Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Python para Pré-processamento e Extracção de Características a partir de Texto Português

João Diogo Coelho Ferreira

Dissertação no contexto do Mestrado em Engenharia Informática, Especialização em Sistemas Inteligentes orientada pelos Profs. Hugo Gonçalo Oliveira e Ricardo Rodrigues, apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática

Setembro 2019

UNIVERSIDADE D
COIMBRA

Resumo

O processamento de linguagem natural é uma área que tem aplicações muito vastas, desde resumo de texto automatizado à tradução automática e criação de agentes conversacionais. Para a criação destas aplicações é necessário utilizar diversas ferramentas complexas que permitam processar grandes quantidades de informação. Actualmente, a maioria opera principalmente sobre a língua inglesa, e embora possam ser treinadas para a língua portuguesa podem ser amplamente melhoradas, para mais se determinadas nuances da língua forem consideradas. Justifica-se por esse motivo a necessidade de criar ferramentas que operem especificamente sobre a língua portuguesa e assim obter melhores resultados.

O presente trabalho propõe o desenvolvimento de ferramentas que permitam processar texto em português e posteriormente transformá-lo em formatos sobre os quais agentes conversacionais consigam operar.

Para isso, foram estudadas ferramentas de processamento de linguagem natural já existentes, e após este estudo iniciaram-se adaptações de modo a tentar obter melhores resultados para língua portuguesa. Este trabalho continuará com a integração de ferramentas que permitam extrair informação e culminará com uma demonstração da sua aplicação à geração de pares de perguntas e respostas, o que poderá ser útil para aplicações como agentes conversacionais.

Neste trabalho foi desenvolvida e disponibilizada ao público uma nova cadeia de Processamento de Linguagem Natural (PLN) para a língua portuguesa, utilizando a linguagem de programação Python.

Palavras-Chave

Inteligência Artificial, Processamento de Linguagem Natural, Reconhecimento de Entidades Mencionadas, Extracção de Informação, Geração de Questões.

Abstract

Natural Language Processing is an area that has very extensive applications, from automated text summarization to automatic translation and conversational agents' creation. In order to create these applications, it is necessary to use several complex tools that allow to process large amounts of information. Currently, most of these tools operate mainly on the English language and, even though thy can be trained to operate in the Portuguese language, they can be vastly improved, especially if some language nuances are considered. This justifies the need to create tools that can operate specifically on the Portuguese language and thus get better results.

The present work proposes the development of tools that allow text processing in Portuguese and later its' transformation into formats on which conversational agents can operate.

For this, the existing natural language processing tools were studied and, after this study, some adaptations were made in order to try to obtain better results for the Portuguese language. This work continues with the integration of tools that allow you to extract information and culminates with a demonstration of its application to the generation of pairs of questions and answers, which may be useful to applications such as conversational agents.

In this work a new pipeline for the Portuguese language was assembled, using the Python programming language.

Keywords

Artificial Intelligence, Natural Language Processing, Named Entity Recognition, Information Extraction, Question Generation.

Agradecimentos

Gostaria de agradecer aos meus orientadores, aos meus pais e irmã, a toda a minha família, à minha namorada e a todos os meus amigos.

Este trabalho foi financiado pela iniciativa Nacional em Competências Digitais da FCT, Portugal INCoDe 2030, no âmbito do projecto de demonstração "Agente Inteligente para Atendimento no Balcão do Empreendedor" (AIA).

Conteúdo

Acrónimos

AC Agente conversacional. 28

PLN Processamento de Linguagem Natural. iii, ix, xvii, 2–5, 7, 9, 11, 12, 14, 20, 21, 23–27, 29, 30, 37, 44, 63, 64

PoS Part-of-Speech. ix, xv, xvii, 8–10, 12–16, 18, 20, 24–26, 28, 30, 32, 33, 35, 37–39, 43, 45–49, 58, 63

PR Pergunta e Resposta. 22, 27, 28

REM Reconhecimento de Entidades Mencionadas. xv, xvii, 8, 17, 18, 21, 24–27, 30, 35, 36, 44, 45, 48, 49, 52, 53, 58, 63, 64

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Samantha: Acha-me estranha?
Theodore: Um pouco...
Samantha: Porquê?
Theodore: Bem, parece uma pessoa, mas é apenas uma voz no computador.
Samantha: Compreendo que a perspectiva limitada de uma mente não artificial possa achar isso. Vai habituar-se.

Her (2013)

O filme "Her", de Spike Jonze, conta a história de um homem que se apaixona por um sistema operativo baseado em Inteligência Artificial (IA). Embora este programa seja, como indicado pelo próprio, "apenas uma voz no computador", apresenta capacidades muito superiores às dos Chatbots e sistemas de diálogo existentes hoje em dia, nomeadamente através da sua capacidade de demonstrar empatia. Apesar da distância, a realidade deste filme parece aproximar-se a um ritmo nunca antes visto. Temos actualmente disponíveis diversos assistentes virtuais (e.g., Alexa, Siri, Cortana,...) que conseguem, de uma forma moderadamente intuitiva, realizar as tarefas básicas de um assistente humano, ainda que com capacidades limitadas ao domínio do auxílio pessoal básico.

Ainda assim, a distância entre os sistemas que temos ao nosso dispor e os assistentes humanos é facilmente aumentada e evidenciada quando alteramos a língua utilizada, por exemplo, do inglês para o português. Por esse motivo, este trabalho passa pela criação de ferramentas que contribuam para um melhor processamento computacional da Língua Portuguesa. Acreditamos que, ao trabalhar nestes temas, podemos aproximar cada vez mais os sistemas inteligentes do sistema ideal e, no caso específico deste trabalho, contribuir para uma melhoria ao nível do estado da arte do processamento computacional de uma das línguas mais faladas do Mundo [1].

Para além de se descrever a organização deste documento, os parágrafos que se seguem apresentam este trabalho, com base na sua motivação, clarificação dos objectivos a atingir, esboço da abordagem a seguir, e o seu enquadramento num projecto de investigação e enumeração das contribuições esperadas.

Na antiguidade, todo o conhecimento era partilhado de familiares mais velhos para os membros mais novos da família. Isso mudou com a invenção da escrita, e mais recentemente com a disponibilização da informação de forma aberta permitindo a qualquer

[1] https://www.babbel.com/en/magazine/the-10-most-spoken-languages-in-the-world/

pessoa ter acesso à mesma. De facto, existe tanta informação que é impossível processá-la toda. A maioria da informação existente, por exemplo na World Wide Web (WWW) e noutros tipos de documentação, não se encontra disponível num formato estruturado e fácil de processar, mas sim em texto corrido (Santos, 1992). Assim, de forma ajudar o ser humano com informação e processamento da sua forma escrita, foram criados sistemas e ferramentas informáticas, que conseguem simplificar as informações descritas neste tipo de texto.

A área do Processamento de Linguagem Natural (PLN), onde se insere este trabalho, é um ramo da Inteligência Artificial que tem como objectivo final desenvolver sistemas capazes de lidar com a linguagem humana, neste contexto chamada linguagem natural, por oposição às linguagens formais (Jurafsky e Martin, 2009).

O desafio do PLN está longe de ser trivial porque, ao contrário do que acontece nas linguagens de programação, as linguagens naturais recorrem a um conjunto de fenómenos que, apesar de as tornarem mais ricas, dificultam o seu processamento automático. Entre esses fenómenos, destaca-se a ambiguidade, presente a vários níveis. Por exemplo, a mesma palavra pode assumir diferentes funções gramaticais ou ter diferentes sentidos (e.g., "banco", instituição financeira, assento, ou forma do verbo bancar). No âmbito dos agentes conversacionais importa mencionar outros fenómenos, tais como a vagueza ("Não sei quem foi o campeão"), variação ("Quem ganhou a corrida?" "Quem venceu?"), anáfora ("Responde-me!" $\rightarrow$ a quê?), elipse ("O Sporting foi campeão? A quem ganhou?"), subjectividade ("Haverá alguém mais bela do que eu?"), ironia ("Fala mais alto que acho que não te ouvem do outro lado da rua") e o eufemismo ("Ele bateu as botas"). Ou seja, será necessário desenvolver uma solução mais avançada que permita de algum modo lidar com todos estes fenómenos e representar a informação transmitida em linguagem natural numa linguagem manipulável pelas máquinas.

Neste trabalho propõem-se três objectivos:

- O objectivo inicial passa pela selecção de um conjunto de ferramentas com vista à sua aplicação em cadeia (i.e., uma *pipeline*) para a extracção de características linguísticas a partir de texto escrito em português. A cadeia deve incluir um conjunto de tarefas de baixo nível (e.g., separação de texto em palavras, identificação da função gramatical das palavras), habitualmente necessárias a várias aplicações no âmbito do PLN, e deverá tirar o máximo partido de ferramentas e recursos já disponíveis. Pretende-se que sejam alterados ou adicionados de raiz os seguintes módulos:
  - Atomizador
  - PoS Tagger
  - Lematizador
  - Reconhecimento de Entidades Mencionadas
  - Extractor de Factos
- O segundo objectivo passa pela disponibilização ao público da nova *pipeline* para que possa ser usada em novas aplicações.
- O terceiro objectivo passa por aplicar a cadeia desenvolvida na geração automática de perguntas a partir de frases em português. Este módulo visa agilizar o processo de criação da base de conhecimento de agentes baseados em listas de perguntas e respostas.

De forma a atingir os objectivos previamente mencionados foi constituído um plano de acção com vários passos.

No primeiro passo tratou-se da análise de diversas cadeias de PLN já disponíveis, tendo em conta o resultado de cada uma (o output que se obtinha com cada cadeia) e a facilidade de alteração da mesma. Apesar de existirem algumas opções a este nível, isto é, ferramentas que permitem o processamento de texto em português, inclusivamente em Python (e.g., NLTK, spaCy), a maior parte é baseada em aprendizagem computacional (em inglês, *Machine Learning*), independentes da língua e treináveis em dados anotados. Isto torna-as flexíveis, mas leva a que acabem por não resolver problemas específicos da língua (Taba, 2013).

Posteriormente, identificaram-se problemas relativos à língua portuguesa e questões específicas que podem ser melhoradas, por exemplo, olhando para o NLPPort. Estes podem passar pelo tratamento de algumas particularidades da língua, tais como o uso de contracções e clíticos, onde os modelos existentes demonstram falhas.

De seguida foram realizadas alterações à cadeia escolhida de forma a resolver estes problemas, sendo depois realizada uma avaliação que comparava a ferramenta com o seu resultado antes de alterações ou com o resultado obtido por outras ferramentas semelhantes.

Depois de realizadas as alterações, foram feitas adições de módulos que não estavam disponíveis de raiz, ou estavam mas implicavam uma implementação separada devido à complexidade de alteração dos mesmos. Mais uma vez, estes módulos foram avaliados utilizando como referência resultados de outras cadeias, ou através de participação em competições.

Tudo isto permitiu o desenvolvimento da nova cadeia de PLN, o NLPyPort, que se baseia na linguagem de programação Python. A escolha desta linguagem prende-se com a crescente popularidade da mesma, que se comprova através da existência de várias bibliotecas e plataformas nela desenvolvidas, com finalidades úteis nos campos da IA e do próprio PLN. Sendo que um objectivo definido era a disponibilização desta cadeia, acreditamos que a opção pelo Python evitará que futuros utilizadores tenham de recorrer a mais do que uma linguagem para desenvolver um sistema nestas áreas. Aliás, desde o início da elaboração desta proposta de dissertação, já houve pessoas a demonstrar interesse na utilização da *pipeline* assim que esta estivesse terminada e actualmente já há pessoas a usarem a mesma.

Como já referido, no contexto deste trabalho, a cadeia é utilizada no fim para prova de conceito de geração automática de perguntas. Ela permite identificar entidades relevantes em cada frase e ainda relações entre elas, constituindo a base para a re-organização do texto no formato de perguntas e para a identificação da respostas. Para além do desafio associado à extracção de informação em formatos simplificados, como triplos sujeito-predicado-objecto, há o desafio de determinar o tipo de perguntas (e.g., "Quem", "O quê", "Onde" ou outros), que se baseia na categoria das entidades envolvidas.

Tendo em conta a complexidade e dimensão desta tarefa, foram escolhidos alguns tipos de relações sobre as quais se geraram modelos de forma a produzir perguntas, reduzindo desta forma o trabalho manual, mas mantendo suficiente para uma demonstração de prova de conceito.

Há ainda a referir que este trabalho está a ser realizado no contexto do projecto de investigação *AIA – Agente Inteligente para Atendimento no Balcão do Empreendedor*, onde se pretende desenvolver mecanismos que permitam interagir através de linguagem natural escrita com o Balcão do Empreendedor (BDE)[2], um portal que constitui um ponto

[2] `https://www.portaldocidadao.pt/balcao-do-empreendedor`

de acesso aos serviços digitais relacionados com o exercício de actividade económica em Portugal. O BDE disponibiliza mais de um milhar de serviços, descritos ao longo de diversas páginas, mas inclui apenas um conjunto com cerca de 200 perguntas e respostas, as chamadas *Frequently Asked Questions* (FAQs).

No contexto do projecto AIA, pares de perguntas e respostas podem ser gerados a partir de textos em documentos e no catálogo de serviços do BDE e servirão para alimentar a base de conhecimento do agente responsável pela resposta automática a perguntas. A nova cadeia poderá ainda ser usada pelos vários módulos em desenvolvimento que necessitem de processar texto e extrair características que levem à sua melhor compreensão, nomeadamente na associação automática de perguntas do utilizador com serviços disponibilizados ou com perguntas de base de conhecimento.

Este trabalho tem como contribuições principais:

- Uma nova cadeia de PLN para o português, desenvolvida em Python, que melhora os resultados de outras cadeias existentes, disponibilizada à comunidade para utilização por outros investigadores em diferentes projectos, já tendo sido usada noutros trabalhos.
- Uma demonstração de extracção de informação para consequente uso num sistema para a geração automática de perguntas e respectivas respostas, a partir de texto corrido em português, que permita, entre outros, alimentar a base de conhecimento de agentes conversacionais, tendo já sido aplicada para participação na competição IBERLEF(Collovini et al., 2019).

