de palavras. Estes *clusters*

permitiriam resolver o problema da esparsidade dos dados. Isto permitiria estimar parâmetros para classes de palavras em vez de palavras separadas. Através da criação de classes de palavras seria possivel reduzir significativamente o tamanho do Modelo de Língua, embora a perplexidade[9] aumentasse ligeiramente. As bases de n-gramas são ficheiro organizados linha-a-linha do tipo:

- (Eu estou aqui), neste caso seria uma linha possivel nessa base 3-gramas.

3.3.1 Operações Possíveis com Modelos de Língua

Na pesquisa foi necessário entrar em contacto com as pessoas que criaram o SRILM, nomeadamente Andreas Stolcke. Do contacto foram criados vários Modelos de Língua e aprendeu-se a testar esses modelos. Seguidamente, apresentam-se os comandos pertencentes ao SRILM de maior relevo para o projecto em causa:

- O comando `ngram-count -read COUNTS -lm LM` cria um Modelo de Língua **LM** usando uma base de n-gramas de ordem 3 extraída para um ficheiro **COUNTS**.
- O comando `ngram-count -text CORPUS -lm LM` cria um Modelo de Língua **LM** usando um corpus **CORPUS**.
- O comando `ngram -lm LMFILE -debug 2 -ppl TEXT` avalia a probabilidade de uma ou mais frases dado o modelo **LM** onde **TEXT** contêm uma ou mais frases (uma por linha). Estes comandos acima descritos possuem várias opções.

O comando `ngram-count -order 3 -text teste.txt -lm LM` produz o seguinte output:

```
warning: discount coeff 1 is out of range: 0
warning: count of count 8 is zero -- lowering maxcount
warning: count of count 7 is zero -- lowering maxcount
warning: count of count 6 is zero -- lowering maxcount
warning: count of count 5 is zero -- lowering maxcount
warning: count of count 4 is zero -- lowering maxcount
warning: count of count 3 is zero -- lowering maxcount
warning: count of count 2 is zero -- lowering maxcount
GT discounting disabled
warning: count of count 8 is zero -- lowering maxcount
warning: count of count 7 is zero -- lowering maxcount
warning: count of count 6 is zero -- lowering maxcount
warning: count of count 5 is zero -- lowering maxcount
warning: count of count 4 is zero -- lowering maxcount
warning: count of count 3 is zero -- lowering maxcount
warning: count of count 2 is zero -- lowering maxcount
GT discounting disabled
```

Vamos agora explicar o significado deste resultado. Para além dos vários *warnings* que aparecem no output final, a linha *GT discounting disabled* necessita ser explicada, até porque os vários *warnings* estão relacionadas com a mesma. Basicamente, a linha *GT discounting disabled* está a indicar-nos que o método Good Turing[7] para suavizar o modelo não está activado. O método GT representa o método *default* do SRILM. Podemos ver este exemplo como sendo um dos piores exemplos de teste (pior caso), dado que apenas estamos a treinar o modelo de língua como um corpus de tamanho reduzido e claro, sem activação de técnicas de *discounting* que iriam claro permitir suavizar o modelo de língua.
O comando `ngram -lm LM -debug 2 -ppl teste2.txt` produz o seguinte output:

```
reading 6 1-grams
reading 6 2-grams
reading 0 3-grams
Espero eu não.
p( $<$unk$>$ | $<$s$>$ )  = [OOV] 0 [ -inf ]
p( $<$unk$>$ | $<$unk$>$ ...)  = [OOV] 0 [ -inf ]
p( $<$unk$>$ | $<$unk$>$ ...)  = [OOV] 0 [ -inf ]
p( $<$/s$>$ | $<$unk$>$ ...)  = [1gram] 0.314286 [ -0.502675 ]
1 sentences, 3 words, 3 OOVs
0 zeroprobs, logprob= -0.502675 ppl= 3.18182 ppl1= undefined

file teste2.txt: 1 sentences, 3 words, 3 OOVs
0 zeroprobs, logprob= -0.502675 ppl= 3.18182 ppl1= undefined
```

Vamos explicar o resultado deste comando. As estatísticas referentes às duas ultimas linhas têm os seguinte significado:

- **3 OOV's** Isto representa o número de palavras desconhecidas, isto é, o número de palavras que aparecem no corpus de teste mas não aparecem no corpus de treino.
- **logprob= -0.502675** Isto representa o logarítmo total de probabilidade ignorando as 3 palavras desconhecidas. Dado que o SRILM no calculo de probabilidades de frases adiciona o *token* <s> no inicio de cada frase e </s> no fim, este logarítmo total já inclui a probabilidade atribuida a </s>. Logo o número total de palavras em que este logarítmo se baseia pode ser visto como `words - OOVs + sentences = 3 - 3 + 1`.
- **ppl =3.18182** Isto representa a média aritmética de $\frac{1}{Probabilidade}$ de cada palavra, isto é, a perplexidade[9]. A perplexidade pode ser vista como o número médio de palavras que podem vir a seguir a uma sequência de palavras. A perplexidade será explicada em pormenor no capitulo seguinte e pode ser vista como: $ppl=10^{(\frac{-logprob}{words-OOVs+sentences})}$.
- **ppl1** Isto representa a perplexidade média por palavra excluindo o *token* </s>. A expressão exacta é : $ppl1=10^{(\frac{-logprob}{words-OOVs})}$.

Para terminar vamos agora explicar o elemento *unk* que aparece no resultado final. Este elemento aparece no resultado final como substituto de palavras que não existam no corpus de treino, significa *unknown* (desconhecida).

Todos os problemas acima descritos seriam resolvidos da seguinte maneira:

- Para o treino seria usado o comando `ngram-count -unkdiscount -interpolate -order 3 -unk -text teste.txt -tolower -lm LM`, usando assim o método de *discount* Kneyser-Ney (`-unkdiscount`) juntamente com `-unk` para evitar os elementos *unk* descritos acima. Este ultimo elemento permite que o modelo de língua atribua uma probabilidade diferente de zero a palavras desconhecidas.
- Para o teste seria usado o comando `ngram -lm LM -debug 2 -ppl teste2.txt`.

3.4 Arquitectura Cliente-Servidor

O SRILM possui uma arquitectura cliente-servidor para situações de uso intensivo de um Modelo de Língua.
Considere-se então um servidor a executar de seguinte maneira:

- `ngram -lm LM -server-port 100 -order 3 &`.

Vamos agora ver o resultado de uma execução do servidor:

```
starting prob server on port 100
```

O cliente teria que se ligar ao servidor através do seguinte comando:

- `ngram -use-server 100127.0.0.1 -cache-served-ngrams -ppl ficheiroteste -debug 2 > & output.txt`

O ficheiro de teste usado contém as seguintes frases:

```
Entretanto , gostaria - como também me foi pedido por um berto número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
 nos diferentes países da União Europeia que foram afectados .
Entretanto , gostaria - como também me foi pedido por um perto número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
 nos diferentes países da União Europeia que foram afectados .
Entretanto , gostaria - como também me foi pedido por um certo número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
 nos diferentes países da União Europeia que foram afectados .
```

Vamos agora ver o resultado de uma execução do cliente:

```
client 38179@127.0.0.1: connection accepted
client 38179@127.0.0.1: 48 probabilities served
```