Tanto as experiências em torno da nova cadeia de PLN como o desenvolvimento do sistema de extracção de informação foram relatados em dois artigo científicos da área de PLN:

- João Ferreira, Hugo Gonçalo Oliveira e Ricardo Rodrigues (2019a). "Improving NLTK for Processing Portuguese". Em: *Symposium on Languages, Applications and Technologies (SLATE 2019)*. In press
- João Ferreira, Hugo Gonçalo Oliveira e Ricardo Rodrigues (2019b). "NLPyPort: Named Entity Recognition with CRF and Rule-Based Relation Extraction". Em: *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2019), co-located with 35th Conference of the Spanish Society for Natural Language Processing (SEPLN 2019)*. CEUR Workshop Proceedings. Bilbao, Spain: CEUR-WS.org, pp. 468–477

Este documento descreve, através de vários capítulos, a realização deste trabalho, que culminou com o desenvolvimento da cadeia NLPyPort. Depois de uma introdução ao problema e uma clarificação de alguns conceitos fundamentais, são detalhados os diversos elementos que constituem essa cadeia, bem como os problemas associados e soluções propostas para esses problemas. Posteriormente, serão abordados os passos necessários para o desenvolvimento de um sistema para a geração de perguntas. Os diversos subtemas podem ser consultados conforme os capítulos brevemente descritos nos seguintes parágrafos.

No capítulo 2, **Conceitos Fundamentais**, serão explicados conceitos básicos e detalhados os módulos mais comuns numa cadeia de PLN, bem como o resultado típico dos mesmos. Serão ainda abordados conceitos próximos destes que, embora não sejam o foco da presente tese, apresentam igualmente uma parte relevante da mesma: a Extracção de Factos e Geração de Questões.

No capítulo 3, **A cadeia de processamento NLPyPort**, será apresentada a cadeia de PLN montada e os métodos escolhidos para realizar cada uma das tarefas.

No capítulo 4, **Experimentação**, serão apresentados os resultados dos testes práticos realizados para escolher a ferramenta base, e os resultados que se obtiveram para o desempenho de cada um dos módulos alterado, aí comparado com a versão anterior, ou adicionado de raiz tendo depois comparado com o módulo base ou com outras *pipelines*.

No capítulo 5, **Conclusão** será feita uma apreciação global do trabalho, uma reflexão relativa aos objectivos e contribuições esperadas e obtidas e finalmente serão propostos próximos passos a seguir de forma a complementar a cadeia de processamento criada e adicionar aplicações à mesma.

Capítulo 2

Conceitos Fundamentais

Nesta secção serão abordados vários conceitos fundamentais do presente trabalho, sendo explorado desta forma o Processamento de Linguagem Natural (PLN). Analisam-se quais os recursos textuais (*corpus*) existentes e como estes foram anotados, quais os constituintes básicos de PLN, os desafios que estes apresentam e posteriormente as aplicações que estes têm.

2.1 Processamento de Linguagem Natural

O Processamento de Linguagem Natural (PLN, em inglês Natural Language Processing) (Jurafsky e Martin, 2009) é uma área da Inteligência Artificial que tem como objectivo dotar os sistemas da capacidade de comunicar da mesma forma que os humanos, ou seja, usando a mesma língua que os humanos usam para falar e escrever, também chamada de natural, por oposição às linguagens formais.

O PLN é normalmente implementado em cadeia (pipeline). A razão para isto deve-se ao facto das tarefas estarem interligadas, sendo que o resultado (output) de uma será a entrada (input) de outra. É de notar que operações de processamento de maior complexidade necessitam, normalmente, de maior quantidade de informação do que operações mais simples.

Para melhor compreender os diferentes elementos e estrutura da pipeline de PLN, esta irá ser considerada em duas categorias.

A primeira categoria, intitulada de **Tarefas**, inclui programas autónomos ou que dependem directamente do output de tarefas anteriores, que abordam uma tarefa muito específica e de forma isolada e que têm por isso uma utilidade limitada, usada essencialmente ao nível do pré-processamento. Consideraram-se três grandes divisões dentro desta:

- Análise Lexical, onde são abordadas as tarefas de *Atomização* (*Tokenização*) e *Normalização*, especificamente *Lematização* e *Radicalização*.
- Análise de Sequências, onde são abordadas as tarefas de *marcação da função gramatical* (part-of-speech (PoS) *tagging*) e *Reconhecimento de Entidade Mencionadas*.
- Análise Semântica, onde é abordada a tarefa de *Extracção de Informação*.

A segunda categoria considerada, intitulada de **Aplicações**, refere-se a tudo que opere por cima dos módulos da pipeline de forma a permitir estender as suas funcionalidades para fins específicos, tais como a resposta automática a perguntas, tradução automática, sumarização e geração de perguntas. Devido ao teor da presente tese, apenas a geração de perguntas será abordada.

Embora não se enquadre nas categorias anteriores devido ao facto de se tratarem recursos textuais já existentes, considerou-se relevante referir e explorar os dados que são geralmente utilizados para treino e avaliação — conhecidos por *Corpora*, ou em português, corpos, na respectiva subsecção.

O capítulo irá culminar com uma explicação de como todos os elementos referidos anteriormente podem ser utilizados para uma aplicação prática — geração de perguntas e respostas que podem ser usadas por um agente conversacional.

Antes de passar para as seguintes secções, é importante perceber alguns conceitos básicos do processamento de linguagem natural, nomeadamente os seguintes termos:

Átomo — Unidade mais reduzida de texto que transmite ainda informação. Geralmente os átomos são constituídos por palavras.

Atomização — Processo de dividir uma frase em átomos.

Tags Part-of-Speech (PoS) - Conjunto de etiquetas que assinalam a relação que uma palavra tem com as que a envolvem. Geralmente considerada como sendo uma gramática.

Marcação PoS — Processo de marcar os diversos átomos com uma *tag* PoS.

Lema — Forma mais elementar de uma palavra, i.e. a forma como esta apareceria no dicionário.

Lematização — Processo de transformar uma palavra no respectivo lema.

Entidade mencionada — Corresponde a uma entidade que é mencionada num texto, podendo ser de diversas categorias.

Reconhecimento de Entidades Mencionadas (REM) — Processo de encontrar entidades mencionadas e marcar as mesmas com a categoria respectiva.

2.2 Corpora

Tal como em muitos outros sistemas inteligentes, cada módulo de um sistema PLN é desenvolvido com base num conjunto de dados de referência, que pode ainda ser usado para a sua aprendizagem, validação ou avaliação. No caso da linguagem natural, esses dados são frequentemente colecções de textos, também chamados de corpora (*corpus*, singular em latim, e usado em inglês; corpo em português). Um *corpus* é então uma colecção de textos disponíveis num formato electrónico[1]. Estes textos são muito importantes para o estudo da língua posta em prática e para a observação e análise de diversos fenómenos linguísticos, pois são necessários para treinar várias etapas da cadeia de processamento sem que seja necessário criar de raiz os mesmos. Para além do texto, alguns *corpora* podem incluir diversos tipos de anotação, tais como separação em *átomos* e marcação PoS. Entre outras utilizações, um *corpus* manualmente anotado pode ser usado como fonte de dados no treino de sistemas que tenham o objectivo de realizar uma anotação semelhante, mas

[1] https://en.oxforddictionaries.com/explore/what-is-a-corpus/

de forma automática.

Os *corpora* anotados manualmente são geralmente considerados como sendo o padrão de ouro pois garantem os resultados mais correctos, uma vez que terão sido analisados e anotados por profissionais experientes na língua, como linguistas, ou com base em várias opiniões, como acontece no *crowdsourcing*. No entanto, devido ao elevado esforço que esta anotação manual exige, corpora manualmente anotados são escassos. Esta situação é ainda agravada quando se pretende recursos de uma língua especifica menos falada e com uma menor comunidade de investigadores interessados como o Português, o que acontece nesta tese. São também estes que se usam geralmente para avaliar e determinar a qualidade das diferentes etapas de PLN.

Vejamos um exemplo de como poderia ficar uma frase anotada.

Ex: *Vivo num Estado de Ironia*

Resultado:

Átomo	Lema	Marcação PoS
Vivo	viver	v-fin
em	em	prp
um	um	art
Estado	estado	n
de	de	prp
Ironia	ironia	n

2.2.1 Tagsets - classes gramaticais

Para a marcação PoS é ainda necessário definir em que categorias as palavras se podem agrupar. Para resolver este problema foram criados *tagsets*, ou seja, listas de *tags PoS* (categorias gramaticais) a usar na tarefa de PoS e respectivo significado. Existem para o português *tagsets* já definidos, como é o caso do *tagset* usado na anotação da Floresta Sintáctica[2], ilustrado na Figura 2.1.

Como se pode ver, existem por vezes subcategorias dentro das categorias gramaticais definidas. Por exemplo, um dado verbo pode ser apenas classificado como verbo — v —, ou pode ter em conta o seu tempo, e ser classificado como verbo infinitivo — v-inf. Isto acontece porque ocasionalmente é necessário obter mais informações sobre a frase, sendo de notar que em grande parte dos casos a classificação mais geral seria suficiente. Pegando no exemplo anterior, se apenas pretendemos encontrar o verbo da frase para compreender uma relação, não será necessário identificar o verbo como estando no infinitivo.

2.2.2 Formatos

Para uniformizar marcações, são usados formatos específicos que ordenam os elementos obtidos. Este formatos surgem por vezes no âmbito de competições e acabam depois por ser adoptados por uma comunidade mais abrangente de investigadores. Destes destacamos o formato **CoNLL**[3], que representa o texto numa tabela, com uma átomo por linha, e informações acerca de cada átomo nas várias colunas, com a seguinte sequência:

ID, FORM, LEMMA, POS, FEAT, HEAD e DEPREL

O ID corresponde à posição da palavra na frase, começando em 1; FORM corresponde

[2]http://linguateca.dei.uc.pt/Floresta/BibliaFlorestal/anexo1.html
[3]http://ufal.mff.cuni.cz/conll2009-st/task-description.html

Símbolo		Categoria
n		nome,substantivo
prop		nome própio
adj		adjectivo
n-adj		flutuação entre substantivo e nome
v	v-fin	verbo finito
	v-inf	infinitivo
	v-pcp	particípio
	v-ger	gerundio
art		artigo
pron	pron-pers	pronome pessoal
	pron-det	pronome determinativo
	pron-indp	pronome independente(com comportamento semelhante ao nome)
adv		advérbio
num		numeral
prp		preposição
intj		interjeição
conj	conj-s	conjunção subordinativa
	conj-c	conjunção coordenativa

Figura 2.1: Lista de *Tags (tagset)* utilizados para classificar gramaticalmente o *corpus* Floresta Sintáctica.

à palavra, tal como encontrada; LEMMA corresponde à versão lematizada da palavra, ou seja, a forma em que ela aparece no dicionário; PoS corresponde à marcação PoS que lhe foi atribuída; FEAT corresponde a uma série de características morfológicas (separadas por "|"); HEAD corresponde ao índice do parente sintáctico, elemento do qual a palavra faz parte se a frase for divida em blocos sintácticos de forma a compreender que elementos pertencem juntos e sobre a qual elemento uma dada palavra se refere (no caso da raiz será 0 pois não depende de nenhuma outra palavra); e DEPREL corresponde à relação sintáctica entre o HEAD e esta palavra. A seguinte anotação mostra um exemplo disto para a palavra "Elogiei".

```
1, Elogiei, elogiar, v, v-fin, PS|1S|IND, 0, STA
```

Existem ainda versões mais recentes e completas do formato CoNLL, que foram apresentadas por Hajič et al. (2009).

2.2.3 Corpora anotados em português

Um *corpus* que se destaca para a língua portuguesa, nomeadamente para as tarefas de *atomização* e *marcação PoS*, é conhecido por Floresta Sintáctica (Freitas, Rocha e Bick, 2008). A Floresta Sintáctica é constituída por um conjunto de textos em português europeu e português do Brasil e pode ser divida em quatro partes:

- *Floresta Virgem* - Com 1.6 milhões de palavras (95 mil frases) obtidas de jornais em português e português do Brasil, no seu estado natural (não anotada). A ideia desta é anotar as suas frases e passar as mesmas para o *Bosque*.
- *Amazónia* - Com 3.8 milhões de palavras (cerca de 194 mil frases) obtidas de blogs e textos de "não-ficção", exclusivamente em português do Brasil, esta é também anotada de forma automática.
- *Selva* - 300 mil palavras divididas entre diferentes modalidades - escrita e falada.

Esta foi revista de forma parcial (supervisionada) em que após a revisão não é feita frase a frase mas sim caso a caso (isto é, analisado por exemplo se existe classificação correta de sintagmas)

- *Bosque* - Com 190 mil palavras (9.368 frases) é a mais pequena subsecção da Floresta Sintáctica e é constituída por texto jornalístico em português europeu e português do Brasil. Este é o único que, por meio de múltiplas revisões, foi completamente revisto manualmente por linguistas.

Devido ao número reduzido de recursos alternativos, a Floresta Sintáctica é comummente utilizada para treinar e testar diversos módulos de PLN para a língua portuguesa, tendo sido usado em ferramentas como o NLTK e o NLPPort. Existem outros corpos alternativos a este como é o caso do **Mac-Morpho**[4], um corpus constituído por textos em português do Brasil e anotado manualmente, contendo um total de um milhão de palavras (Fonseca e Rosa, 2013).

Um segundo *corpus* que merece destaque para a tarefa de **Reconhecimento de Entidades Mencionadas** é o corpo criado por Freitas et al. (2010a), o Segundo HAREM, construído para a segunda edição de uma campanha para a melhoria de reconhecimento de entidades na língua portuguesa. Nesta, é proposto um total de 10 categorias de entidades distintas:

- PESSOA
- TEMPO
- OBRA
- ORGANIZAÇÃO
- ABSTRACÇÃO
- LOCAL
- ACONTECIMENTO
- VALOR
- COISA
- OUTRO

A distribuição destas categorias pode ser vista na figura 2.2, retirada de Freitas et al. (2010b). Para além destas categorias existem ainda os tipos, segmentos das categorias. Por exemplo, dentro da categoria "Local" existem os seguintes tipos:

- FÍSICO
- HUMANO
- VIRTUAL
- OUTRO

[4]http://nilc.icmc.usp.br/macmorpho/

Estas não foram consideradas nesta tese pois a sua especificidade torna o problema de identificação de entidades mencionadas ainda mais complexo e considerou-se que o valor acrescentado não compensava a perda de performance. É ainda relevante referir que a notação utilizada foi a de **BIO**, que é uma forma de identificar entidades marcando-as com "B" (Beginning, ou em português início), "I" (Inside, ou em português dentro) e "O" (Outside, ou em português fora). Nos casos em que a entidade é do categoria "B" ou "I", esta é ainda seguida da categoria de entidade.

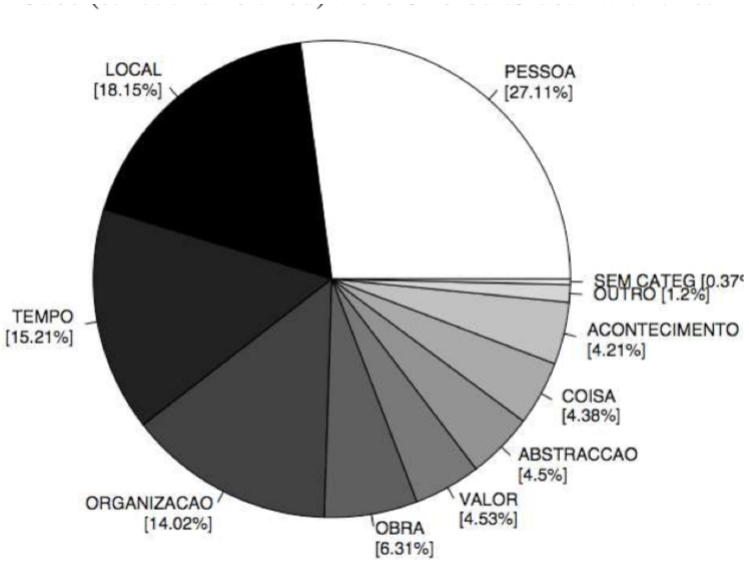


Figura 2.2: Distribuição de categorias na colecção de ouro do Segundo HAREM (Freitas et al., 2010b)

2.3 Tarefas

Serão explorados os diferentes tipos de **tarefas** de uma pipeline de PLN, que são comummente constituídas pelos elementos de segmentação de frases e atomização (em inglês, *tokenization*), anotação morfossintática (*part-of-speech tagging*, abreviado para *PoS tagging*), *normalização*, e *reconhecimento de entidades mencionadas* (Rodrigues, Gonçalo Oliveira e Gomes, 2018) .

Uma tarefa que merece menção mas que devido a ser um problema praticamente resolvido apenas será levemente abordado é a **segmentação de frases** que, como o nome indica, tem como objectivo ler um texto ou documento de entrada de informação e segmentar o mesmo em frases. Embora este processo seja relativamente simples — podemos facilmente supor que o funcionamento se baseia em encontrar "." e dividir a frase nesse local — é necessário que exista um módulo para a sua segmentação devido a excepções existentes. Um exemplo disto é a utilização de abreviaturas como "Dr." ou "Sra." que tendem a ser marcadas com um ponto, e que não correspondem ao fim de uma frase. Uma forma de filtrar estas instâncias é a utilização de listas onde foram previamente colocadas estas abreviaturas, que podem depois ser consultadas durante a segmentação. Um simples exemplo deste processo pode ser visto com as seguintes frases:

Ex: *Um revivalismo refrescante. Tudo, claro está, muito arborizado.*

Resultado: *Um revivalismo refrescante.| Tudo, claro está, muito arborizado.*

2.3.1 Análise Lexical

A análise lexical é constituida pela Atomizaçao e Normalização. Estes consistem em processar informação considerando apenas uma palavra individual e trabalhando sobre a mesma, não sendo por isso necessária muita informação sobre o contexto em que esta se insere.

Atomização

A **atomização** (em inglês, *tokenization*) é um processo muito importante de processamento de linguagem natural e trata-se mesmo de um processo crítico, devido ao impacto que pode ter na tarefas que a seguem. Isto acontece uma vez que um *átomo* (ou átomo) é geralmente o elemento mais pequeno que transmite ainda informação — uma só letra, quando retirada de uma palavra não transmite por si só o sentido da palavra).

Nas restantes fases do processamento é comum usar o *átomo* como ponto de partida e acrescentar novas informações ao mesmo. Existem alguns desafios na *atomização*, dos quais podemos destacar a existência de termos multi-palavra (e.g., gira-discos ou Nova Iorque), o reconhecimento de contracções — instâncias em que verbos e palavras são abreviadas e concatenadas (e.g., "daquele" em vez de "de aquele") e "clíticos — pronomes pessoais que têm em conta a entidade ou entidades que participam na acção ("viu-me" em vez de "viu a mim" , de forma a facilitar a análise, apesar de não ser português correcto). É ainda de notar que alguns deste problemas são significativos dependendo da língua, isto é, os exemplos dados podem ser críticos em português mas podem não se aplicar ao inglês. Por todos este motivos, é de extrema importância garantir que esta fase gera o resultado mais adequado possível, tendo em conta a língua alvo e o objectivo final.

Ex: *Um revivalismo refrescante.*

Resultado: *Um | revivalismo | refrescante|* .

Normalização

O processo de **Normalização** consiste em normalizar as palavras de um texto, isto é, em tentar tornar o texto mais uniforme de modo a facilitar a associação de palavras que, apesar de escritas de forma diferente, têm a mesma origem e, muito provavelmente, transmitem um sentido muito próximo. Existem para isto duas tarefas que, embora semelhantes, permitem utilizações diferentes — a **Lematização** (*Lemmatization*) e a **Radicalização** (*Stemming*).

A **Lematização** é a operação responsável por simplificar as palavras à sua forma mais elementar, como aparece num dicionário. Isto é especialmente importante quando a palavra em questão apresenta diversas conjugações, pois esta simplificação torna as duas conjugações na mesma palavra e permite assim utilizar um maior nível de abstracção. Esta etapa usa como base os *átomos* gerados pelo *atomizador*. Para casos em que a mesma forma pode ser uma variação de diferentes lemas (e.g., "feito" substantivo ou forma do verbo "fazer"), será necessário recorrer ainda às *tags* geradas no PoS *tagging* e assim realizar a desambiguação e escolher a forma simplificada correta. Este método de normalização permite a perda de menos informação quando comparado com outros tipos de normalização, uma vez que mantém uma palavra existente, ainda que noutra forma, na frase.

Ex: *Eu estive a estudar para os exames.*

Resultado: *Eu* ***estar*** *a estudar para* ***o exame****.*

Por sua vez, a tarefa de **Radicalização** simplifica a palavra para a sua forma mais base, o seu radical, normalmente sem recorrer a informação como a classe gramatical. Devido a este processo, pode haver perda de informação pois duas palavras diferentes podem ter o mesmo radical.

Ex: *Estive a estudar para os exames mas foram fáceis.*

Resultado: ***Est*** *a* ***estud*** *para* ***o exam*** *mas* ***ser fac****.*

2.3.2 Análise de Sequências

Para as seguintes etapas comummente consideradas numa *pipeline* de PLN, seria insensato realizar uma abordagem sem considerar a natureza do problema. Tanto as tarefas de marcação PoS como as tarefas de Reconhecimento de Entidades mencionadas podem ser encaradas como um problema de marcação de sequências: a palavra actual é influenciada pela palavra que a precede e pela que a procede, e por vezes até por características destas. Mais que isso, as palavras podem por vezes ser influenciadas por múltiplas palavras anteriores e seguintes.

Embora estes problemas sejam normalmente abordados através de análise e previsão sequencial, a um nível mais específico, a natureza de cada um é algo diferente. Por exemplo, as *tag*s usadas e respectiva distribuição são diferentes, o que também leva a que as características mais adequadas a cada problema sejam diferentes. Isto acontece por motivos como a quantidade de informação disponível aquando da previsão, e do tipo de previsão que se pretende realizar.

Iremos por esse motivo abordar separadamente as estratégias mais comuns utilizadas em cada uma das tarefas préviamente identificadas.

PoS Tagging

O **Part-of-Speech Tagging** — PoS Tagging, ou marcação PoS — é a operação responsável por estabelecer a relação que uma palavra tem com as que a precedem e que a sucedem, de forma a estabelecer uma conexão com as restantes palavras da frase. Uma forma mais comum de entender o PoS é olhando para o mesmo como sendo uma gramática: um conjunto de regras que tem como objectivo compreender como a antecedência e procedência de certas palavras afectam outras.

Ex: *Um revivalismo refrescante.*

$\texttt{"Um"}_{art}$
$\texttt{"revivalismo"}_{n}$
$\texttt{"refrescante"}_{adj}$
$\texttt{"."}_{punc}$

Como descrito anteriormente, um *Part-Of-Speech Tagger* pretende analisar as frases de forma a perceber qual a relação que cada palavra apresenta com as palavras adjacentes, e conseguir posteriormente resolver alguns dos problemas de ambiguidade previamente

identificados. O resultado ideal de um módulo *PoS Tagger* é a atribuição da função gramatical correta a cada uma das palavras em contexto (Jurafsky e Martin, 2009).

Para a maioria das arquitecturas usadas para *PoS Taggers* é necessário treino, para que seja possível realizar as previsões necessárias de forma a melhor determinar quais as *tags* a serem atribuídas a cada palavra. Para este treino usam-se corpus anotados. Como tal, a confiança da atribuição das *tags PoS* é limitada pela qualidade da anotação do corpus.

Sendo uma das primeiras etapas a ser realizada, é importante que os melhores resultados possíveis sejam obtidos, uma vez que qualquer erro nesta será propagado e escalado nas próximas etapas. Existem hoje diversos modelos que permitem obter resultados bastante satisfatórios para o inglês. A maioria dos sistemas consegue atingir uma percentagem de 97% de *átomos* correctamente marcados, em textos bem formados. Destes destaca-se o Standford Tagger (Manning et al., 2014), baseado em entropia máxima que atinge resultados de 97,32%. Embora existam alguns programas que atingem resultados ligeiramente melhores — como por exemplo o NLP4J [5], que relata 97,64% — a utilização do Standford Tagger por um número significativo de ferramentas faz com que este seja o favorito para marcação PoS.

Para o português, os resultados são ligeiramente mais baixos, estando entre os 88% e 96%, casos em que já foi realizada alguma adaptação para melhorar os mesmos (Branco e Silva, 2004).

Aborda-se de seguida de forma mais aprofundada algumas das técnicas mais comuns para a marcação PoS.

PoS baseado em regras O problema central da marcação PoS é a escolha da classe para cada palavra. Os primeiros modelos que surgiram capazes de resolver este problema foram modelos baseados em regras.

Estes modelos tinham dois passos. Primeiro, utilizavam um dicionário para identificar todas as possíveis categorias gramaticais a que uma palavra podia pertencer. De seguida utilizavam regras manuais que permitissem desambiguar a palavra, atribuindo assim uma única *tag* a cada palavra (Brill, 1992). Por exemplo, uma regra poderia ser:

```
Mudar a tag "A" para "B" quando a palavra que procede tem a tag "Z".
```

PoS baseado em *Hidden Markov Models (HMM)* Um segundo modelo utilizado para realizar a marcação PoS baseia-se na *indução Bayesiana*. Esta abordagem considera a marcação PoS como um problema de classificação de sequências.

Para obter a *tag* estatisticamente mais correta, determina-se todas as sequências de *tags* que podem ocorrer tendo em conta o actual contexto/frase em questão. O passo seguinte consiste em escolher de todas estas sequências a mais provável, recorrendo-se para isso à escolha do conjunto de *tags* com maior probabilidade estatística.

Este método apresenta um problema: é virtualmente impossível conseguir calcular todas as combinações de *tags* que podem ocorrer, para todas as combinações de palavras existentes (combinações estas que são infinitas). De modo a simplificar isto, foram utilizadas *Hidden Markov Models*(HMM)(Maia et al., 2006).

A utilização de HMM baseia-se na preposição simplificativa de que apenas duas con-

[5] https://github.com/emorynlp/nlp4j

dições necessitam de ser verificadas. Em primeiro lugar, pressupõem que a probabilidade de um *átomo* aparecer depende apenas da própria *tag* e não das palavras nem *tags* envolventes. Em segundo lugar, assume que a *tag* que aparece apenas depende das *tags* das n palavras anteriores - modelo *n-grama*.

Assumir as condições indicadas anteriormente permite uma simplificação significativa do problema de classificação de sequências. Tendo em conta que não é possível determinar com toda a certeza de que uma *tag* para uma palavra está correta, é utilizado um *Hidden Markov Model*, pois este permite tirar conclusões mesmo quando não se tem a certeza dos resultados anteriores. Para isto são calculados dois tipos de probabilidades, a probabilidade de transição e a probabilidade de observação.

Depois de geradas estas probabilidades, recorre-se ao algoritmo de *Viterbi* (Forney, 1973) para determinar com um maior grau de certeza qual a *tag* da próxima palavra.

PoS baseado em Transformações Outro método para treinar um *PoS Tagger* é o treino baseado em transformações. Trata-se de uma abordagem de Aprendizagem Computacional (*Machine Learning*) que mistura as duas abordagens previamente abordadas.

O treino tem como objectivo criar regras que depois podem ser usadas para alterar *tags* que se acha estarem incorrectamente atribuídas. De seguida, classificam-se as palavras com a *tag* mais provável de uma forma cega; isto é, não se observa o contexto da palavra, sendo atribuída a *tag* que tem maior probabilidade de classificar a palavra. Por último, o texto marcado é percorrido e quando se encontra uma regra aplicável, a *tag* é substituída pela nova *tag* que a regra considera mais correta.

Mais uma vez, poderíamos ver uma regra do tipo:

```
Mudar a tag "A" para "B" quando a palavra que procede tem a tag "Z".
```

Esta regra foi vista anteriormente nos PoS baseados em regras; a diferença chave é que nos sistemas baseados em regras esta tinha sido criada manualmente, enquanto que no PoS baseado em transformações estas regras são aprendidas automaticamente utilizando métodos de Aprendizagem Computacional.

PoS utilizando Modelos de Entropia máxima Um método bastante utilizado passa por utilizar modelos de entropia máxima para determinar a palavra. Este tendem a ser bastantes precisos — 96.6 % de precisão total de *tags* de acordo com Ratnaparkhi (1996). A razão para isto é que, ao contrário de um modelo HMM estes não dependem apenas da palavra e da *tag* anterior, mas sim de todo conjunto de características disponíveis, podendo por isso usar informação de todas as palavras anteriores da frase. Este modelo pode não só utilizar características de todas as palavras anteriores, como características das *tags* destas e pode utilizar até características da palavra actual. Este enorme conjunto de características permite que a determinação da *tag* actual seja mais informada e consequentemente mais correta.