Para terminar iremos apresentar o output do processamento efectuado:

```
server 100@127.0.0.1: probserver ready
Entretanto , gostaria - como também me foi pedido por um berto número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
nos diferentes países da União Europeia que foram afectados .
p( Entretanto | <s> )  =  8.49102e-05 [ -4.07104 ]
p( , | Entretanto ...)  =  3.2184e-06 [ -5.49236 ]
p( gostaria | , ...)  =  7.06318e-11 [ -10.151 ]
p( - | gostaria ...)  =  8.13224e-08 [ -7.08979 ]
p( como | - ...)  =  2.85049e-07 [ -6.54508 ]
p( também | como ...)  =  4.05163e-09 [ -8.39237 ]
p( me | também ...)  =  8.49376e-10 [ -9.0709 ]
p( foi | me ...)  =  4.47085e-08 [ -7.34961 ]
p( pedido | foi ...)  =  4.05975e-10 [ -9.3915 ]
p( por | pedido ...)  =  2.5038e-07 [ -6.6014 ]
p( um | por ...)  =  2.56195e-08 [ -7.59143 ]
p( berto | um ...)  =  4.86743e-14 [ -13.3127 ]
p( número | berto ...)  =  5.96197e-05 [ -4.22461 ]
p( de | número ...)  =  5.85882e-07 [ -6.23219 ]
p( colegas | de ...)  =  1.32782e-10 [ -9.87686 ]
p( - | colegas ...)  =  4.35752e-08 [ -7.36076 ]
p( que | - ...)  =  2.08848e-06 [ -5.68017 ]
p( observássemos | que ...)  =  0 [ -inf ]
p( um | observássemos ...)  =  0.00380803 [ -2.4193 ]
p( minuto | um ...)  =  8.60995e-11 [ -10.065 ]
p( de | minuto ...)  =  7.52107e-06 [ -5.12372 ]
p( silêncio | de ...)  =  1.464e-10 [ -9.83446 ]
p( por | silêncio ...)  =  5.40082e-07 [ -6.26754 ]
p( todas | por ...)  =  1.00466e-09 [ -8.99798 ]
p( as | todas ...)  =  7.24369e-08 [ -7.14004 ]
p( vítimas | as ...)  =  2.70166e-10 [ -9.56837 ]
p( , | vítimas ...)  =  4.01985e-06 [ -5.39579 ]
p( nomeadamente | , ...)  =  4.2063e-10 [ -9.3761 ]
p( das | nomeadamente ...)  =  1.19201e-07 [ -6.92372 ]
p( tempestades | das ...)  =  1.25372e-11 [ -10.9018 ]
p( , | tempestades ...)  =  8.78295e-05 [ -4.05636 ]
p( nos | , ...)  =  4.62882e-09 [ -8.33453 ]
p( diferentes | nos ...)  =  7.62009e-10 [ -9.11804 ]
```

```
p( países | diferentes ...)  =  1.3381e-08 [ -7.87351 ]
p( da | países ...)  =  1.7112e-07 [ -6.7667 ]
p( União | da ...)  =  5.95703e-10 [ -9.22497 ]
p( Europeia | União ...)  =  3.99402e-09 [ -8.39859 ]
p( que | Europeia ...)  =  5.5077e-07 [ -6.25903 ]
p( foram | que ...)  =  2.21804e-09 [ -8.65403 ]
p( afectados | foram ...)  =  7.34514e-11 [ -10.134 ]
p( . | afectados ...)  =  1.17649e-05 [ -4.92941 ]
p( </s> | . ...)  =  0.999997 [ -1.32346e-06 ]
1 sentences, 41 words, 0 OOVs
1 zeroprobs, logprob= -304.197 ppl= 2.62684e+07 ppl1= 4.02643e+07

Entretanto , gostaria - como também me foi pedido por um perto número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
nos diferentes países da União Europeia que foram afectados .
p( Entretanto | <s> )  =  8.49102e-05 [ -4.07104 ]
p( , | Entretanto ...)  =  3.2184e-06 [ -5.49236 ]
p( gostaria | , ...)  =  7.06318e-11 [ -10.151 ]
p( - | gostaria ...)  =  8.13224e-08 [ -7.08979 ]
p( como | - ...)  =  2.85049e-07 [ -6.54508 ]
p( também | como ...)  =  4.05163e-09 [ -8.39237 ]
p( me | também ...)  =  8.49376e-10 [ -9.0709 ]
p( foi | me ...)  =  4.47085e-08 [ -7.34961 ]
p( pedido | foi ...)  =  4.05975e-10 [ -9.3915 ]
p( por | pedido ...)  =  2.5038e-07 [ -6.6014 ]
p( um | por ...)  =  2.56195e-08 [ -7.59143 ]
p( perto | um ...)  =  4.29596e-10 [ -9.36694 ]
p( número | perto ...)  =  1.18038e-08 [ -7.92798 ]
p( de | número ...)  =  5.85882e-07 [ -6.23219 ]
p( colegas | de ...)  =  1.32782e-10 [ -9.87686 ]
p( - | colegas ...)  =  4.35752e-08 [ -7.36076 ]
p( que | - ...)  =  2.08848e-06 [ -5.68017 ]
p( observássemos | que ...)  =  0 [ -inf ]
p( um | observássemos ...)  =  0.00380803 [ -2.4193 ]
p( minuto | um ...)  =  8.60995e-11 [ -10.065 ]
p( de | minuto ...)  =  7.52107e-06 [ -5.12372 ]
p( silêncio | de ...)  =  1.464e-10 [ -9.83446 ]
p( por | silêncio ...)  =  5.40082e-07 [ -6.26754 ]
```

```
p( todas | por ...)  =  1.00466e-09 [ -8.99798 ]
p( as | todas ...)  =  7.24369e-08 [ -7.14004 ]
p( vítimas | as ...)  =  2.70166e-10 [ -9.56837 ]
p( , | vítimas ...)  =  4.01985e-06 [ -5.39579 ]
p( nomeadamente | , ...)  =  4.2063e-10 [ -9.3761 ]
p( das | nomeadamente ...)  =  1.19201e-07 [ -6.92372 ]
p( tempestades | das ...)  =  1.25372e-11 [ -10.9018 ]
p( , | tempestades ...)  =  8.78295e-05 [ -4.05636 ]
p( nos | , ...)  =  4.62882e-09 [ -8.33453 ]
p( diferentes | nos ...)  =  7.62009e-10 [ -9.11804 ]
p( países | diferentes ...)  =  1.3381e-08 [ -7.87351 ]
p( da | países ...)  =  1.7112e-07 [ -6.7667 ]
p( União | da ...)  =  5.95703e-10 [ -9.22497 ]
p( Europeia | União ...)  =  3.99402e-09 [ -8.39859 ]
p( que | Europeia ...)  =  5.5077e-07 [ -6.25903 ]
p( foram | que ...)  =  2.21804e-09 [ -8.65403 ]
p( afectados | foram ...)  =  7.34514e-11 [ -10.134 ]
p( . | afectados ...)  =  1.17649e-05 [ -4.92941 ]
p( </s> | . ...)  =  0.999997 [ -1.32346e-06 ]
1 sentences, 41 words, 0 OOVs
1 zeroprobs, logprob= -303.954 ppl= 2.59133e+07 ppl1= 3.97064e+07

Entretanto , gostaria - como também me foi pedido por um certo número de colegas -
que observássemos um minuto de silêncio por todas as vítimas ,
nomeadamente das tempestades ,
nos diferentes países da União Europeia que foram afectados .
p( Entretanto | <s> )  =  8.49102e-05 [ -4.07104 ]
p( , | Entretanto ...)  =  3.2184e-06 [ -5.49236 ]
p( gostaria | , ...)  =  7.06318e-11 [ -10.151 ]
p( - | gostaria ...)  =  8.13224e-08 [ -7.08979 ]
p( como | - ...)  =  2.85049e-07 [ -6.54508 ]
p( também | como ...)  =  4.05163e-09 [ -8.39237 ]
p( me | também ...)  =  8.49376e-10 [ -9.0709 ]
p( foi | me ...)  =  4.47085e-08 [ -7.34961 ]
p( pedido | foi ...)  =  4.05975e-10 [ -9.3915 ]
p( por | pedido ...)  =  2.5038e-07 [ -6.6014 ]
p( um | por ...)  =  2.56195e-08 [ -7.59143 ]
p( certo | um ...)  =  4.45123e-10 [ -9.35152 ]
p( número | certo ...)  =  1.1392e-08 [ -7.9434 ]
```

```
p( de | número ...)  =  5.85882e-07 [ -6.23219 ]
p( colegas | de ...)  =  1.32782e-10 [ -9.87686 ]
p( - | colegas ...)  =  4.35752e-08 [ -7.36076 ]
p( que | - ...)  =  2.08848e-06 [ -5.68017 ]
p( observássemos | que ...)  =  0 [ -inf ]
p( um | observássemos ...)  =  0.00380803 [ -2.4193 ]
p( minuto | um ...)  =  8.60995e-11 [ -10.065 ]
p( de | minuto ...)  =  7.52107e-06 [ -5.12372 ]
p( silêncio | de ...)  =  1.464e-10 [ -9.83446 ]
p( por | silêncio ...)  =  5.40082e-07 [ -6.26754 ]
p( todas | por ...)  =  1.00466e-09 [ -8.99798 ]
p( as | todas ...)  =  7.24369e-08 [ -7.14004 ]
p( vítimas | as ...)  =  2.70166e-10 [ -9.56837 ]
p( , | vítimas ...)  =  4.01985e-06 [ -5.39579 ]
p( nomeadamente | , ...)  =  4.2063e-10 [ -9.3761 ]
p( das | nomeadamente ...)  =  1.19201e-07 [ -6.92372 ]
p( tempestades | das ...)  =  1.25372e-11 [ -10.9018 ]
p( , | tempestades ...)  =  8.78295e-05 [ -4.05636 ]
p( nos | , ...)  =  4.62882e-09 [ -8.33453 ]
p( diferentes | nos ...)  =  7.62009e-10 [ -9.11804 ]
p( países | diferentes ...)  =  1.3381e-08 [ -7.87351 ]
p( da | países ...)  =  1.7112e-07 [ -6.7667 ]
p( União | da ...)  =  5.95703e-10 [ -9.22497 ]
p( Europeia | União ...)  =  3.99402e-09 [ -8.39859 ]
p( que | Europeia ...)  =  5.5077e-07 [ -6.25903 ]
p( foram | que ...)  =  2.21804e-09 [ -8.65403 ]
p( afectados | foram ...)  =  7.34514e-11 [ -10.134 ]
p( . | afectados ...)  =  1.17649e-05 [ -4.92941 ]
p( </s> | . ...)  =  0.999997 [ -1.32346e-06 ]
1 sentences, 41 words, 0 OOVs
1 zeroprobs, logprob= -303.954 ppl= 2.59133e+07 ppl1= 3.97064e+07

file input1.txt: 3 sentences, 123 words, 0 OOVs
3 zeroprobs, logprob= -912.106 ppl= 2.60311e+07 ppl1= 3.98915e+07
```

Uma conclusão a retirar do uso da arquitectura Cliente-Servidor do SRILM é o facto de não aparecerem OOVs (Out Of Vocabulary Words), porque todas as palavras são implicitamente adicionadas ao vocabulário. Outra conclusão a retirar do uso desta arquitectura tem a ver com o facto destes OOVs serem contados como *zeroprobs*, ou seja, palavras com probabilidade zero. Para terminar é

importante referir que tanto os OOVs como *zeroprobs* são equivalentes no calculo da perplexidade, ou seja, não interferem.