Devido à sua popularidade, este método já está implementado em diversos *toolkits* de raiz, sendo normalmente necessário dados (corpora anotados) para treino do modelo para a língua desejada.

Reconhecimento de Entidades Mencionadas

A tarefa de **REM** (em inglês *Named Entity Recogonition - NER*) consiste em detectar, identificar e classificar correctamente as entidades contidas numa frase. Estas entidades são classificadas de acordo com categorias pré-definidas, sendo as mais comuns *Pessoa, Organização, Data, Local e Valor*. Existem alguns desafios associados a estas tarefas, tais como a dificuldade na resolução de sinónimos e resolução de entidades cruzadas (Nadkarni, Ohno-Machado e Chapman, 2011) — por exemplo, conseguir na frase "O Pedro estuda e ele joga", identificar que "ele" refere-se a "Pedro".

O seguinte exemplo mostra o resultado de identificação e classificação correcta de entidades:

Ex: *O Carlos nasceu em Viseu.*

Resultado: $O[Carlos]_{Pessoa}$ *nasceu em* $[Viseu]_{Local}$.

Como descrito anteriormente, o reconhecimento de entidades consiste em identificar entidades relevantes para a compreensão de uma frase. Regra geral, as entidades mencionadas são nomes de coisas, como pessoas, organizações ou locais. Em domínios específicos, pode ser importante considerar outras categorias de entidade. Se o objectivo for obtenção de informação em literatura médica será provavelmente relevante procurar introduzir palavras em classes como "Medicamento" e "Doença". Contudo esta informação não será provavelmente relevante para outros contextos bastando para isso uma identificação mais geral, isto é, quando não há foco num domínio específico, como "Misc". Existem no entanto classes que são quase sempre consideradas pois tendem a transmitir informação relevante, sendo estas geralmente as classes de "Pessoa", "Organização" , "Data", "Local", "Valor".

Abordagens comuns Em 2003, a conferência CoNLL[6] apresentou a tarefa de extracção de entidades de forma independente da linguagem. Para isso, foram consideradas duas linguagens, o inglês e o alemão. A avaliação dos resultados tem em contra três medidas:

$$\text{Precisão (P)} = \frac{(EntidadesEsperadas) \cap (EntidadesObtidas)}{EntidadesObtidas}$$

$$\text{Abrangência (A)} = \frac{(EntidadesEsperadas) \cap (EntidadesObtidas)}{EntidadesEsperadas}$$

$$\text{Medida-F} = 2 * \frac{(P * A)}{(P + A)}$$

Os resultados da medida F1 desta tarefa estiveram entre perto de 60% e perto de 88% para o inglês, e entre 47% e 72% para o alemão[7]. Isto evidência que, embora existam abordagens que tentem reconhecer entidades de forma independente da língua, as nuances existentes na mesma não permitem que os resultados sejam iguais para todas as línguas.

Freitas et al. (2010a) apresenta o Segundo HAREM a segunda edição de uma campanha para a melhoria de reconhecimento de entidades na língua portuguesa. Este difere do primeiro em apenas alguns pontos, nomeadamente na alteração de uma categoria, tendo o corpus do primeiro HAREM sido posteriormente adaptado para se ajustar a estas alterações.

Nesta campanha houve um total de 27 submissões, realizadas por 10 participantes distintos. Os melhores valores resultantes deste foram um Medida-F de 57,11% na classificação de entidades, considerando as categorias e tipos. A média dos valores obtidos para

[6] http://www.conll.org

[7] https://www.clips.uantwerpen.be/conll2003/ner/

a Medida-F foi inferior a 40%. Como se pode observar, o Reconhecimento de Entidades Mencionadas não é um processo fácil, sendo que os melhores resultados podem apenas ser considerados satisfatórios. Mais informações sobre o HAREM podem ser encontradas em https://www.linguateca.pt/LivroSegundoHAREM/.

Existem diversos métodos para o reconhecimento de entidades mencionadas, que iremos agora abordar.

A primeira abordagem explorada para realizar o reconhecimento de entidades consiste na utilização de lista de entidades previamente construídas, recorrendo a recursos já existentes como a Wikipédia (Nothman, Curran e Murphy, 2008) .

No entanto, a abordagem mais utilizada é considerar REM como sendo um problema de marcação de sequência palavra a palavra (Jurafsky e Martin, 2009) baseada em aprendizagem computacional, seja esta supervisionada, semi-supervisionada ou não supervisionada. De forma a permitir a aprendizagem, são fornecidas algumas características tais como a capitalização de letras (se a palavra tem ou não letras maiúsculas), a pontuação , a existência de dígitos, a morfologia e as *tags* PoS que foram revistas na secção anterior (Nadeau e Sekine, 2007).

Recentemente, devido à utilização de redes neuronais com arquitecturas optimizadas para o processamento de sequências, tais como as Bi-LSTM ou Bi-LSTM-CRF, têm sido obtidos melhores resultados na CoNLL 2003 [8].

O **CRF** é um algoritmo que se baseia em Hidden Markov Models e Modelos de Entropia máxima, aplicado de forma semelhante à explicada anteriormente. No entanto, enquanto que os HMM são modelos generativos, o CRF trata-se de um modelo Discriminativo, à semelhança dos modelos de Entropia Máxima — isto é, este modelo tenta aprender de forma funcional P(Y|X), utilizado os dados de treino para determinar os parâmetros para esta função. Para isso, o problema é modelado através de um grafo não direccionado, em que as características são os vértices, e a ligação entre as mesmas são as arestas. A ausência de ligação entre duas arestas significa que estas características são condicionalmente independentes.

O CRF pretende factorizar a probabilidade conjunta dos elementos de Y num produto normalizado de funções potenciais estritamente positivas. O sistema constrói todas as combinações de características possíveis. O passo seguinte é determinar quais as características que são condicionalmente independentes. Na fase inicial de treino, são consideradas todas as combinações possíveis entre características e as classes de resultados. O sistema aprende depois quais as funções (pesos das arestas do grafo) que maximizam as previsões e quais minimizam (Lopes, Teixeira e Gonçalo Oliveira, 2019). Posteriormente, para garantir que é possível factorizar a probabilidade conjunta é necessário assegurar que variáveis independentes não apareçam na mesma função potencial. Uma forma de fazer isto é obrigar a que cada função operacional opere sobre vértices que formem um clique máximo do grafo.

O resultado final é um grafo não direccionado que consegue determinar qual a função que melhor descreve os sistema, sem que o utilizador tenha de se preocupar em considerar quais as características que são significativas, uma vez que todas são consideradas e as que tiverem menor impacto não irão ser consideradas. Adicionalmente, os pesos/parâmetros de cada função são também calculados automaticamente de forma a reflectir as características úteis.

[8] hrefhttps://aclweb.org/aclwiki/CONLL-2003_(State_of_the_art)

2.3.3 Análise Semântica

Uma outra tarefa a considerar na construção de uma *pipeline* é a extracção de relações. Existem vários algoritmos para esta tarefa, mas destacam-se os algoritmos baseados em aprendizagem não-supervisionada, uma vez que estes permitem extracção de dados sem um forte conhecimento prévio da informação e, por esse motivo, adaptam-se melhor a um corpo tão geral como a Web. Resumem-se de seguida algumas das abordagens mais comuns a esta tarefa.

Hearst (1992) apresentou uma abordagem inicial para a extracção de relações de um tipo específico, baseada em **padrões textuais**. Para isso, eram construídas manualmente expressões regulares que descrevessem os diversos formatos que as relações poderiam tomar e depois verificava-se se a expressão em questão encaixava neste molde, caso em que seria considerado que as duas entidades se relacionavam da forma identificada. Este método não necessitava de qualquer tipo de treino, uma vez que apenas era necessário testar se a frase em questão encaixava em algum dos moldes. No entanto, uma vez que os moldes eram construídos manualmente, esta abordagem estaria sempre limitada aos padrões iniciais disponibilizados.

Snow, Jurafsky e Ng (2005) apresentam uma abordagem baseada em **aprendizagem supervisionada** para a mesma tarefa. Para isso, é escolhido um número fixo de relações e entidades, e é treinado um corpo manualmente. Estes dados são usados para treinar um classificador. De seguida, começa-se por encontrar duas entidades e treina-se um classificador binário de filtragem para determinar se estas entidades são relacionadas por algum tipo de relação ou se não existe qualquer relação entre elas, utilizando para isso exemplos positivos e negativos previamente fornecidos. Por último, um classificador que recebe duas entidades, préviamente identificadas como relacionadas, é treinado para identificar qual a relação entre elas. Este método é no entanto difícil de aplicar, quer pelo facto de que a anotação manual é trabalhosa – é necessário anotar manualmente as **relações entre entidades** – quer pelo facto dos modelos não serem capazes de generalizar resultados para tipos diferentes de texto.

Pantel e Pennacchiotti (2006) apresentam um método baseado em aprendizagem **levemente-supervisionada**. A abordagem é semelhante a Hearst (1992), mas tem uma diferença chave : o processo de descoberta de padrões e relações é automatizado. Para isto, parte de uma série de pares de entidades entre as quais se estabelece a relação alvo (tuplos) considerado de confiança e que seja frequente no contexto referido por exemplo, num *corpos*. Pegando depois nas entidades constituintes destes pares, procura-se no texto frases em que ocorrências destas que estejam próximas uma da outra. As frases obtidas são depois generalizadas para obter padrões. De seguida, estes padrões são utilizados para encontrar novos pares de entidades e estes são usados para encontrar novos padrões. Isto permite que os padrões para relações sejam aprendidos de forma automática. Este método apresenta no entanto problemas quando são identificadas relações erradas (entidades que não estão relacionadas mas estão na mesma frase) havendo no entanto abordagens para reduzir estes erros. De forma a medir a confiança das relações extraídas, o sistema tem em conta dois factores: a performance do padrão encontrado nos tuplos já descobertos, e a produtividade do padrão, contabilizada pelo número de ocorrências na colecção de documentos e estes são depois combinados para gerar uma probabilidade que traduz a confiança no padrão encontrado.

Etzioni et al. (2011a) apresentam o **ReVerb**, um extractor de informação baseado em **aprendizagem não supervisionada**. Este método distingue-se por seguir o paradigma **Open Information Extraction**, designação que lhe é atribuído devido ao facto de não

ser necessário identificar os tipos de relação que se pretende encontrar . Para isto, apresenta uma expressão regular que utiliza as *tags* PoS para extrair, em primeiro lugar a relação, extraindo para isso o verbo e nos casos em que faça sentido informação próxima do mesmo, e em segundo lugar, os argumentos dessa relação, procurando para isso os sintagmas nominais mais próximos que não sejam pronomes relativos, que depois se verifica para saber se satisfaz a expressão regular , apresentada na figura 2.3. Esta é a única regra que constitui o sistema.

$$V \mid VP \mid VW^*P$$
V = verb particle? adv?
W = (noun | adj | adv | pron | det)
P = (prep | particle | inf. marker)

Figura 2.3: Expressão regular de *tags* PoS utilizadas pelo ReVerb. (Etzioni et al., 2011a)

Este sistema é o mais adequado para extracção de informação da Web devido à sua capacidade de extrair informação de forma aberta e sem serem necessários mais elementos- (Jurafsky e Martin, 2009).

A principal dúvida que o ReVerb apresenta é a sua capacidade de adaptação a línguas para além do inglês, sendo necessário testar o mesmo e possivelmente fazer alterações para que o mesmo funcione para a língua portuguesa. Para a língua inglesa, é relatado que o algoritmo conseguiu detectar 86% das relações num teste realizado com 500 frases, que foi depois avaliado manualmente por dois elementos de júri e que marcavam a relação extraída como sendo correcta ou incorrecta.

2.4 Aplicações

No contexto de PLN, consideramos como aplicações todas as operações que actuam no resultado da *pipeline* de forma a obter resultados específicos e concretos relativos a um domínio. Podemos pensar em aplicações como a Classificação Automática de Texto ou Resposta Automática a Perguntas.

Para que um agente comunique através de linguagem natural, não basta ser capaz de a processar. É também necessário ser capaz de traduzir os resultados do seu funcionamento para linguagem natural. A Geração de Linguagem Natural (GLN), que anda normalmente lado-a-lado com o PLN, tem precisamente este objectivo (Gatt e Krahmer, 2018). Entre as tarefas associadas, algumas têm como entrada dados que se pretendem traduzir em linguagem natural (*data-to-text*). Destes, destacamos a produção de boletins meteorológicos (Sripada, Reiter e Davy, 2003) e a geração de legendas para imagens (Xu et al., 2015).

2.4.1 Geração Automática de Perguntas

Uma aplicação dentro da GLN é a geração automática de perguntas. Esta trata-se do processo de utilizar informação formatada para gerar a partir da mesma pares de perguntas e respostas. Uma exemplo de utilização destas são os agentes conversasionais, que tentam pesquisar em todas as perguntas que conhecem uma semelhante à pergunta colocada pelo utilizador e, encontrando uma com um grau de proximidade suficiente, dão a resposta correspondente ao utilizador.

Uma dificuldade inerente à geração de perguntas é que depende da informação previ-

amente extraída, e por esse motivo existe uma probabilidade de propagação de erros que cause erros mais graves nesta aplicação.

Na maioria dos casos, a geração de perguntas recorre a vários tipos de informação para poder produzir perguntas. Isto é, não se utiliza apenas a informação fornecida pelo texto, mas também "padrões" ou modelos de perguntas sobre os quais se tenta encaixar a nova informação obtida. Iremos agora rever alguns dos métodos mais comuns de geração de perguntas.

Silveira (2008) propõe a criação de um *framework* generalizado de geração de perguntas em que são utilizados dois elementos como entrada; o texto em si e modelos de perguntas (gerados anteriormente e guardados numa base de dados), sendo que depois o texto é adaptado de forma a encaixar nestes modelos e gerar assim perguntas. Mostra também como os diversos elementos de uma PLN se integram no processo de geração de perguntas.

Kalady, Elikkottil e Das (2010) apresentam uma abordagem para a extracção de perguntas para a língua inglesa, em que geram diversos tipos de perguntas. Para a geração de perguntas de domínio aberto, utilizam a marcação obtida pelo REM e conforme esta geram as perguntas. Por exemplo, se um elemento for classificado como PERSON, geram uma pergunta começada por "Who", se um elemento for classificado como LOCATION a pergunta irá ser iniciada por "Where". Para este tipo de perguntas utilizam depois expressões regulares, que usam o tipo de pergunta como explicado anteriormente e alteram a estrutura da frase de forma a perguntar algo sobre o elemento em falta. A expressão regular muda conforme a categoria das entidades encontrados na frase. Para a geração de perguntas de resposta Sim/Não, após o reconhecimento de entidades, faz-se inversão dos verbos auxiliares (*"Do"* e *"Does"*) para tornar a afirmação numa pergunta.Por exemplo, a frase :

```
"Jonh writes code."
```

Seria transformada em :

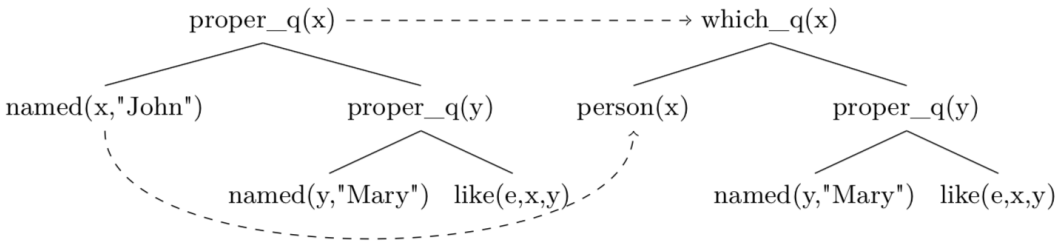
`"What does Jonh write?"`.

Os autores destacam ainda o facto de que é possível gerar múltiplas perguntas da mesma frase, dependendo das entidades seleccionadas para questionar.

Yao e Zhang (2010) apresentam uma alternativa para a geração de perguntas recorrendo a **Minimal Recursion Semantics (MRS)**. Na sua essência, este método consiste em primeiro assinalar entidades e termos, passando estes a ser considerados um *átomo* que nunca é separado (e.g., "Filipa" e "Alexandra" iria ser agregado e apareceria sempre em conjunto "Filipa_Alexandra"). Depois disto passa-se para os módulos de MRS, começando primeiro por dividir frases complexas em frases simples, encontrando para isso o verbo e detectando depois os argumentos à sua volta. O passo seguinte trata-se da transformação MRS para frases simples. Para isso, faz-se um mapeamento das *previsões elementárias — EPS, elementary prediction sentences —* de relações. Observemos a Figura 2.4. A árvore foi transformada identificando uma relação do tipo "which" e uma entidade pessoa, que se considera uma relação "named", uma vez que sabemos que é uma pessoa que tem um nome.

Sabendo que as perguntas do tipo "Who" são constituídas por EPS que contenham relações "which" e "named", podemos concluir que é possível gerar uma pergunta deste tipo para esta frase, uma vez que ambos os elementos estão presentes. Assim, substituindo directamente a pessoa por esta relação obtemos a questão "Who likes Mary?".

Diéguez, Rodrigues e Gomes (2011) apresentam uma abordagem para geração de PRs



(a) "John likes Mary" → "Who likes Mary?"

Figura 2.4: Exemplo de geração de perguntas usando MRS (retirado de Yao e Zhang (2010))

na língua portuguesa que utiliza em raciocínio baseado em casos — **Case-Based Reasoning**. Este método consiste em comparar nova informação com um conjunto de casos previamente acrescentados para determinar se existe algum caso próximo deste, de forma a determinar se a informação recebida pode ser utilizada pelo módulo de geração de perguntas, e caso exista, os casos anteriores são reutilizados nesta situação. Este sistema apresenta ainda um factor de aprendizagem, pois quando se detectam estas situações elas são adicionadas aos casos préviamente anotados e passam a ficar disponíveis para o futuro. De seguida a informação é passada para o módulo de **geração de perguntas**, que usa um modelo para gerar o formato das perguntas. Posteriormente utiliza-se um módulo que determina um padrão de pesquisa – uma combinação de *tags*, lemmas e palavras. Quando um padrão é detectado, é utilizado um conjunto de operações do mesmo para gerar a pergunta. O resultado final é um conjunto de pares Pergunta e Resposta (PR). Por exemplo, assumindo que a regra encontrada é :

```
ser obj
```

Será criada um modelo semelhante a:

```
Como $verbo $objecto?
```

Em que se traduz $verbo no verbo da frase, $objecto no sujeito da frase e "Como" é escolhido como pronome interrogativo.

Aplicando a uma frase como:

```
A mesa é grande.
```

"A mesa" seria o objecto, e "é" (ser) o verbo. Usando o modelo anterior seria obtida a pergunta:

```
Como é a mesa?
```

Este método apresenta mais uma característica interessante, que é o facto de que, embora as regras (isto é, os *templates*) do módulo de geração de perguntas sejam definidas manualmente, os padrões de pesquisa para cada um deles podem ser encontrados automaticamente.

Ali, Chali e Hasan (2010) descrevem um sistema de geração de perguntas a partir

de frases simples. Este artigo apresenta um modelo a considerar, uma vez que todos os recursos necessários para para a geração das perguntas pode ser obtidos utilizando os módulos da *pipeline* de PLN, sendo no entanto evidente pelos resultados do mesmo que há muito espaço para melhorias futuras. O primeiro passo consiste na obtenção de frases simples ou de triplos constituídos pelas duas entidades em questão e a relação entre elas. Para isto recorre-se à *pipeline* tanto para encontrar as entidades como para encontrar a relação (análise semântica). Recebendo como entrada as frases simples ou triplos processadas pelo módulo de PLN, o tipo de frase é inicialmente classificado, sendo o tipo constituído pelas categorias de entidades encontradas. Esta classificação identifica a frase como sendo de categoria relevantes, como é o caso de "Pessoa", quando se trata de uma pessoa (e.g.,"João"), "Organização", quando são entidades e organizações (e.g.,"Universidade de Coimbra"), "Local", quando se tratam de locais (e.g.,"Coimbra") e outros que se possam considerar relevantes.

A categoria de entidades escolhida condiciona o tipo de perguntas geradas, uma vez que a cada categoria de entidade corresponde uma pergunta padrão que pode ser aplicada. Por exemplo, se a entidade for "Pessoa" o tipo de pergunta irá ser perguntas iniciadas por "Quem". De seguida é proposta uma análise para um processamento em classes mais gerais. Li e Roth (2006) usam 6 classes: abreviatura, descrição, entidade, humano, localização ou valor numérico; e posteriormente classes mais detalhadas dentro das várias possíveis para o contexto apresentado anteriormente —os autores consideram 50 classes mais detalhadas. Depois, de forma a decidir qual das classes mais refinadas usar, é analisada a relação entre as palavras da frase, sendo atribuído um tipo de palavra de pergunta (como "Onde" e "Quem") de forma a permitir gerar múltiplas perguntas. Para melhor ilustrar isto, analisemos a seguinte frase:

O João foi a Espanha.

A frase deverá ser identificada como tendo uma entidade da categoria "Pessoa" (João), um verbo (ir) e uma entidade da categoria "Local" (Espanha), podendo depois, tendo em conta a relação entre estes elementos, gerar uma nova frase com uma estrutura semelhante a:

Pessoa Verbo Local.

Isto determina o tipo de regras que poderá depois ser utilizada na construção de perguntas.

O último passo é a geração da pergunta em si. É utilizada uma tabela que, tendo em conta a regra identificada anteriormente, permite a geração de perguntas através de manipulação de frases com o uso de expressões regulares. Se a frase contêm entidades das categorias Humano e Local , e se pretende interrogar sobre o lugar, uma expressão regular para a criação de uma pergunta pode ser:

Onde *<Verbo>* ***o*** *<Humano> ?*

Em que <Verbo> é substituído pelo verbo encontrado anteriormente ("Foi") e <Humano> substituído pelo sujeito anteriormente encontrado ("João"). A resposta neste caso será a entidade que não foi usada neste padrão ("Espanha"). Uma tabela que componha estas regras é um elemento essencial para a geração de perguntas e é um dos elementos em falta para a Língua Portuguesa, sendo por isso um dos elementos que teria de ser criado manualmente caso se opte por esta abordagem.

Zhou et al. (2017) apresentam uma abordagem mais moderna ao problema de geração de perguntas. Para gerar perguntas recorrem a uma **Rede Neuronal** que recebe como entrada tanto as palavras como as suas características, por exemplo tamanho e uso de maiúsculas. Embora não existam ainda muitos estudos feitos em que este método é utilizado, os resultados parecem promissores, sendo que o modelo apresenta uma boa performance tanto na precisão como abrangência para a maioria do tipo de perguntas.

2.4.2 Chatbots e sistemas de diálogo

Uma aplicação comummente considerada quando se fala de PLN são os chatbots. Os chatbots tratam-se de agentes inteligentes capazes de comunicar na língua humana, ou seja, perante uma pergunta em linguagem natural, eles têm a capacidade de a processar e tentar responder-lhe da melhor forma possível. Para isto, têm de recorrer muitas vezes a aplicações como a geração de perguntas, e respectivas respostas, para adicionar dados à sua base de informação num formato que permita melhor mapear estas perguntas com as do utilizador e poder dessa forma retornar a resposta mais adequada, podendo ainda recorrer a outros módulos de geração de linguagem natural para que a interacção com o utilizador seja mais natural.

As aplicações são assim os verdadeiros elementos finais, que embora não pertençam à *pipeline* em si, representam de forma mais natura o resultado final do trabalho realizado por uma *pipeline*.

2.5 Toolkits PLN

Para se poder averiguar como melhor podem ser resolvidos os problemas relacionados com uma *pipeline* de PLN, é necessário explorar quais as *pipelines* que existem neste momento que conseguem operar sobre a língua portuguesa. A maioria destes kits oferece as tarefas de PLN básicas (atomização, análise sintáctica, *PoS Tagging*, REM) para a língua inglesa. Quando se fala da língua portuguesa verificamos que por vezes não existem soluções imediatamente disponíveis, e para resolver isto são geralmente fornecidos módulos que podem ser utilizados para treinar modelos para qualquer língua. Porém, estes módulos não existem para todas as tarefas, havendo por isso elementos em falta (Rodrigues, Gonçalo Oliveira e Gomes, 2018). É ainda de notar que, mesmo quando estas tarefas existem para a língua portuguesa, o desempenho é normalmente inferior que para o inglês.

Após uma análise das *toolkits* existentes e da sua adaptabilidade para a língua portuguesa, destacaram-se as seguintes: **NLTK**, **SpaCy**, **OpenNLP**, **NLPPort** e **CoreNLP**. Para cada um destes iremos numa primeira fase analisar quais os módulos que disponibiliza e, nos módulos considerados que podem ser usados como base do projecto, iremos também analisar a facilidade de uso e facilidade de alteração dos mesmos.

Dos toolkits enumerados apenas o SpaCy e o NLTK utilizam a linguagem Python, sendo que um destes teria necessariamente de ser um ponto de partida; considera-se no entanto importante analisar o NLPPort, que é uma *pipeline* desenvolvida na linguagem de programação Java, pois apresenta resultados que se pretende alcançar em diversos módulos e por ser uma ferramenta desenvolvida dentro do grupo de investigação, que se sabe dispor de diversos recursos que podem ser úteis para a construção de uma nova *pipeline*.

Devido à adesão quase global a redes sociais, existe uma grande abundância de dados que podem ser agregados na forma de um corpus, possivelmente menos estruturados

(Pinto, Gonçalo Oliveira e Oliveira Alves, 2016). Um exemplo disto são os *tweets*. A especificidade deste tipo de textos levou a criação de *toolkits* de PLN especializados para a análise de redes sociais, como é o caso do TwitterNLP (Ritter, Etzioni e Clark, 2012). No entanto, tendo em conta o teor do trabalho, foram excluídos *toolkits* especializados em análise de texto em formatos alternativos.

Foram também considerados, mas excluídos por não serem os mais apropriados, os *toolkits* **LinguaKit**[9] (que foi rejeitado devido à sua linguagem nativa – Perl) e o **CoreNLP** [10] (que foi descartado devido a não possuir módulos para o português de raiz).

2.5.1 OpenNLP

O OpenNLP[11] é um *toolkit* disponibilizado abertamente pela fundação Apache que utiliza a linguagem de programação Java e que dispõe de módulos para segmentação de orações, atomização (com três possibilidades de implementação – *"Whitespace" tokenizer, Simple tokenizer e Learnable tokenizer*), marcação PoS (que utiliza de entropia máxima, havendo também um modelo baseado em perceptrões), REM, lematizador para a língua inglesa (baseado em métodos estatísticos e dicionários) e análise sintáctica.

Embora disponha de um grande número de módulos de uma *pipeline* PLN, estes são maioritariamente direccionados para o inglês, isto é, existem diversos modelos que apresentam resultados diferentes na língua inglesa. No entanto estes resultados não são garantidos para o português, existindo apenas modelos treinados para a atomização, divisão de frases e marcação PoS[12].

2.5.2 NLPPort

O NLPPort (Rodrigues, Gonçalo Oliveira e Gomes, 2018) foi criado sobre o OpenNLP (foram acrescentadas camadas ao OpenNLP) de forma a complementar as suas falhas relativamente à língua portuguesa. Utiliza os modelos do OpenNLP acrescentando os elementos necessários. Por exemplo, acrescenta uma fase de pré-processamento que permite separar "clíticos" antes da *atomização* e *PoS tagging*, fazendo com que os resultados destes sejam mais correctos).

Este *toolkit* é bastante completo, apresentando módulos para segmentação de orações (SenPORT), *atomização* (TokPORT), Reconhecimento de entidades (EntPORT), *PoS tagging* (TagPORT),lematização (LemPORT), analise sintática (ChkPORT), análise de dependências (DepPORT) e extração de factos (FacPort). É ainda de notar que o DepPORT utiliza como base o MaltParser[13], um sistema de *parsing* de dependências, tendo sido depois criado um novo modelo e realizadas as adaptações necessárias.