3.5 Conclusões

O SRILM usa como *defaults* ordem N=3 e método de discounting GT (Good Turing). O SRILM não trabalha com classes de palavras. O SRILM possui um conjunto de funcionalidades vasto mas apenas uma pequena porção vai se usada neste projecto. Desde logo serão usados no projecto os comandos `ngram-count` e `ngram` para treino e teste de Modelos de Língua. O SRILM possui várias técnicas para suavização do Modelo de Língua, ou seja, torná-lo menos rígido. Por isso e depois de todos testes realizados decidiu-se usar o método de *discount* Kneyser-Ney. O SRILM possui uma arquitectura Cliente-Servidor que permite mecanismos de cache que permite aumentos de performance ao guardar os registos das pesquisas no Cliente, ou seja, permite criar um servidor de probabilidades que permita responder a vários clientes e guardar as probabilidades em cada cliente. Para tradução por computador, os Modelos de Língua criados pelo SRILM são actualmente muito utilizados. São por exemplo, usados pelo sistema de Tradução MOSES[10].

4

Lingua::srilm

A criação de modelos estatísticos de língua baseados em n-gramas é uma técnica muito usado no Processamento de Língua Natural, nomeadamente, para obter a correcção e a fluência de, por exemplo, uma frase em qualquer língua. É uma técnica muito usada em aplicações relacionadas com a Tradução Automática. No entanto, a ferramenta mais usada actualmente para a construção de Modelos de Língua, o SRILM, é escrito totalmente em C++. Ora isto apresenta-se como um problema para o investigador/criador que trabalha com uma linguagem de scripting como o Perl[11]. Nesta fase do documento vamos procurar explicar como construir um interface, usando o SWIG[12], para o SRILM de modo que através da linguagem Perl, os Modelos de Língua possam ser usados. Vamos criar o módulo Lingua::srilm, que demonstre o uso deste interface em Perl. Finalmente iremos criar uma arquitectura Cliente-Servidor que irá permitir que o Modelo de Língua seja usado por vários clientes. Sendo assim o Modelo de Língua é apenas carregado uma vez em memória e o servidor pode satisfazer muitos pedidos.

4.1 SWIG-SRILM

O SRILM é uma ferramenta que torna fácil a tarefa de construção e uso de Modelos de Língua em larga escala. Outro facto importante é a permissão de ver o código-fonte deste pacote de software, desde estruturas de memória, header files e bibliotecas. No entanto este pacote está escrito em C++. Como estamos a trabalhar em Perl isto torna-se um problema, dado que pretendemos usar um Modelo de Língua criado pelo SRILM. Uma maneira elegante de ultrapassar este desafio é aplicar o Simplified Wrapper and Interface Generator (SWIG) para criar um interface usando a

API do SRILM que permita o carregamento e uso dos Modelos de Língua directamente no código Perl.

4.2 Introdução ao SWIG

Esta parte do documento descreve as bases do SWIG e mostra como pode ser usado para construir um interface para usar Modelos de Língua. O SWIG é uma ferramenta extremamente útil para criar interfaces para ligação de pacotes de software, como o SRILM, que não permite a ligação a uma variedade de linguagens de scripting como Perl. SWIG é um projecto open-source que está abrangido pelas regras da General Public Licence. O uso do SWIG é muito simples. Se já existe um programa escrito em C que já execute as operações que pretendemos, a única coisa que temos que fazer é criar um ficheiro *interface* para o SWIG executar. Vamos apresentar em seguida um exemplo de um interface para o SWIG e depois o caso prático que usamos para o nosso projecto.

- A directiva %module define o nome do módulo que irá ser criado pelo SWIG.
- A directiva %...%block fornece uma ligação para inserir código adicional para os header file ou declarações adicionais baseadas em C.
- A seguir as funções que vão aparecer no interface Perl são declaradas.

Fornecidos o programa em C e o *interface file* para o SWIG processar, o futuro módulo Perl pode agora ser facilmente compilado e usado como indica a Listagem 4.2 abaixo. Esta Listagem inclui comentários que explicam cada comando em detalhe. Para saber mais acerca do SWIG são aconselhados alguns elementos, como o site oficial do SWIG[12], entre outros, que pode consultar na bibliografia.

```
%module srilm
%{
/* Include the header files etc. here */
#include "Ngram.h"
#include "Vocab.h"
#include "Prob.h"
#include "srilm.h"
#include "LMClient.h"
%}

%include srilm.h
```

Excerto de Código 4.1: SWIG interface file para o ficheiro srilm.c

No excerto de código 4.1 é demonstrado como criar um interface file para o SWIG, com o nome *swig_perl.i*.

No excerto de código 4.2, a linha começada por *swig -perl* indica ao SWIG que precisamos de um módulo Perl. Isto vai gerar um interface chamado srilm_perl_wrap.c e um módulo Perl chamado srilm.pm.

```
SRILM_LIBS=/home/Manel/Mestrado/srilm/lib/i686
SRILM_INC=/home/Manel/Mestrado/srilm/include
PERL_INC=/usr/lib/perl5/5.10.0/i386-linux-thread-multi/CORE

perl: clean srilm.so

srilm.so: srilm.o srilm_perl_wrap.o
        g++ -shared $^ -loolm -ldstruct -lmisc -L$(SRILM_LIBS) -o $@

srilm_perl_wrap.o: srilm_perl_wrap.c
        g++ -c -fpic $< -I/usr/local/include/ -I$(SRILM_INC) -I$(PERL_INC)

srilm_perl_wrap.c: srilm_perl.i
        swig -perl $<

srilm.o: srilm.c
        g++ -c -fpic $< -I/usr/local/include/ -I$(SRILM_INC) -I$(PERL_INC)

clean:
        \rm -fr srilm.o srilm_*_wrap.* srilm.so
}
```

Excerto de Código 4.2: Makefile para compilar e usar o interface file

4.3 SWIG-PERL-SRILM

Esta secção descreve como criar um interface Perl para o SRILM usando o SWIG. Assim que o program C é criado, deve ser definido um interface *file* para o SWIG processar. Escrever esta definição é uma tarefa simples assim que o conjunto de funções a ser exposto no módulo for determinado. Assim que o interface for definido, os passos definidos na Makefile são executados para compilar o módulo SRILM Perl que pode ser importado directamente no código Perl como se fosse qualquer outro módulo Perl normal. Neste ponto, o interface Perl para usar qualquer Modelo de Língua criado com o SRILM está compilado e pronto a usar em qualquer código Perl.

No excerto de código 4.3 aparece o *header file* do programa C que foi criado para gerar o interface para Perl.

```
#ifndef SRILMWRAP_H
#define SRILMWRAP_H

#ifdef __cplusplus
  extern "C" {
#else
    typedef struct Ngram Ngram; /* dummy type to stand in for class */
#endif

Ngram* initLM(int order);
void deleteLM(Ngram* ngram);
unsigned getIndexForWord(const char* s);
const char* getWordForIndex(unsigned i);
int readLM(Ngram* ngram, const char* filename);
float getWordProb(Ngram* ngram, unsigned word, unsigned* context);
float getUnigramProb(Ngram* ngram, const char* word);
float getBigramProb(Ngram* ngram, const char* ngramstr);
float getTrigramProb(Ngram* ngram, const char* ngramstr);
float getSentenceProb(Ngram* ngram, const char* sentence, unsigned length);
unsigned corpusStats(Ngram* ngram, const char* filename, TextStats &stats);
float getCorpusProb(Ngram* ngram, const char* filename);
float getCorpusPpl(Ngram* ngram, const char* filename);
int howManyNgrams(Ngram* ngram, unsigned order);
unsigned getProbServer(Ngram* lm,unsigned port, unsigned maxClients =0);
char* bestSentence(Ngram* lm,char** frases);
char* bestSentenceContext(char* frases[]);
#ifdef __cplusplus
  }
#endif

#endif
```

Excerto de Código 4.3: srilm.h

No excerto de código 4.4 aparece o código do ficheiro C que foi criado com as funções necessárias para posteriormente ser criado o interface Perl para o SRILM.

Neste ficheiro C incluem-se:

- Funções principais que vão fazer parte da extensão Perl do SRILM;
- Funções auxiliares, necessárias para processamento secundário.