A existência de todos estes módulos torna o OpenNLP uma solução bastante versátil para diversos problemas. A sua arquitectura permite ainda que componentes criados para este possam ser reutilizados e aproveitados para melhorar *toolkits* que sejam construídos noutras linguagens de programação.

[9] https://linguakit.com/pt/analise-completa
[10] https://stanfordnlp.github.io/CoreNLP/human-languages.html
[11] http://opennlp.apache.org/docs/1.9.1/manual/opennlp.htmlopennlp
[12] http://opennlp.sourceforge.net/models-1.5/
[13] http://www.maltparser.org

2.5.3 SpaCy

O SpaCy[14] é uma ferramenta de PLN aberta desenvolvida na linguagem Python. O funcionamento do SpaCy consiste em criar uma classe base à qual é acrescentada mais informação conforme esta se vai obtendo.

Este oferece uma metodologia de processamento que consiste em criar uma *pipeline* pré-definida que pode depois ser aplicada a qualquer texto (Fig. 2.5).

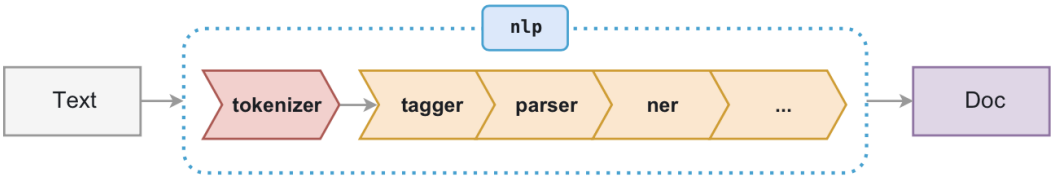


Figura 2.5: Exemplo de uma *pipeline* de processamento utilizada no SpaCy

Para o português existe uma *pipeline* pré-definida que é constituída pelo PoS tagger, o parser de dependências e REM. Para estas tarefas utiliza uma Rede Neural Convolucional (RNC). Não é fornecida na página oficial informação adicional sobre as restantes etapas da *pipeline* nem da sua adaptação especifica para a língua portuguesa. No entanto, de forma semelhante ao OpenNLP, há instruções de como poderiam ser adaptados os diversos elementos para outras línguas para além do inglês.

O SpaCy apresenta uma vantagem em relação a outros *toolkits* devido à sua fácil utilização, sendo por isso bastante intuitivo para utilizadores que não são experientes e estando pronto a utilizar sendo necessário apenas obter o código fonte e escrever um número reduzido de linhas de código para que se obter o processamento de toda a *pipeline*.

Dito isto, podemos considerar que o SpaCy pode apresentar um bom ponto de partida para a montagem de uma nova *pipeline*, em que os pontos fracos deste poderiam ser melhorados para obter resultados finais melhores.

2.5.4 NLTK

Por último temos o NLTK - *Natural Language ToolKit* (Bird e Loper, 2004). Esta é a ferramenta de PLN mais conhecida para Python, e existe por isso um grande número de contribuições realizadas ao longo do tempo de forma a implementar cada vez mais elementos. No entanto, confirma-se mais uma vez que estas contribuições são maioritariamente realizadas para a língua inglesa, sendo muito mais reduzidas para a língua portuguesa. Pode-se por isso verificar que não existem nativamente implementados elementos como o lematizador, mas existem módulos de Atomização, marcação PoS, análise sintáctica e REM.

Ao contrário do SpaCy, o NLTK funciona de uma forma mais complexa, em que cada módulo tem de ser individualmente chamado e tendo cada um de ser alimentados com a informação correta no formato desejado. Embora isto possa inicialmente parecer uma desvantagem, apresenta também uma vantagem; ao ser segmentado à partida é possível

[14] https://spacy.io

fazer adições a um dado módulo ou alterar completamente o mesmo sem ser necessário alterar código base do NLTK. Isto não só facilita que uma nova *pipeline* seja montada a partir do mesmo, mas também permite que sejam realizadas alterações para que a nova *pipeline* possa ser adaptada para uma utilização mais amigável que requer apenas uma chamada, como acontece com o SpaCy.

É ainda necessário notar que apesar de tudo o indicado anteriormente, o NLTK não é um framework necessariamente simples de compreender e alterar interiormente. Como iremos ver posteriormente, nas circunstâncias em que foi preciso realizar algumas alterações que estavam fora do contexto normal no NLTK (como re-treinar o modelo de REM para utilizar um conjunto de categorias diferentes) isto revelou-se uma tarefa extremamente complexa.

Tendo tudo isto em conta, o NLTK é também um potencial ponto de partida para a montagem de uma nova *pipeline*. No entanto, de forma a despistar se este era o melhor para a adaptar de forma a criar uma nova *pipeline* mais testes tiveram de ser realizados, como iremos ver na secção 4, **Experimentação**.

2.5.5 Comparação de toolkits

De forma a melhor compreender as limitações de maior importância, começou-se por analisar quais os elementos que cada uma tem e os resultados foram condensados na tabela 2.1.

Comparação Toolkits	**OpenNLP**	**NLPPort**	**SpaCy**	**NLTK**
Linguagem	Java	Java	Python	Python
Atomização	Sim	Sim	Sim	Sim
PoS Tagging	Sim	Sim	Sim	Sim
Análise Sintática	Sim	Sim	Sim	Sim
Lematizador	Sim	Sim	Não	Não
REM	Sim	Sim	Sim	Sim

Tabela 2.1: Tabela de comparação das tarefas realizadas pelas diferentes *toolkits* PLN testadas.

Como podemos ver, a maioria das ferramentas tem implementados, ainda que não de raiz, para o português, os elementos considerados como essenciais para uma *pipeline* de PLN. Esta comparação inicial apenas observou a capacidade de cada ferramenta em termos do que disponibiliza à partida, e como decomposto nas secções anteriores, foi também analisada a facilidade de adaptação das potenciais ferramentas bem como a sua facilidade de uso.

Tendo isto em conta, podemos considerar que qualquer ferramenta das criadas em Python (SpaCy e NLTK) poderia ser um bom ponto de partida, pois dispõem dos mesmos módulos e por isso é necessário realizar testes mais concretos de forma a perceber qual a ferramenta que melhores resultados apresenta para cada tarefa.

2.6 Visão integradora

Tendo clarificado os diversos elementos que irão ser abordados nesta tese, é agora possível explicar todos os passos para a criação de um conjunto de pares PR que poderão ser usadas

num Agente conversacional (AC) baseado em PR (Figura 2.6).

Partindo dos dados num formato de texto, começamos por utilizar a *pipeline* de pré-processamento, para dividir em frases e depois em *átomos* o texto. De seguida, realizam-se tarefas de processamento mais complexas como o PoS Tagging, para atribuir a cada *átomo* a função gramatical (*tag PoS*) correspondente e, se necessário, a Lematização, para permitir desambiguação de átomos. Com estes dados identificam-se as entidades mencionadas, terminando neste passo a *pipeline* de pré-processamento.

O passo seguinte é a extracção de informação para uma forma estruturada, como triplos. Existem vários métodos para a obtenção das mesmas como foi visto anteriormente, podendo ser utilizado, por exemplo um sistema baseado em extracção de informação através de expressões regulares.

Finalmente, é necessário gerar a pergunta a partir da informação extraída recorrendo para isso a modelos que, tendo em conta o tipo das relações, são utilizados para gerar perguntas em linguagem natural sobre as mesmas.

O resultado é um conjunto de pares PR que permitem questionar sobre a relação entre as duas entidades e caso elas seja relacionadas responder à pergunta sobre a mesma.

Para associar as perguntas feitas pelo utilizador a perguntas da base de conhecimento, algumas das quais geradas automaticamente, utilizam-se métodos como Similaridade Semântica Textual (Fonseca et al., 2016). Este seria o passo seguinte a realizar no desenvolvimento de um agente baseado em pares PR. A produção do mesmo está fora do âmbito, mas fica um conjunto de ferramentas que pode servir de base a esta arquitectura, para além de poder ser usada em muitas outras aplicações PLN para o português.

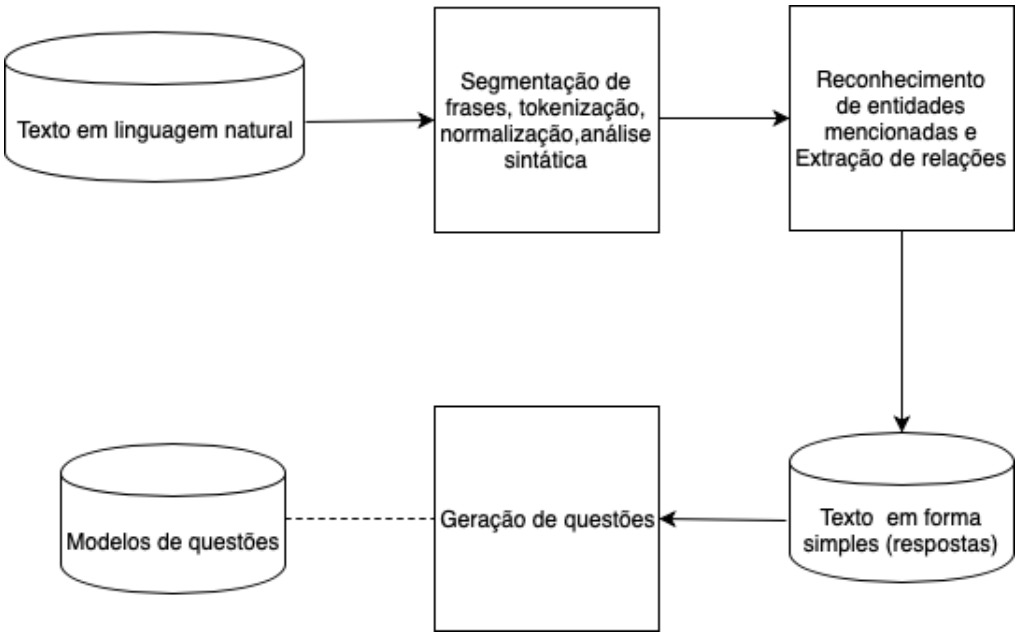


Figura 2.6: Arquitectura de um programa de geração de perguntas. Inspirado pelo modelo apresentado por Silveira (2008).

Capítulo 3

A cadeia de processamento NLPyPort

A principal contribuição do trabalho desenvolvido é uma nova cadeia de Processamento de Linguagem Natural para o Português, que utiliza a linguagem de programação Python. Como referido anteriormente, os motivos para a escolha desta linguagem devem-se, entre outros, à sua crescente popularidade e à facilidade de aprendizagem, o que permite que exista um público maior para a cadeia de Processamento de Linguagem Natural (PLN).

A cadeia foi desenvolvida a partir da cadeia base NLTK, à qual que foram adicionados módulos ou realizadas alterações a módulos já existentes, que devido à especificidade da língua não conseguiam assegurar resultados tão satisfatórios como desejado, quando comparados com outras cadeias que operam também sobre o português. Foram ainda alterados módulos que utilizavam algoritmos que não eram óptimos e que apresentavam sérias dificuldades de utilização e de alteração, e cuja documentação deixava em dúvida como poderiam ser utilizados.

Tirando partido de outros *toolkits* como o NLPPort, que disponibilizam recursos em formatos genéricos, foram reaproveitados recursos que iriam de outra forma exigir uma elevada carga de trabalho manual. Houve ainda algum cuidado de forma a permitir que a cadeia possa ser facilmente utilizada, e espera-se que desta forma permita que mesmo pessoas com conhecimento limitado, quer ao nível da programação, quer ao nível do PLN, a possam utilizar.

O seguinte exemplo demonstra a capacidade do NLPyPort de obter mais informações sobre a frase quando comparada com a *pipeline* de partida, apesar de a informação fornecida a ambas ser a mesma.

Tomemos a frase *"O António Costa deu um passeio no Porto."*

Os resultados obtidos após aplicação de ambas as cadeias (original e criada) podem ser observados na Figura 3.1 .

Como se pode ver, o número de átomos obtidos é diferente e há mais informação para cada um deles. Assumindo que estes estão correctos, permitem mais aplicações futuras pois existem mais dados sobre os quais trabalhar.

As próximas secções irão descrever com maior detalhe as alterações que foram realizadas nos módulos das diferentes tarefas e o resultado final obtido. Começaremos com o atomizador (*tokenizador*), ao qual foi acrescentada uma nova camada, que permite agora

O António Costa deu um passeio no Porto.

Átomo	*PoS*
O,	art
António,	prop
Costa,	prop
deu,	v-fin
um,	art
passeio,	n
no,	adv
Porto,	prop
.	punc

Átomo	*Lemma*	*PoS*	*Entidades*
O,	o,	art,	O
António,	antónio,	prop,	B-PESSOA
Costa,	costa,	prop,	I-PESSOA
deu,	dar,	v-fin,	O
um,	um,	art,	O
passeio,	passeio,	n,	O
em,	em,	prp,	O
o,	o,	art,	O
Porto,	porto,	n,	B-LOCAL
.	.,	punc,	O

Figura 3.1: Resultados obtidos pela cadeia base NLTK (esquerda) e pela cadeia montada NLPyPort (direita).

realizar separação de contracções e clíticos. De seguida, expoem-se o Part-of-Speech (PoS) Tagger, que foi alterado para utilizar dois corpos manualmente anotados. Posteriormente iremos apresentar um módulo adicionado para realizar uma tarefa que não estava disponível de base na *pipeline* a lematização. Ou seja, este módulo iria ser construído de raiz, tirando partido de recursos já existentes. Depois disso será abordado o Reconhecimento de Entidades Mencionadas (REM), módulo em que foi aplicado um algoritmo diferente que permite não só obter melhor resultados mas também facilita o processo de treino utilizando diferentes textos, processo que se revelou extremamente complexo na *pipeline* original.

Finalmente serão expostas as tarefas de Extracção de Relações e a introdução à geração de perguntas, módulos que se revelaram complexos e para os quais se apresentam apenas abordagens iniciais.

3.1 Atomizador

Sendo o passo inicial, os resultados do atomizador são de extrema importância pois um erro num átomo (*token*) propagar-se-á para as etapas seguintes, uma vez que a maioria (se não todas) as tarefas relacionadas com PLN necessitam de utilizar os átomos.

Como tal, é necessário analisar os erros que podem estar a ocorrer no atomizador base do NLTK e perceber como podem ser melhorados. A forma escolhida de fazer isto foi comparar os resultados obtidos pela *pipeline* ao processar um dos recursos considerados como sendo padrão de ouro com os esperados pelo mesmo, o Bosque.

Depois deste processo, foi possível concluir que existem três tipos de excepções críticas onde o atomizador parece falhar: na resolução de **clíticos**, **contracções** e **entidades**. Observemos para já as duas primeiras excepções.

Os **clíticos** são formas de palavras criadas por concatenação, usando um hífen, a partir de uma forma verbal e de um pronome próprio (“[eu] vi-o” -> “[eu] vi a ele”). Por sua vez, as **contracções** são formadas quando uma preposição e um pronome ou artigo são concatenados numa só palavra (“no” -> “em o”).

Estes elementos não são considerados pelo NLTK uma vez que não são habituais nou-

tras línguas, sendo no entanto muito utilizados na língua portuguesa e específicos à mesma. De forma a resolver este problema, fez-se uso de listas que foram já criadas (manualmente) para uso no NLPPort. Estas listas foram escritas no formato XML, e as regras estão num formato semelhante ao seguinte:

```
<replacement target="ao">a o</replacement>
```

Este exemplo é usado para a resolução de uma contracção, "ao", que se procura no texto e é substituída por "a o".

Para a resolução de clíticos podemos observar o seguinte exemplo, que opera de uma forma semelhante ao anterior:

```
<replacement target="-me"> a mim</replacement>
```

É ainda de notar que a lista de resolução de clíticos tem 214 clíticos e respectivas substituições, e a resolução de contracções tem por sua vez 154 contracções e respectivas substituições.

Assim, um pequeno módulo que filtra estas palavras foi acrescentados depois do atomizador do NLTK que verifica se as palavras contraídas existem na lista, e caso existam transforma-as na sua forma mais estendida, aumentando por isso o número de palavras e consequentemente a informação que é possível obter das mesmas.

Esta operação em termos de código traduziu-se em isolar e fornecer ao NLTK a frase para que ele realize a sua atomização, e depois chamar para cada uma das excepções uma função que consulta a lista e compara todos os termos com todos os átomos fornecido pelo NLTK, parando quando há um correspondente e substituindo o mesmo.

A figura 3.2 mostra a diferença de atomização utilizando o NLTK ou o NLPyPort. Como pode ser visto, o NLPyPort gera três átomos adicionais, tendo resolvido uma contracção e um clítico.

```
Brinquei com a pulseira dela, e ela deu-me a mão.
```

NLTK	NLPyPort
Brinquei	Brinquei
com	com
a	a
pulseira	pulseira
dela	de
,	ela
e	,
ela	e
deu-me	ela
a	a
mão	mim
.	deu
	a
	mão
	.

Figura 3.2: Resultados obtidos na atomização pela cadeia base NLTK e pela cadeia montada NLPyPort.

A adição deste módulo melhorou significativamente os resultados, como iremos ver no capítulo seguinte.

Considerando agora a terceira excepção, a resolução de entidades, os erros encontrados devem-se ao facto de a anotação manual distinguir os nomes das entidades juntando as palavras e substituindo os espaços por "underscores" ("_") ("Presidente da República" -> "Presidente_da_República"). A forma de resolver esta excepção é voltar a alimentar o atomizador após o processo de Reconhecimento de Entidades Mencionadas, sendo que os átomos para estas já se encontrariam unidos. A utilização deste método melhorou ligeiramente os resultados mas o seu impacto nos mesmos foi inferior ao das excepções referidas anteriormente.

3.2 Part of Speech Tagger

O passo seguinte a considerar na *pipeline* é o PoS Tagger. O NLTK já apresentava um PoS Tagger que tinha um modelo para a língua portuguesa; este, no entanto, tinha sido treinado no corpus **Floresta sintá(c)tica** que, como foi referido na secção 5, é constituída por várias colecções de texto das quais uma grande parte não foi manualmente revista. Isto pode levar a propagação de erros pois o treino estava a ser realizado com base em recursos que podem ter anotações incorrectas.

De forma a melhorar este processo, realizou-se um treino utilizando um corpo alternativos, o **Bosque**, secção da Floresta que foi manualmente anotada. Um exemplo de anotação PoS pode ser observada na figura 3.3.

```
O João Ferreira é estudante na Universidade de Coimbra.
```

Átomo	*PoS*
O	art
João	prop
Ferreira	prop
é	v-fin
estudante	n
na	adv
Universidade	n
de	prp
Coimbra.	n
.	punc

Átomo	*PoS*
O	art
João	prop
Ferreira	prop
é	v-fin
estudante	n
em	prp
a	art
Universidade	n
de	prp
Coimbra	prop
.	punc

Figura 3.3: Resultados obtidos na marcação PoS pela cadeia base NLTK (esquerda) e pela cadeia montada NLPyPort (direita).

Para realizar este treino, foi utilizado o código disponibilizado em https://-github.com/fmaruki/Nltk-Tagger-Portuguese, que realiza o treino facilmente tendo apenas de ser indicadas as *tags* que se pretende que sejam convertidas para outras *tags* (por exemplo, "N|DATA" para "num") de forma a que o Tagset seja uniforme. O resultado deste treino é um ficheiro ".pickle", que pode depois ser facilmente utilizado pelo NLTK, sendo apenas necessário indicar que se quer utilizar o mesmo. O bloco de código seguinte exemplifica como este poderia ser utilizado.

```
def load_tagger(model_path):
```

```
        f = open(model_path, 'rb')
        return (pickle.load(f))

def nlpyport_pos(tokens,model_path):
        tagger = load_tagger(model_path)
        tags = [tagger.tag(tokens)]
        return tags
```

Estas alterações permitiram melhorias na marcação PoS, como iremos ver no capítulo seguinte.

3.3 Lematizador

Um dos processos mais trabalhosos da *pipeline* foi o módulo de lematização. Devido ao largo leque de aplicações que este pode ter, decidiu-se que era essencial para a cadeia de processamento; no entanto, o NLTK não fornece nenhum lematizador para a língua Portuguesa e este teve por isso de ser implementado de raiz.

Por outro lado, a plataforma NLPPort, em Java, mas de código aberto, inclui um lematizador. Foi por isso decidido que faria sentido reutilizar os recursos deste e até aproveitar a arquitectura de modo a replicar, em Python, o seu lematizador, o LemPort.

A arquitectura do LemPort pode ser dividida e considerada em duas partes: a primeira consiste na utilização de um **léxico de ocorrências** mais comuns e a segunda na aplicação de **conjunto de regras hierarquicamente definidas**. Mais precisamente, caso a palavra não exista no léxico, as regras são aplicadas para realizar modificações no átomos até se chegar a um lema válido.

O funcionamento da primeira parte (léxico de ocorrências) é simples. O programa percorre o léxico de palavras não lematizadas e caso a palavra a testar exista e a *tag* PoS seja a mesma que a descrita na entrada, a sua forma lematizada é devolvida e o processo de lematização termina aqui. Este léxico (Ranchhod, Mota e Baptista, 1999) é constituído por um total de 938.450 entradas inflacionadas obtidas de um léxico de 120,000 lemas, e uma entrada neste léxico assemelha-se ao seguinte exemplo:

```
abafavas,abafar.V+z1:I2s
```

O primeiro elemento, "abafavas", corresponde à forma inflacionada, o segundo corresponde ao lema da palavra, neste caso "abafar", o terceiro elemento corresponde à marcação PoS ,neste caso "V". Depois destes elementos que aparecem sempre podem aparecer elementos adicionais que indiquem subtipos da palavra ou outros atributos morfo-sintácticos, estes no entanto não são necessários para a lematização da forma como é feita. As *tags* PoS permitem desambiguar qual a forma de lematização correcta a usar, como se torna evidente com o seguinte exemplo:

```
mata,matar.V+z1:P2s:P4s:P3s:Y2s
mata,mata.N+z1:fs
```

Dependendo de a palavra se referir ao verbo "matar" ("O assassino voltou a **matar**.") ou ao nome mata ("Os corpos estavam enterrados na **mata**.") o lema obtido irá ser diferente, respectivamente "matar" e "mata".

De forma a realizar apenas uma vez a leitura do léxico (em vez de reler a lista para cada

átomo) é utilizada uma lista de entradas que é criada no inicio da lematização, usando para isso uma estrutura de dados com este propósito.

É ainda necessário referir que cada palavra tem uma classificação e que este é utilizado para lematizar a palavra na forma mais comum possível. Esta classificação é atribuída tendo em conta frequência da palavra e foi determinado pelas listas frequência de palavras no projecto AC/DC — Acesso a Corpora/Disponibilização de Corpora — um projecto com o objectivo de disponibilizar corpora português via Internet, com certa de 40 milhões de palavras em texto português (Santos e Bick, 2000). Tal como acontece para as entradas do léxico, os valores destas entradas são carregados para a memória no início do processo de lematização para reduzir tempo de consulta e processar assim mais rapidamente para quantidades elevadas de texto. Uma entrada deste dicionário terá um aspecto semelhante ao seguinte exemplo:

```
40595 sim
```

Em que o número "40595" corresponde ao número de ocorrências da palavra "sim".

De forma a retornar a palavra mais comum, todos as possíveis normalizações para uma dada palavra, obtidas com recurso ao léxico, são primeiro guardadas numa estrutura do tipo lista em python e depois o seu rank é procurado nesta lista de frequências. No fim deste processo a palavra mais provável de corresponder à lematização correcta é retornada.

Caso uma palavra não seja encontrada, é necessário realizar normalizações para formas progressivamente mais simples e voltar a testar a sua existência no léxico. Desta forma, o tamanho do léxico a consultar é mais reduzido pois apenas contem as versões mais simples de cada palavra, e não contem as suas variantes — por exemplo, contem a forma simples da palavra e não necessita da forma no plural, pois a palavra irá ser processada até alcançar a forma simples. Se assumirmos a palavra "normais" , e esta não for encontrada no léxico, irá ser realizada uma normalização de número para a palavra "normal". Esta palavra por sua vez já existirá no léxico e irá ser retornada de forma a terminar o processo de lematização. Por este motivo, após cada normalização aplicada pelo segundo módulo o léxico é consultado de novo. Outro exemplo ilustrativo é a forma verbal "trabalhei" que normalizada para "trabalhar".

A segunda parte aplica um conjunto de regras hierárquicas que actuam sobre o átomos nos casos em que o léxico não consegue encontrar palavras. Para isto, um conjunto de regras é aplicado de forma hierárquica, resultando em palavras progressivamente mais normalizadas. A ordem de normalização é a seguinte:

- (1) Normalização de advérbios
- (2) Normalização de número
- (3) Normalização de superlativos
- (4) Normalização de aumentativos
- (5) Normalização de diminutivos
- (6) Normalização de género
- (7) Normalização de verbos (regulares e irregulares)

Tal como no processo anterior, esta ordem permite assegurar que existe um menor número de regras que é necessário aplicar para normalizar palavras. Para ilustrar isto, observemos o seguinte exemplo:

Palavra inicial: "brancas"

A primeira normalização realizada seria a normalização de número:

"brancas" –> "branca"

A regra encontrada para normalizar esta palavra seria a que considera um "s" no fim da palavra e o remove de forma a colocar a palavra no singular. Posteriormente seria aplicada a regra de normalização de género:

"branca" –> "branco"

A regra aqui aplicada encontra o "a" em que a palavra termina e transforma-o em "o". Devido à implementação hierárquica seria necessário haver uma regra que procura as letras "as" e as substitui por "o", pois a palavra já tinha posteriormente sido normalizada em termos de número.

É ainda importante referir que, quando se realiza qualquer processo de normalização é ainda verificada a marcação PoS da palavra. Este processo é realizado pois é necessário conhecer a classificação da palavra para assegurar a lematização correta. Por exemplo, as frases *"Quando é que te casas?"* e *" Tenho duas casas."* irão resultar em duas normalizações diferentes, "casar" e "casa" respectivamente.

Este último exemplo permite ilustrar uma das principais vantagens deste módulo: resolução de palavras ambíguas ao identificar a sua origem. Para casos mais difíceis existe ainda a tarefa de *Word Sence Desambiguation* que pode ser traduzido para o português como desambiguação do sentido da palavra, mas esta não será aqui abordada.

3.4 Reconhecimento de Entidades Mencionadas

O módulo de Reconhecimento de Entidades Mencionadas utiliza todos os elementos anteriormente descritos para identificar entidades. O NLTK disponibiliza um módulo de REM, mas devido à falta de documentação este é quase impossível de treinar com novos *corpus* ou de ser alterado; mesmo depois de horas de exploração e com guias detalhados sobre como realizar o treino com outros corpus (como o guia anexo de Pires (2017)) ainda se obtiveram erros devido às *tags* utilizadas serem diferentes, tendo de algum modo de ser fornecidas mas não estando isto explicado em nenhum local da documentação. Por esse motivo, um novo módulo de REM pronto a usar foi acrescentado à *pipeline*. Este é baseado em Conditional Random Fields (CRF), algoritmo que foi aplicado anteriormente na tarefa de REM com sucesso na língua portuguesa (Pires, 2017). O CRF utilizado é o disponibilizado pelo CRFsuite[1], originalmente criado por Cho et al. (2013), e que foi depois adaptado para o scikit-learn, uma biblioteca de aprendizagem computacional disponível de forma aberta em Python.

Para treino deste módulo foi utilizada a colecção HAREM, constituída pelo primeiro HAREM(Santos et al., 2006) e segundo HAREM(Freitas et al., 2010b). Estes utilizam a notação BIO descrita anteriormente.

Para além da facilidade de treino que este módulo de CRF pronto a usar apresenta,

[1] http://www.chokkan.org/software/crfsuite/

os resultados do mesmo melhoraram comparativamente aos obtidos pelo disponibilizado pelo NLTK, como iremos ver na secção seguinte.

O trabalho realizado originou três versões de REM na forma de modelos já treinados, sendo que cada modelo melhorava no anterior. A primeira versão, mais simples, utilizava o CRF mas não tinha em conta as letras maiúsculas, uma característica importante para a detecção de entidades. A segunda versão implementava o mesmo que a primeira mais as letras maiúsculas. A última e mais recente versão utilizava o mesmo que as anteriores mas tinha em conta ainda uma outra característica: os "Word Embeedings".