Vamos apenas identificar as funções principais, que são:

- `Ngram* initLM(int order)`, que permite inicializar um Modelo de Língua com um vocabulário inicial de determinada ordem;
- `int readLM(Ngram* ngram, const char* filename)`, que carrega um ficheiro contendo um Modelo de Língua;
- `int howManyNgrams(Ngram* ngram, unsigned order)`, que retorna o número de n-gramas de ordem *order* que existem no Modelo;
- `float getUnigramProb(Ngram* ngram, const char* word)`, retorna a probabilidade um unigrama;
- `float getBigramProb(Ngram* ngram, const char* ngramstr)`, retorna a probabilidade de um bigrama;
- `float getTrigramProb(Ngram* ngram, const char* ngramstr)`, retorna a probabilidade de um trigrama;
- `float getSentenceProb(Ngram* ngram, const char* sentence, unsigned length)`, que retorna a probabilidade de uma frase com cardinal *length*;
- float getCorpusProb(Ngram* ngram, const char* filename), que retorna o logarítmo total de probabilidade dum corpus;
- `float getCorpusPpl(Ngram* ngram, const char* filename)`, que retorna a perplexidade[9] presente num corpus;

O código do ficheiro C a seguir apresentado está composto por diversos comentários para melhor percepção do que foi feito.

```
#include "Prob.h"
#include "Ngram.h"
#include "Vocab.h"
#include "srilm.h"
#include "LMClient.h"
#include <cstdio>
#include <cstring>
```

```
#include <cmath>
#include "ctype.h"

Vocab *swig_srilm_vocab;
const float BIGNEG = -99;
// Inicializa o modelo ngram
Ngram *
initLM (int order)
{
  swig_srilm_vocab = new Vocab;
  return new Ngram (*swig_srilm_vocab, order);
}

// Apaga o modelo da memoria
void
deleteLM (Ngram * ngram)
{
  delete swig_srilm_vocab;
  delete ngram;
}

// Fornece um index para uma string
unsigned
getIndexForWord (const char *s)
{
  unsigned ans;
  ans = swig_srilm_vocab->addWord ((VocabString) s);
  if (ans == Vocab_None)
    {
      printf ("Trying to get index for Vocab_None.\n");
    }
  return ans;
}

/ Fornece uma palavra para um index
const char *
getWordForIndex (unsigned i)
{
  return swig_srilm_vocab->getWord ((VocabIndex) i);
}

// Le um ficheiro LM para um modelo
int
readLM (Ngram * ngram, const char *filename)
{
  File file (filename, "r");
  if (!file)
    {
```

```
      fprintf (stderr, "Error:: Could not open file %s\n", filename);
      return 0;
    }
  else
    return ngram->read (file, 0);
}

// Get the ngram probability of the given string
/*float getNgramProb(Ngram* ngram, const char* ngramstr, unsigned order) {
    const char* words[10];
    unsigned int indices[order];
    int numparsed, histsize, i, j;
    char* scp;
    float ans, adder;

    // Duplicate string so that we don't mess up the original
    scp = strdupa(ngramstr);

    // Parse the given string into words
    numparsed = Vocab::parseWords(scp, (VocabString *)words, 10);
    if(numparsed != order) {
        fprintf(stderr, "Error: Given order %d not correct.\n", order);
        return 0;
    }

    // Get indices for the words obtained above, if you don't find them, then
        add them
    // to the vocabulary and then get the indices.
    swig_srilm_vocab->addWords((VocabString *)words, (VocabIndex *)indices,
        order);
    ans = 0;

    for(i=(order-1); i>=0; i--) {

        histsize = i;

        if(histsize == 0) {
            unsigned int hist[1] = {Vocab_None};
            if(indices[i] == swig_srilm_vocab->ssIndex()) {
                adder = LogP_One;
            }
            else {
                adder = getWordProb(ngram, indices[i], hist, cap);
                if(adder == LogP_Zero)
                    return BIGNEG;
            }
        }
        else {
```

```
            unsigned int hist[histsize+1];
            for(j=0; j<histsize; j++)
                hist[j] = indices[i-j-1];
            hist[histsize] = Vocab_None;
            adder = getWordProb(ngram, indices[i], hist, cap);
            if(adder == LogP_Zero)
                return BIGNEG;
        }
        ans += adder;
    }
    ans /= log10(exp(1.0));
    return ans;
}
*/

// Fornece a probabilidade de uma palavra
float
getWordProb (Ngram * ngram, unsigned w, unsigned *context)
{
  return (float) ngram->wordProb (w, context);
}

// Fornece um server de probabilidades...por questoes de complexidade do codigo
// optou-se por apenas incluir este metodo para exemplificar a chamada do
    servidor.
unsigned
getProbServer (Ngram * lm, unsigned port, unsigned maxClients)
{
  return lm->probServer (port, maxClients);
}

// Fornece a probabilidade de um unigram
float
getUnigramProb (Ngram * ngram, const char *word)
{
  unsigned index;
  float ans;

  // Preenche o array de historico com o token vazio
  unsigned hist[1] = { Vocab_None };

  // Fornece um index para esta palavra
  index = getIndexForWord (word);

  // Faz o calculo da probabilidade da palavra
  ans = getWordProb (ngram, index, hist);

  // Se a probabilidade e zero, retorna a constante representativa
```

```
  // log(0).
  if (ans == LogP_Zero)
    return BIGNEG;

  return ans;
}

// Fornece a probabilidade um bigram
float
getBigramProb (Ngram * ngram, const char *ngramstr)
{
  const char *words[2];
  unsigned indices[2];
  unsigned numparsed;
  char *scp;
  float ans;

  // Cria uma copia da string de entrada por seguranca
  scp = strdupa (ngramstr);

  // Faz o parsing de bigramas em palavras
  numparsed = Vocab::parseWords (scp, (VocabString *) words, 2);
  if (numparsed != 2)
    {
      fprintf (stderr, "Error: Given ngram is not a bigram.\n");
      return -1;
    }

  // Adiciona palavras ao vocabulario
  swig_srilm_vocab->addWords ((VocabString *) words, (VocabIndex *) indices,
                              2);

  // Preenche o array de historico
  unsigned hist[2] = { indices[0], Vocab_None };

  // Compute the bigram probability
  ans = getWordProb (ngram, indices[1], hist);

  // Retorna a representacao de log(0) se necessario
  if (ans == LogP_Zero)
    return BIGNEG;

  return ans;
}

// Fornece a probabilidade de um trigrama
float
getTrigramProb (Ngram * ngram, const char *ngramstr)
```

```
{
  const char *words[6];
  unsigned indices[3];
  unsigned numparsed;
  char *scp;
  float ans;

  // Duplicate
  scp = strdupa (ngramstr);

  numparsed = Vocab::parseWords (scp, (VocabString *) words, 6);
  if (numparsed != 3)
    {
      fprintf (stderr, "Error: Given ngram is not a trigram.\n");
      return 0;
    }

  swig_srilm_vocab->addWords ((VocabString *) words, (VocabIndex *) indices,
                              3);

  unsigned hist[3] = { indices[1], indices[0], Vocab_None };

  ans = getWordProb (ngram, indices[2], hist);

  if (ans == LogP_Zero)
    return BIGNEG;

  return ans;
}

float
getSentenceProb (Ngram * ngram, const char *sentence, unsigned length)
{
  float ans;
  const char *words[15];
  unsigned indices[2];
  unsigned numparsed;
  char *scp;
  TextStats stats;

  // Create a copy of the input string to be safe
  scp = strdupa (sentence);

  // Parse the bigram into the words
  numparsed = Vocab::parseWords (scp, (VocabString *) words, 15);
  if (numparsed != length)
    {
      fprintf (stderr,
```

```
                "Error: Number of words in sentence does not match given length
                    .\n");
      return -1;
    }

  ans = ngram->sentenceProb (words, stats);
  if (ans == LogP_Zero)
    return BIGNEG;

  return ans;
}

unsigned
corpusStats (Ngram * ngram, const char *filename, TextStats & stats)
{
  File corpus (filename, "r");

  if (!corpus)
    {
      fprintf (stderr, "Error:: Could not open file %s\n", filename);
      return 1;
    }
  else
    ngram->pplFile (corpus, stats, 0);
  return 0;
}

float
getCorpusProb (Ngram * ngram, const char *filename)
{
  TextStats stats;
  if (!corpusStats (ngram, filename, stats))
    return stats.prob;
}

float
getCorpusPpl (Ngram * ngram, const char *filename)
{
  TextStats stats;
  float ans;

  if (!corpusStats (ngram, filename, stats))
    {
      int denom =
        stats.numWords - stats.numOOVs - stats.zeroProbs + stats.numSentences;
      if (denom > 0)
        {
          ans = LogPtoPPL (stats.prob / denom);
```

```
        }
      else
        {
          ans = -1.0;
        }
      return ans;
    }
}


// Quantos ngrams tem o modelo?
int
howManyNgrams (Ngram * ngram, unsigned order)
{
  return ngram->numNgrams (order);
}

//Retorna o tamanho de um array de strings
int tam(char* frases){
 int tamanho = 0;
 int i=0;
  while (frases[i])
    {
      if(isblank(frases[i])){tamanho++;}
      i++;
    }
return tamanho+1;
}
int size(char ** input){
int i=0;
int tamanho=-1;
while (input[i])
    {
      tamanho++;
      i++;
    }
return tamanho;
}

//Retorna a melhor frase
char*
bestSentence (Ngram* lm,char** frases)
{
//{"Por um perto","Por um certo","Por um berto"};
  char *melhor_frase = (char *) malloc (sizeof (char) * 255);
  strcpy(melhor_frase,"");
  float melhor_prob = 0.0;
  float *probs_frases = NULL;
```

```
  int i = 0;
  int tamanho = 0;
  int j=0;

  tamanho=size(frases);
  probs_frases = (float *) malloc (sizeof (float) * tamanho);
  i = 0;
  j=0;

  while (frases[i] && j<tamanho)
    {

      probs_frases[i] = getSentenceProb (lm,frases[i],tam(frases[i]));
      i++;
      j++;
    }
  i = 0;
  while (probs_frases[i])
    {
      if (melhor_prob == 0)
        {
          melhor_prob = probs_frases[i];
          melhor_frase = frases[i];
        }
      else
        {
          if (probs_frases[i] > melhor_prob)
            {
              melhor_prob = probs_frases[i];
              melhor_frase = frases[i];
            }
        }
      i++;
    }

return melhor_frase;
}

//Retorna melhor palavra num contexto
char* bestSentenceContext(char* frases[]){
char* melhor_palavra=NULL;
return melhor_palavra;
}
```

Excerto de Código 4.4: srilm.c

No excerto de código 4.5 aparece o código que fizemos num programa de teste que usa o módulo *srilm.pm* anteriormente criado.

```
!/usr/bin/perl -w

# Usa o modulo SRILM que criamos atraves do SWIG.
use srilm ;
#Escolher a ordem que para inicializar o modelo
my $n = srilm::initLM(3);

#Carrega o modelo
srilm::readLM($n, "SAMPLELM");

# Metodo para saber quantos trigrams existem no modelo,por exemplo.. ( 1=
    unigrams, 2=bigrams )
print "Existem " . srilm::howManyNgrams($n, 1) . " unigrams neste Modelo de
    Lingua\n";
print "Existem " . srilm::howManyNgrams($n, 2) . " bigrams neste Modelo de
    Lingua\n";
print "Existem " . srilm::howManyNgrams($n, 3) . " trigrams neste Modelo de
    Lingua\n\n";

# Questionar o modelo de lingua acerca das probabilidades(logaritmicas) dos
    varios ngrams(n=1,2,3...).
# De notar que um modelo de lingua deste tipo a tecnica de "smoothing" backoff,
     por isso
# se um determinado n-gram nao esta presente no modelo de lingua, vai usar um n
    -gram de ordem inferior.

$p1 = srilm::getUnigramProb($n, "Foi");
print "p('Foi') = " . $p1 . "\n";
$p2 = srilm::getBigramProb($n, "Foi pedido");
print "p('Foi pedido') = " . $p2 . "\n";
$p3 = srilm::getTrigramProb($n, "Foi pedido por");
print "p('Foi pedido por') = " . $p3 . "\n\n";

# Questionar o modelo de lingua para obter o log. final para a probabilidade
    para uma sequencia.
# De que isto e diferente da probabilidade de um n-gram dado que
  # 1. Para uma sequencia, o SRILM adiciona <s> e </s>  no seu inicio
  #    e fim respectivamente
  # 2. O log prob de uma probabilidade e a soma de todas as
  # n-gram log prob individuais
$sprob = srilm::getSentenceProb($n,"Foi pedido por um",4);
print "p ('Foi pedido por um') = " . $sprob . "\n\n";

# Questionar o Modelo de Lingua para obter a log prob total para o ficheiro
    chamado 'corpus'
$corpus = 'input.txt';
```

```
$corpus_prob = srilm::getCorpusProb($n, $corpus);
print "Logprob para o ficheiro " . $corpus . " = " . $corpus_prob . "\n";

# Questionar o Modelo de Lingua para obter a perplexidade para o ficheiro '
    corpus'
$corpus_ppl = srilm::getCorpusPpl($n, $corpus);
print "Perplexidade para o ficheiro " . $corpus . " = " . $corpus_ppl . "\n";
#my @array=("Por um certo","Por um perto","Por um berto");

#Tentativa de usar Funcoes C mas aumentava muito a complexidade do codigo
my @array=("Por um perto","Por um certo","Por um berto");
#print"A melhor frase e: ".srilm::bestSentence($n,\@array)."\n";
```

Excerto de Código 4.5: teste.pl

No excerto 4.6 aparece o output do ficheiro de teste que criamos e que usa o módulo *srilm.pm* anteriormente criado. o interface para Perl.

```
root@localhost swig-srilm]# perl teste.pl
SAMPLELM: line 174: warning: non-zero probability for <unk> in closed-
    vocabulary LM
Existem 4779 unigrams neste Modelo de Lingua
Existem 9583 bigrams neste Modelo de Lingua
Existem 1515 trigrams neste Modelo de Lingua

p('Foi') = -99
p('Foi pedido') = -4.55147504806519
p('Foi pedido por') = -2.56245136260986

p ('Foi pedido por um') = -11.0214958190918

Logprob para o ficheiro input.txt = -522.68359375
Perplexidade para o ficheiro input.txt = 14069.625
```

Excerto de Código 4.6: output teste.pl

De referir que este output representa de uma forma muito semelhante o output do comando *ngram* que vem com o SRILM.

4.4 Processo de Desenvolvimento

O projecto iniciou-se com um profundo estudo da arte em questão, com uma componente de leitura enorme. Os próximos passos estiveram relacionados com análise dos recursos que iriam ser necessários ao projecto, tais como:

- Corpora Monolíngue;
- Bases de n-gramas;

Segui-se a investigação na área dos modelos de língua, entrando por último no estudo e uso da linguagem Perl como ferramenta de programação.

4.4.1 Estado da Arte

Quanto aos modelos de língua, existem várias ferramentas, mas a mais usada actualmente é o SRILM Toolkit que usa como default uma base de N-gramas, sendo N=3.

4.4.2 Tecnologias Escolhidas

Foram escolhidas as seguintes tecnologias para a realização deste projecto:

- Perl, como linguagem de programação;
- SRILM, como ferramenta de criação, treino e teste de modelos de língua;
- SWIG, como ferramenta geradora de interfaces, isto é, geradora de extensões C para, por exemplo, Perl.
- recursos disponíveis num servidor online;

4.4.3 Criação de Modelos de Língua

Foram criados vários Modelos de Língua com a ferramenta SRILM, nomeadamente usou-se o corpora CETEMPublico (língua portuguesa), para criar Modelos para a Língua portuguesa e assim testar se o modelo se adequava ou não.

Foram realizados vários testes com os modelos criados de modo a analisar a perplexidade dos corpus em questão. A perplexidade[9] pode ser vista em termos do número médio de palavras que pode seguir uma sequência de palavras. Mede a dificuldade imposta pela Língua no processo de reconhecimento.

Outra medida associada é a entropia[9], pois avalia a incerteza média na determinação de uma palavra gerada pela fonte de informação e cuja fórmula, em termos matemáticos, pode ser definida assim: tendo uma sequência de palavras de comprimento N como W=*w1,w2...,wn* a entropia possui a fórmula $H(W)=-1/N * logP(w1, w2, .., wn)$ onde o logaritmo está associado à base 2. A perplexidade por sua vez toma a seguinte forma: PP= $P(w1, w2, ..., wn)^{\frac{-1}{N}}$ onde P(w1,w2,...,wn) é a probabilidade da sequência. Basicamente a perplexidade representa a base 2 tendo como expoente a entropia. Antes de mais vão ser apresentados um conjunto de testes que foram realizados com vários corpus, todos eles monolíngue, entre eles:

- CETEMPublico;
- CETENFolha[13];
- EN.out[14], relativo ao Parlamento Europeu (versão inglesa);
- PT.out[14], versão portuguesa.

Criaram-se testes de :

- 1 KB;
- 50 KB;
- 100 KB;
- 5 MB;
- 10 MB;
- 50 MB;
- 100 MB;

Foi usado o comando `ngram-count -text TRAINDATA -lm LM` para gerar (treinar) os vários Modelos de Língua e por seguinte analisar os resultados dos vários testes que foram aplicados aos Modelos usando o comando `ngram -lm LM -debug 2 -ppl TEXTTRAIN` para calcular a perplexidade nos vários testes. De referir que os comandos usados ao fim de vários testes, tanto para treino dos modelos como para testes de perplexidade foram:

- Para o treino: `ngram-count -unkndiscount -interpolate -order 3 -unk -text CETEMPublico1.7 -tolower -lm LM`.

- O método de `smooth` usado acima usa Kneser-Ney *discount*.
- Para o teste de perplexidade: `ngram -lm LM -debug 2 -ppl input.txt`.

De referir que apesar de terem sido usados os modelos enormes que foram criados ao longo do treino, quando se chegou à parte do desenvolvimento se usaram amostras mais pequenas por questões de rapidez e eficiência.

Vamos agora apresentar os vários testes realizados:

- Testes realizados com o corpus EuroParl-PT ao nível da perplexidade e log. probabilidade com repetição de frases pequenas.

  A tabela seguinte demonstra o teste efectuado:

Nº Repetições de Frases Pequenas	EuroParl-PT-PPL	EuroParl-PT-LogProb
0	2060730	-992
1	1752430	-1012
10	579733	-1193
100	37901	-3009
1000	12682	-21160

Tabela 4.1: Perplexidade e Log. Probabilidade com Frases (pequenas) Repetidas

A Figura 3 mostra o gráfico com a evolução da perplexidade mediante o aumento de frases (pequenas).

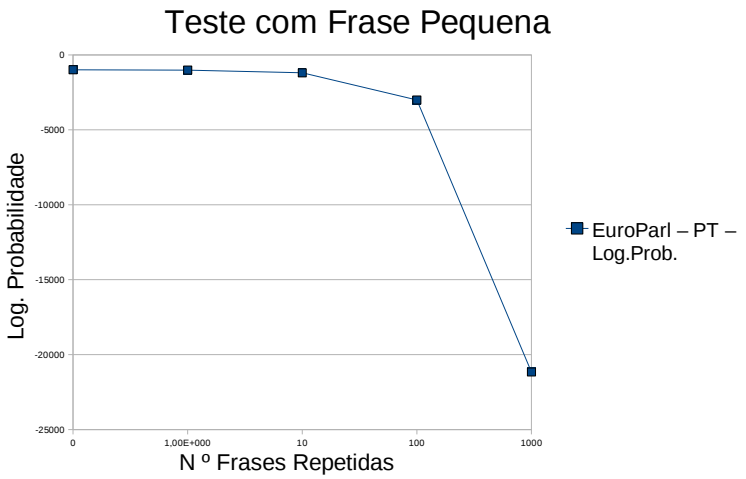

Figura 4.1: Evolução da Perplexidade com Aumento de Frases Pequenas.

- Testes realizados com o corpus EuroParl-PT ao nível da perplexidade e log. probabilidade com repetição de frases enormes.

  A tabela seguinte demonstra o teste efectuado:

Nº Repetições de Frases Enormes	EuroParl-PT-PPL	EuroParl-PT-LogProb
1	2411190	-1149
10	4278400	-2567
100	6512470	-16742
1000	6985700	-158491

Tabela 4.2: Perplexidade e Log. Probabilidade com Frases (enormes) Repetidas

A Figura 4 mostra o gráfico com a evolução da perplexidade mediante o aumento de frases (enormes).

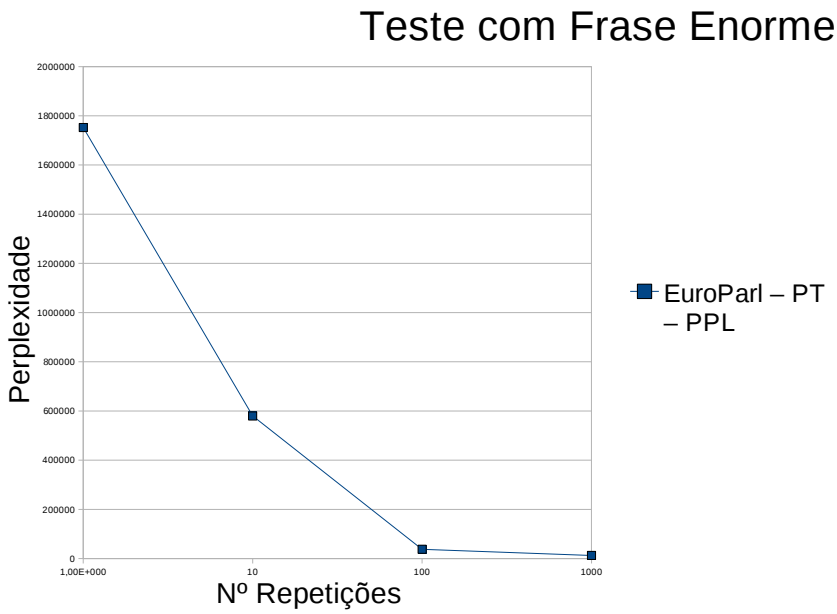

Figura 4.2: Evolução da Perplexidade com o Aumento de Frases Enormes.

A Figura 5 mostra o gráfico com a evolução da perplexidade e do log. probabilidade mediante o aumento de frases (enormes).

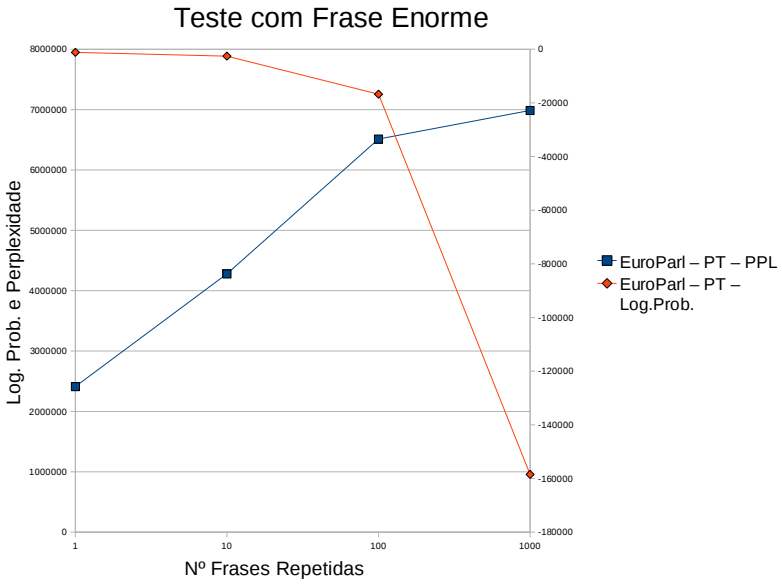

Figura 4.3: Evolução da Perplexidade e do Log. Probabilidade com aumento de Frases Enormes.

- Testes realizados com o corpus um ficheiro de teste ao nível da perplexidade e log. probabilidade com repetição de frases.

  A tabela seguinte demonstra o teste efectuado:

Nº Repetições de Frases Pequenas	Ficheiro Teste-PT-PPL	Ficheiro Teste-LogProb
1	3621970	-604
10	5067820	-1764
100	5927120	-13363
1000	6056530	-129357
10000	6066130	-1289250

Tabela 4.3: Perplexidade e Log. Probabilidade com Frases Repetidas

A Figura 6 mostra o gráfico com a evolução da perplexidade mediante o aumento do dicionário.

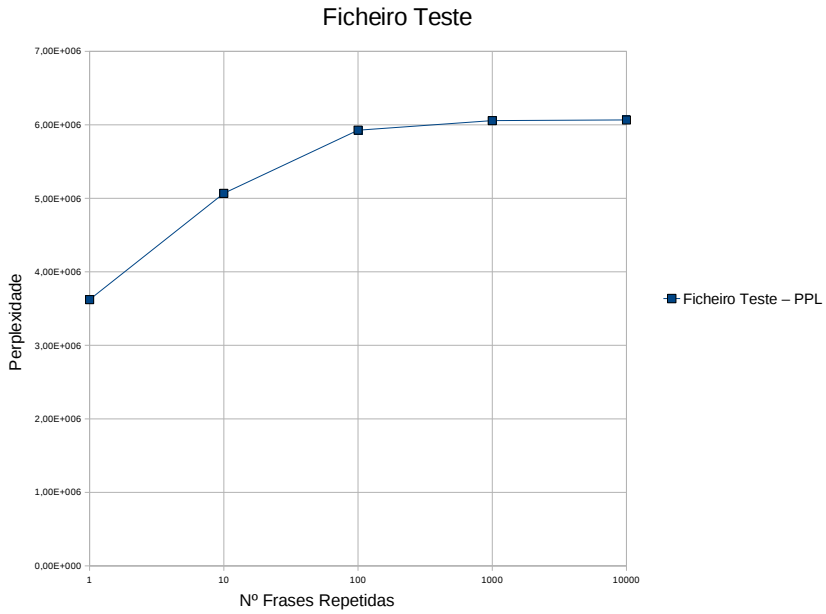

Figura 4.4: Evolução da Perplexidade mediante o Aumento do Dicionário.

A Figura 7 mostra o gráfico com a evolução do log. Probabilidade mediante o aumento do dicionário.

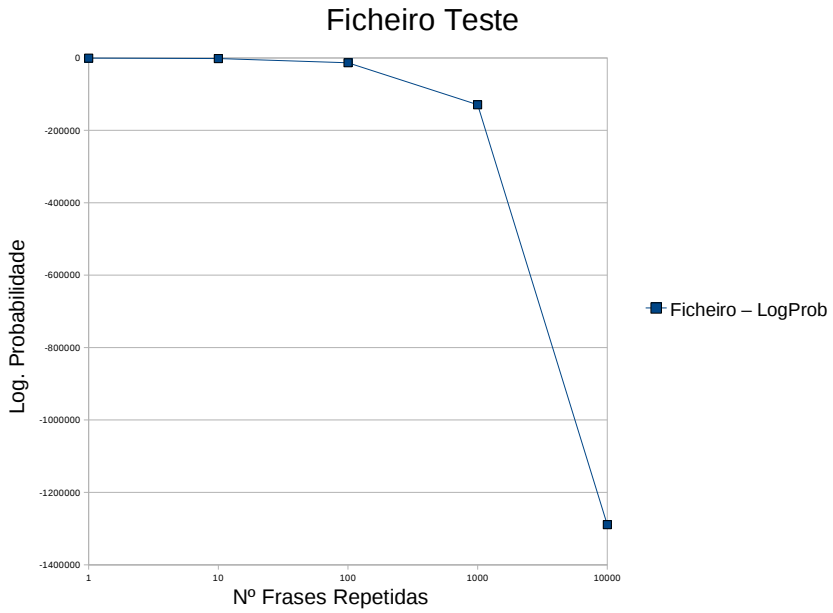

Figura 4.5: Evolução da Log Probabilidade mediante o Aumento do Dicionário.

- Testes realizados com os vários corpus (CETEMPublico, CETENFolha, EuroParl-PT, EuroParl-EN) ao nível da perplexidade com o aumento do número de palavras.

  A tabela seguinte demonstra o teste efectuado para os corpus EuroParl-PT e CETENFolha:

Nº de Palavras	EuroParl-PT	CETENFolha
1KB	2252290	978136
50KB	5501630	654917
100KB	7008480	687492
5MB	6200000	727471
50MB	6059070	690483
100MB	6326820	624135
200MB	6326820	560000

Tabela 4.4: Perplexidade com o aumento do número de palavras (EuroParl-PT e CETENFolha)

A tabela seguinte demonstra o teste efectuado para os corpus EuroParl-EN e CETEMPublico:

Nº de Palavras	EuroParl-EN	CETEMPublico
1KB	7094820	38
50KB	14010200	42
100KB	14016600	42
5MB	14404500	42
50MB	13776400	44
100MB	13633000	36
200MB	13633000	27

Tabela 4.5: Perplexidade com o aumento do número de palavras (EuroParl-EN e CETEMPublico)

A Figura 8 mostra o gráfico com a evolução da perplexida mediante o aumento do número de palavras comparando os vários corpus.

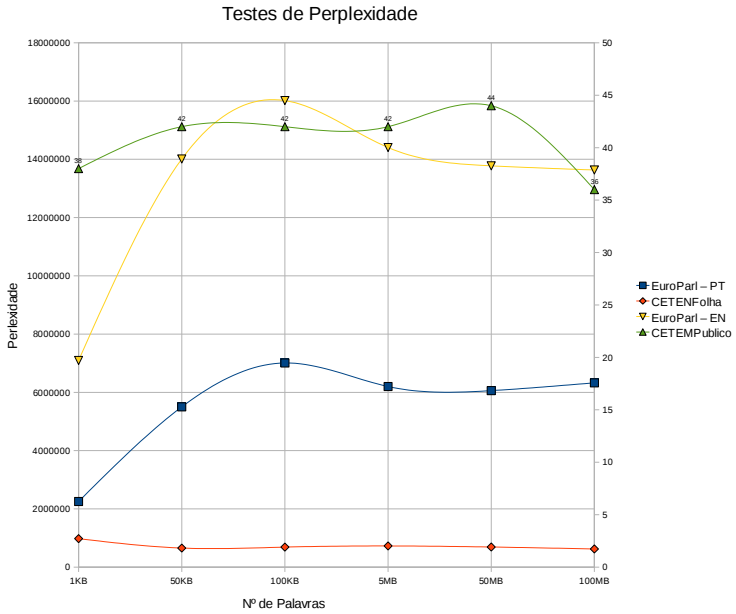

Figura 4.6: Evolução da Perplexidade dos Vários Corpus

Registam-se as seguintes conclusões a tirar destes testes:

- Como esperado, a perplexidade tende a diminuir à medida que o número de frases utilizadas no treino aumenta. Isto deve-se ao facto de, estatisticamente, os modelos que vão sendo treinados melhorarem conforme mais ocorrências de bigramas e trigramas são registradas no dicionário.
- Obviamente, é obvio que conforme vai aumentando o dicionário a perplexidade vai crescendo. Uma possível razão para a perplexidade não sofrer muitas alterações da metade da curva em diante deve-se à maneira como o dicionário foi crescendo. No inicio, palavras que possuem grande frequência na base de textos são adicionadas ao dicionário e competem dentro do modelo, com altas probabilidades. Na fase seguinte, o dicionário é aumentado apenas com palavras pouco frequentes e cujas probabilidades pouco peso têm nas decisões do modelo.
- De notar que a perplexidade do corpus CETEMPublico (usado como corpus de teste) é a melhor (neste caso a menor), dado que o Modelo de Língua foi treinado com este corpus.
- Quanto menor for a perplexidade de um corpus melhor esse corpus se aproxima da língua em questão.

4.5 Módulos Desenvolvidos

Da leitura de toda a bibliografia que foi consultada e dados os requisitos que são pedidos no projecto, deve ser realçado o facto de o sistema desenvolvido, a par de outros já existentes, ir permitir diferentes abordagens no uso de Modelos de Língua.

4.5.1 Lingua::srilm

Vamos apresentar a seguir o módulo Lingua::srilm, gerado através do SWIG. No excerto de código 4.7 aparece o código do módulo *srilm.pm* anteriormente criado.

```
# This file was automatically generated by SWIG (http://www.swig.org).
# Version 1.3.40
#
# Do not make changes to this file unless you know what you are doing--modify
# the SWIG interface file instead.

package srilm;
use base qw(Exporter);
use base qw(DynaLoader);
package srilmc;
bootstrap srilm;
package srilm;
@EXPORT = qw();

# ---------- BASE METHODS -------------

package srilm;

sub TIEHASH {
    my ($classname,$obj) = @_;
    return bless $obj, $classname;
}

sub CLEAR { }

sub FIRSTKEY { }

sub NEXTKEY { }

sub FETCH {
    my ($self,$field) = @_;
    my $member_func = "swig_${field}_get";
    $self->$member_func();
}

sub STORE {
    my ($self,$field,$newval) = @_;
```

```
    my $member_func = "swig_${field}_set";
    $self->$member_func($newval);
}

sub this {
    my $ptr = shift;
    return tied(%$ptr);
}


# ------- FUNCTION WRAPPERS --------

package srilm;

*initLM = *srilmc::initLM;
*deleteLM = *srilmc::deleteLM;
*getIndexForWord = *srilmc::getIndexForWord;
*getWordForIndex = *srilmc::getWordForIndex;
*readLM = *srilmc::readLM;
*getWordProb = *srilmc::getWordProb;
*getUnigramProb = *srilmc::getUnigramProb;
*getBigramProb = *srilmc::getBigramProb;
*getTrigramProb = *srilmc::getTrigramProb;
*getSentenceProb = *srilmc::getSentenceProb;
*corpusStats = *srilmc::corpusStats;
*getCorpusProb = *srilmc::getCorpusProb;
*getCorpusPpl = *srilmc::getCorpusPpl;
*howManyNgrams = *srilmc::howManyNgrams;
*getProbServer = *srilmc::getProbServer;
*bestSentence = *srilmc::bestSentence;
*bestSentenceContext = *srilmc::bestSentenceContext;

# ------- VARIABLE STUBS --------

package srilm;

1;
```

Excerto de Código 4.7: srilm.pm

Para terminar, é importante referir que para este módulo ser usado é necessário copiar (também) para directoria de trabalho, o ficheiro objecto *srilm.so*.

4.6 Arquitectura Cliente-Servidor

Actualmente, uma das tendências nas enormes aplicações de Processamento de Língua Natural diz respeito ao paralelismo no que toca a execução das mesmas. Se cada aplicação, um cliente, carregar o Modelo de Língua para memória cada vez que quer executar as suas operações, torna-se claro que isto é um desperdício e torna todo o sistema em si lento. Em vez da abordagem acima descrita, é usada uma arquitectura Cliente-Servidor, onde o servidor carrega o modelo apenas uma vez e permite aos clientes ligarem-se e obterem todas as respostas que quiserem. Sendo assim, se possível, será útil ter uma máquina com o Modelo de Língua carregado online e permitir o acesso a várias aplicações que desejem questionar o Modelo de Língua. Se houver problemas ao nível da latência da rede (atrasos na rede), deve ser útil pensar em introduzir uma cópia do Modelo de Língua a executar em cada cliente. No entanto, ao nível das universidades e politécnicos a latência da rede é aceitável logo podemos obter ganhos substanciais ao usar a arquitectura Cliente-Servidor.

Nesta fase do documento, vamos ver como criar um servidor de Modelos de Língua através de um Modelo de Língua criado pelo SRILM usando uma arquitectura Cliente-Servidor[15].

4.6.1 XML-RPC com Perl

O protocolo XML-RPC[16], eXtended Markup Language-Remote Procedure Call, permite a clientes remotos executar procedimentos definidos num servidor. Este protocolo usa XML para codificar os pedidos ao servidor e HTTP (HyperText Transfer Protocol) para transportar esses pedidos. Apesar de haver outros protocolos bastante evoluidos, como SOAP, XML-RPC continua a ser uma opção agradável porque foi desenvolvida para ser o mais simples possivel, permitindo ao mesmo tempo a transmissão de estruturas de dados complexas, processamento e respectivo retorno.

Na versão 5 do Perl é possivel usar o módulo Frontier::Daemon, obtido claro pelo CPAN[17], como processo servidor escrito inteiramente em Perl, que disponibiliza os seus métodos. É importante referir que o uso de uma arquitectura Cliente-Servidor neste projecto aumento muito a sua performance[1].

4.6.2 Servidor de Modelos de Língua

Este servidor, além das suas funcionalidades, usa as funções do módulo *srilm.pm* criado pelo SWIG, para disponibilizar as suas próprias funções.

[1] O Processamento passou de 20 minutos para 2 minutos em alguns casos, num computador com as especificações técnicas já referidas.

No excerto de código 4.8 aparece uma parte do código do servidor de Modelos de Língua.

```
#!/usr/bin/perl -w

use strict;
#use warnings;
use Frontier::Daemon;
# Usa o modulo SRILM que criamos atraves do SWIG.
use Lingua::srilm;
# O endereco do servidor.Este endereco diz ao cliente onde se ligar.
my $address = "localhost";
# Numero da porta onde se ligar no servidor.Primeiro e preciso verificar quais
    a portas que estao livres.
my $port = 8080;
# O modelo de lingua que queremos usar.
my $lmfilename = "SAMPLELM";
# A ordem do modelo de lingua.
my $lmorder = 3;
#Escolher a ordem que para inicializar o modelo
my $lm=srilm::initLM($lmorder);
#Carrega o modelo
srilm::readLM($lm,$lmfilename);
# Metodo para saber quantos trigrams existem no modelo,por exemplo.. ( 1=
    unigrams, 2=bigrams )
sub howManyNgrams{
 my $num=shift;
 return {'ngrams'=>srilm::howManyNgrams($lm,$num)};
}
.
.
.

# Ligar ao servidor http://localhost:8080/RPC2
my $methods = {'sample.howManyNgrams' => \&howManyNgrams,
               'sample.getUnigramProb'=> \&getUnigramProb,
               'sample.getBigramProb'=> \&getBigramProb,
               'sample.getTrigramProb'=> \&getTrigramProb,
               'sample.getSentenceProb'=> \&getSentenceProb,
               'sample.getCorpusProb'=> \&getCorpusProb,
               'sample.getCorpusPpl'=> \&getCorpusPpl,
               'sample.bestSentence'=> \&bestSentence,
               'sample.bestSentenceCtx'=> \&bestSentenceCtx,
               'sample.deleteLM'=> \&deleteLM
              };
 Frontier::Daemon->new(LocalPort => $port, methods => $methods)
     or die "Couldn't start HTTP server: $!";
```

Excerto de Código 4.8: servidor.pl

No excerto de código 4.9 aparece uma parte do código do cliente que vai questionar o servidor de Modelos de Língua.

```
#!/usr/bin/perl -w
use NewModule;
use Frontier::Client;
use strict;
#use warnings;
use Acme::Comment type => 'C++';
# Cria um objecto para representar o servidor XML-RPC .
my  $server_url = 'http://localhost:8080/RPC2';
my  $server = Frontier::Client->new(url => $server_url);
my  $corpus= "input.txt";

# Limpa o ecra para melhor observar o output

my $clear=`clear`;
 print $clear;

# Fazer pedidos ao servidor e obter os resultados.
 my $order =1;
# Quantos n-grams de diferentes ordens existem ?
 my $result1 = $server->call('sample.howManyNgrams',$order);
 my $ngrams = $result1->{'ngrams'};
 print "\nExistem " .$ngrams. " unigrams neste Modelo de Lingua.\n\n";
 $order++;
```

Excerto de Código 4.9: cliente.pl

É importante referir, que por questões de facilidade de escrita de código as funções `bestSentence` e `bestSentenceCtx` foram criadas no código Perl do servidor.

Na listagem 4.10 aparece o *output* do cliente depois de questionar o servidor, que se assemelha bastante ao *output* do comando *ngram* do SRILM.

```
[root@localhost swig-srilm]# perl servidor.pl &
[2] 16589
[root@localhost swig-srilm]# SAMPLELM: line 174: warning: non-zero probability
    for <unk> in closed-vocabulary LM

[root@localhost swig-srilm]# perl cliente.pl

Existem 4779 unigrams neste Modelo de Lingua.

Existem 9583 bigrams neste Modelo de Lingua.

Existem 1515 trigrams neste Modelo de Lingua.

p('Foi') = -99

p('Foi pedido') = -4.55147504806519

p('Foi pedido por') = -2.56245136260986

p ('Foi pedido por um') = -11.1236295700073

Logprob para o ficheiro input.txt = -522.68359375

Perplexidade para o ficheiro input.txt = 14069.625


Qual a melhor frase?
Por um certo
Por um perto
Por um berto
A melhor frase e: Por um certo


Qual a melhor palavra mediante o contexto?
Era uma vez numa gruta
Era uma fez numa gruta
Era uma nez numa gruta
A melhor palavra e: vez
```

Excerto de Código 4.10: output cliente-servidor

4.7 Resultados

Penso que depois de terem sido criados os Modelos de Língua, associados a uma forte componente de investigação, e as várias componentes necessárias, estão criadas as condições para que o projecto seja levado a bom porto.
Ficam as seguintes conclusões:

- É importante referir que apesar de muitas funções terem a sua assinatura, a sua implementação deve ser melhorada e em alguns casos, dado tratar-se dum projecto protótipo, deve ser efectuada. Isto terá como resultado um aumento da complexidade do código inerente.
- O módulo Lingua::srilm desenvolvido está pronto a funcionar e irá ser colocado online para posterior estudo.
- A arquitectura Cliente-Servidor veio aumentar a performance do projecto em si.
- O módulo Lingua::srilm futuramente poderá ser usado de uma forma modular.

5

Conclusões e Trabalho Futuro

Neste capítulo, as conclusões deste trabalho são apresentadas, assim como as sugestãos para um trabalho futuro.

5.1 Conclusões

- Embora o SRILM não trabalhe com classes de palavras é, neste momento, uma ferramenta muito utilizada para criação de Modelos de Língua.

- Embora pareça algo normal de assimilar, os Modelos de Língua produzidos pelo SRILM não determinam se uma sequência está correcta ou não, apenas atribuem uma probabilidade à sequência em questão.

- Embora actualmente o estado da arte dos Modelos de Língua aconselhe o uso de base de dados n-gramas de ordem 5, foram usadas bases de dados de ordem 3 (que representa o default do SRILM).

- Foi escolhida a ferramenta para criar os Modelos de Língua necessários usando corpus ou bases de n-gramas de ordem 3.

- O SRILM permitiu-me usar um corpus de lingua portuguesa (CETEMPublico) de modo a criar um Modelo de Língua para a nossa língua. Todas as ferramentas acima descritas estão disponíveis para uso, embora o SRILM seja a ferramenta que vai futuramente ser usada.

- Dado que o SRILM foi construido em C++ foi necessário usar um gerador de interfaces para criar uma extensão Perl do SRILM. Foi usado o pacote de software SWIG que permitiu que o projecto pudesse ser construido em Perl e assim usar as funcionalidades do SRILM.
- Foi criada uma arquitectura Cliente-Servidor em Perl, baseada em XML-RPC usando os módulos Frontier::Daemon e Frontier::Client disponiveis através do CPAN.
- O SRILM já possui uma implementação Cliente-Servidor. Por isso e apesar de ser provável que a geração da implementação cliente-servidor para Perl através do SWIG fosse mais eficiente, optou-se pela criação da arquitectura usando XML-RPC. O principal motivo foi o aumento da complexidade inerente ao código que seria necessário desenvolver.
- Foi criado o novo módulo Lingua::srilm que vai ser posto online para consulta.
- Na realidade ao usar este projecto, seria bom activar mecanismos de cache tanto no cliente como no servidor.

5.2 Trabalho Futuro

O futuro deste projecto passa pela adição de extensões e permitir o seu uso em aplicações existentes ou que vierem futuramente a ser desenvolvidas.

Bibliografia

[1] Le Zang. Reference to Statistic Language Modeling available at: http://homepages.inf.ed.ac.uk/lzhang10/slm.html.

[2] Alberto Simões. Natools. Disponivel para Download em: http://linguateca.di.uminho.pt/natools/.

[3] Corpus relativo ao jornal publico: Cetempublico. Disponivel para Download em: http://www.linguateca.pt/cetempublico/.

[4] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[5] Joshua Goodman. The state of art in language modeling. In *AMTA*, 2002. Slides of the tutorial available at: http://www.research.microsoft.com/~joshuago/amta-tutorial-v6.ppt.

[6] Simon King et al. *System Documentation Edition*. Centre for Speech Technology, University of Edinburgh, Edinburgh, Scotland, 1.2 edition, January 2003. This is a MANUAL for the tools ngram-test and ngram-build and other features.

[7] Andreas Stolcke. Srilm - an extensible language modeling toolkit. In *Proc. Intl. Conf. on Spoken Language Processing*, volume 2, September 2002. http://www.speech.sri.com/cgi-bin/run-distill?papers/icslp2002-srilm.ps.gz.

[8] Steven Bird, Ewan klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009. http://www.nltk.org/book.

[9] Luis Pessoa. Modelos da Língua para o Português do Brasil Aplicados ao Reconhecimento de Fala Contínua: Modelos Lineares e Modelos Hierárquicos (Parsing). Master's thesis, Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e Computação., Brasil, 1999.

[10] Site relativo ao sistema de tradução moses. Informação Disponivel em: http://statmt.org/moses/.

[11] Reference to The Perl Programming Language available at: http://www.perl.org.

[12] David M. Beazley, David Fletcher, and Dominique Dupont. Perl extension building with swig. In *O'Reilly Perl Conference*, 1998. Paper available at: `http://www.swig.org/papers/Perl98/swigperl.pdf`.

[13] Corpus relativo ao jornal folha de são paulo (brasil): Cetenfolha. Disponivel para Download em: `http://www.linguateca.pt/CETENFolha/`.

[14] Corpora paralelo com versão portuguesa-inglesa do parlamento europeu : Europarl (pt-en). Disponivel para Download em: `http://www.statmt.org/europarl/v5/pt-en.tgz`.

[15] Nitin Madnani. Querying and Serving N-gram Language Models with Python. *The Python Papers*, 4(2), 2009.

[16] Eric Kidd. *XML-RPC HOWTO*, 0.8 edition, April 2001. MANUAL for the XML-RPC available at: `http://xmlrpc-c.sourceforge.net/xmlrpc-howto/xmlrpc-howto.html`.

[17] Reference to the Comprehensive Perl Archive Network available at: `http://www.cpan.org`.