Embora não seja o foco desta tese, é importante compreender o que são "Word Embeedings". Estes tratam-se de modelos de distribuição semântica que utilizam vectores numéricos que são aprendidos de forma não supervisionada, tendo em conta a sua distribuição em diversos textos. Embora o resultado deste processo possa ser difícil de compreender para um ser humano, a utilização prática destes modelos mostra que são efectivamente capazes de agrupar palavras em grupos semanticamente próximos, o que pode ser relevante quando se pretende identificar entidades (Chiu e Nichols, 2016).

Existem modelos já criados e foi um destes que foi acrescentado de forma a compreender se melhores resultados poderiam ser obtidos considerando estes uma característica. O modelo escolhido, devido à sua popularidade e resultados obtidos por pares foi o FastText [2](Joulin et al., 2016). Entre outros textos, o modelo utiliza como fonte para análise de distribuição de palavras a Wikipédia. Este modelo considera não só a semântica da palavra mas também a forma da mesma.

Na versão final são consideradas para a classificação utilizando o CRF as seguintes características, considerando uma janela de 5 átomos:

- Átomo (e.g., João)
- Sinal de Pontuação (e.g.,"!",",")
- Apenas caracteres ASCII
- Apenas caracteres minúsculos
- Apenas caracteres maiúsculos
- Apenas caracteres alfabéticos
- Apenas números (e.g.,"123)"
- Caracteres Alfa-Numéricos (e.g.,"lote **92B**")
- Começa com letra maiúscula (e.g., João)
- Termina com "a"
- Termina com "s"
- Forma da palavra - "A" sendo uma letra maiúscula, "a" uma minúscula, "#" numero e "-" puntuação" (e.g.,"Margarida2." ficaria "Aaaaaaaaa#-")
- Tamanho da palavra (número de caracteres)
- Prefixos (e.g.,"**pre**ocupar")

[2]https://fasttext.cc

- Sufixos (e.g.,"chuv**isco**")
- Tags PoS (e.g.,"v-inf", "nprop")
- Lema (e.g., "casa", derivado do átomo "casarão".)
- Word Embeddings

Com a excepção dos Word Embeddings que, como referido anteriormente, são obtidos utilizando modelos criados previamente todos os elementos acima descritos podem ser obtidos por observação directa do átomo, ou por processamento utilizando os elementos da cadeia de PLN descritos anteriormente.

3.5 Extracção de Relações

O módulo de extracção de relações pode ser bastante complexo tendo em conta que pretende identificar como elementos da frase se relacionam. Para simplificar este processo, assumiu-se que apenas existiriam relações entre duas entidades, e que esta relação estaria nas palavras entre a primeira e segunda entidade. Adicionalmente foi decidido que se iria tentar extrair informação de uma forma aberta, em vez de tentar procurar relações de tipos específicos. Destes pontos de partida, foram construídos dois sistemas de extracção de relações.

3.5.1 FactPort - primeira versão

O primeiro sistema foi construído para a participação na tarefa de avaliação conjunta Collovini et al., 2019. Este sistema utiliza em parte as ideias de Hearst (1992) e Pantel e Pennacchiotti (2006) de estabelecer um sistema de regras para encontrar relações, utilizado para isso sequências de *tags* PoS, que vão sendo aprendidas automaticamente. Devido à sua complexidade este sistema irá ser descrito nas secções seguintes.

Padrões baseados em Tags PoS

A ideia do sistema desenvolvido foi utilizar relações encontradas para desenvolver um conjunto de regras que pudessem ser aplicadas em múltiplas situações. A primeira ideia, como apresentada por Hearst (1992), seria fazer um sistema que utilizaria directamente as palavras. Contudo, este sistema não é aplicável de forma geral — é necessário que apareçam as palavras especificamente detectadas anteriormente para ser detectada a relação. De forma a combater a falta de generalidade decidiu-se que iria ser usado um elemento ainda próximo das palavras, mas ao mesmo tempo mais geral — as *tags* de marcação PoS. Assim, o sistema começa por identificar a sequência de *tags* PoS para uma dada frase e depois adiciona estas seu conjunto de regras. Isto permite que se mantenha um teor mais geral que consegue identificar relações de vários tipos.

Aprendizagem de regras

De forma a obter regras, o sistema necessita de um conjunto de frases com relações préviamente identificadas. O sistema começa por obter cada uma das etiquetas morfossintácticas

(*tags* PoS) das palavras, e percorre depois a frase marcando as posições em que a palavra identificada faz parte da relação, e consequentemente as *tags* que fazem parte da relação. Observemos a seguinte frase:

```
A Apple opera em os EUA.
```

Fornecendo ao sistema a relação a encontrar:

```
opera em
```

O sistema começa por encontrar para a frase as *tags* PoS correspondentes, e marca as posições das palavras da relação na frase, ou seja:

Frase :	A	Apple	opera	em	os	EUA	.
Pos :	art	prop	v-fin	prp	art	prop	punc
Parte da relação :	0	0	1	1	0	0	0

O resultado final desta regra seria então:

```
0011000|art prop v-fin prp art prop punc
```

Neste caso, os 0s correspondem a posições em que a palavra não faz parte da relação, e os 1s a posições em que faz. A parte final (depois de "|") corresponde à sequência de *tags* em si.

As regras são posteriormente ordenadas por tamanho; esta ordenação tem como objectivo garantir que o sistema tenta primeiro detectar casos mais específicos, que são por esse motivo maiores, e caso estes falhem avance para casos mais gerais. A existência de regras de grande e pequena dimensão garantem que o sistema consegue simultaneamente manter especificidade e generalidade, aproveitando-se das vantagens de cada um destes modos de abrangência.

Um exemplo de uma regra mais específica pode ser:

```
00000000000001000010000111|art n v-fin adv art n n conj adv v-inf
v-ger adv v-fin prp art n punc v-pcp prp art n n punc punc prp art n
```

Enquanto que uma regra mais geral pode ser tão simples como:

```
1|v-fin
```

Quando o sistema encontra numa frase uma sequência que corresponde a uma regra que reconhece, ele retorna as palavras nas posições que estão marcadas com os 1s.

Geração de regra adicionais

De forma a garantir que o sistema gera o máximo de regras possíveis que façam sentido é realizado um passo extra que leva à geração de mais uma regra. Este consiste em adicionar regras que partem de uma regra de maiores dimensões e com muitos 0s, e retirar os mesmos, mantendo apenas os 1s da relação. A lógica disto assume que se a relação

tinha previamente sido identificada e utilizada como exemplo, então continuará a fazer sentido mesmo que as palavras entre elas sejam removidas.

Pegando no exemplo anterior, para além da regra gerada:

```
0011000|art prop v-fin prp art prop punc
```

Seria também gerada uma nova regra utilizando apenas os elementos que pertencem à regra:

```
11|v-fin prp
```

Este método permite assim utilizar regras mais especificas para gerar regras mais gerais, aumentando assim a cobertura do sistema.

3.5.2 FactPyPort - segunda versão

Embora o sistema anterior tenha mostrado uma performance satisfatória no IberLEF, existem dúvidas sobre até que ponto as relações identificadas são significativas. Isto irá ser explicado e explorado na secção 4, mas partindo deste princípio, um segundo sistema foi posteriormente construído.

Este sistema é inspirado no sistema ReVerb(Etzioni et al., 2011b), utiliza também um conjunto de regras sobre as *tags* PoS, mas ao contrário do descrito anteriormente, estas regras são definidas à partida, não sendo por isso necessário dados de treino. Mais que um conjunto de regras, a forma de detecção de relações recorre ao uso de expressões regulares, permitindo assim que este seja aplicado a um conjunto mais geral de situações. Após a aplicação deste conjunto de regras foi possível concluir que, apesar da regra expressão regular ter sido desenvolvido originalmente para o inglês, pode em parte ser aplicada ao Português. Este sistema apenas utiliza apenas regras, descritas por uma única expressão regular que pode ser vista na Figura 2.3.

Para exemplificar consideremos a seguinte frase:

```
O João é filho do Carlos.
```

Após adição da marcação PoS obtemos:

Frase :	O	João	é	filho	de	o	Carlos	.
PoS :	art	prop	v-fin	adj	prp	art	prop	punc

A relação encontrada pelo ReVerb será a seguinte:

```
é filho de
```

O padrão encontrado pode ser descrito pela expressão VW*P, correspondendo o átomo "é" ao primeiro verbo, seguido do adjectivo "filho", e acabando com a preposição "de".

Devido aos factores mencionados anteriormente, o teste de extracção de factos torna-se um processo complicado e moroso. Por esse motivo, este sistema de extracção de factos foi desenvolvido mas apenas brevemente testado, funcionando como prova de conceito para a sua aplicabilidade para a língua portuguesa. Para isso, um pequeno número de exemplos

foram escolhidos e utilizados para testar o sistema. Os resultados podem ser observados na tabela 3.1, sendo que a relação está no formato de triplos, constituído pela primeira entidade, relação e segunda entidade separadas por vírgulas.

Frase	**Relação**
O João Diogo estuda em Coimbra.	João Diogo,estuda em,Coimbra
O Académico ganhou ao Benfica.	Académico ,ganhou a o ,Benfica
O Sporting perdeu 1-0 com o Porto.	Sporting ,perdeu ,1-0
Uma Licença de Actividade custa 500 euros.	Uma Licença de Actividade ,custa ,500 euros
Uma Licença de Actividade custa 500 €.	Uma Licença de Actividade ,custa ,500 €
Linus Torvalds criou o sistema operativo Linux, que foi lançado em 1991.	Linus Torvalds ,criou o ,Linux Linux ,foi lançado ,em 1991 Linux ,lançado ,em 1991
Portugal tem uma divida de 500 Milhões à Europa.	Portugal ,tem uma ,500 Milhões Portugal ,divida de ,500 Milhões
A Ordem Dos Engenheiros (OE) realiza exames a 21 de Agosto.	Ordem Dos Engenheiros ,realiza exames ,a 21 de Agosto

Tabela 3.1: Triplos extraídos das frases pelo módulo de extracção de informação.

Cada frase foi dada ao sistema apenas uma vez, mas em casos em que o sistema detectou mais do que duas entidades, tentou encontrar relações entre elas e o resultado disto são múltiplas relações partindo de uma só frase.

3.6 Geração automática de Perguntas

Um último elemento que foi criado como prova de conceito de uma possível aplicação do NLPyPort foi a geração de perguntas. Este sistema simples assume que foram encontradas relações através de uma das técnicas anteriormente descritas e parte do principio que existe um triplo (Entidade 1, relação, Entidade 2) e que se sabe a categoria de cada uma destas entidades. A partir deste ponto, o sistema recorre a modelos que usam as categorias das entidades para determinar a formatação da pergunta. Observemos um exemplo disto com a seguinte frase:

O João vive em Viseu.

Assumindo que a relação é correctamente extraída, ela é representada pelo triplo:

(João, vive em, Viseu)

E as entidade encontradas serão, respectivamente, da categoria Pessoa e Local. O sistema irá depois percorrer os modelos e encontrar um que utilize estes duas categorias de entidade:

PESSOA LOCAL

Onde <relação> <entidade> ?

Quem <relação> <entidade> ?

O primeiro caso irá substituir <entidade> pela entidade identificada como pessoa, enquanto que o segundo irá substituir <entidade> pela entidade identificada como local. Em ambos os casos a <relação> irá ser substituída pela relação encontrada, neste caso "vive em", sendo que no primeiro caso a preposição "em" irá ser removida. As questões obtidas a partir deste modelo serão as seguintes:

Onde <vive> <João> ?

Quem <vive em> <Viseu> ?

De forma a gerar questões é então necessário gerar primeiro modelos de perguntas. Os modelos criados para demonstração de conceito podem ser vistos na tabela 3.2.

Tipo Entidade 1	**Tipo Entidade 2**	**Perguntas**
PESSOA	LOCAL	Onde <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
PESSOA	PESSOA	Quem <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
PESSOA	VALOR	Quando/Quanto <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
OBRA	VALOR	Quando/Quanto <relacao> <entidade> ?
		O que <relacao> <entidade> ?
PESSOA	ABSTRACCAO	O que <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
ABSTRACCAO	TEMPO	Quando <relacao> <entidade> ?
		O que <relacao> <entidade> ?
ORGANIZACAO	TEMPO	Quando <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
LOCAL	VALOR	Quanto <relacao> <entidade> ?
		Quem <relacao> <entidade> ?
VALOR	LOCAL	Quem <relacao> <entidade> ?
		Quanto <relacao> <entidade> ?

Tabela 3.2: Modelos de geração de perguntas baseados nas categorias das entidades.

A partir das relações obtidas e ilustradas na tabela 3.1 foram geradas questões, utilizando para isso modelos e categorias das entidades. O resultado disso pode ser visto na tabela 3.3.

Relação	**Pergunta**	**Resposta**
João Diogo ,estuda em ,Coimbra	Onde estuda João Diogo ?	Coimbra.
	Quem estuda em Coimbra ?	João Diogo.
Académico ,ganhou a o ,Benfica	Quem ganhou Académico ?	Benfica.
	Quem ganhou a o Benfica ?	Académico.
Sporting ,perdeu ,1-0	Quando/Quanto perdeu Sporting ?	1-0.
	Quem perdeu 1-0 ?	Sporting
Uma Licença de Actividade ,custa ,500 euros	Quando/Quanto custa Uma Licença de Actividade ?	500 euros.
	O que custa 500 euros ?	Uma Licença de Actividade.
Uma Licença de Actividade ,custa ,500 €	Quando/Quanto custa Uma Licença de Actividade ?	500 €.
	O que custa 500 € ?	Uma Licença de Actividade.
Linus Torvalds ,criou o ,Linux	O que criou Linus Torvalds ?	Linux.
	Quem criou o Linux ?	Linus Torvalds.
Linux ,foi lançado ,em 1991	Quando foi lançado Linux ?	Em 1991.
	O que foi lançado em 1991 ?	Linux.
Linux ,lançado ,em 1991	Quando foi lançado Linux ?	Em 1991.
	O que lançado em 1991 ?	Linux.
Portugal ,tem uma ,500 Milhões	Quanto tem divida Portugal ?	500 Milhões.
	Quem tem uma 500 Milhões ?	Portugal.
Portugal ,divida de ,500 Milhões	Quanto tem divida Portugal ?	500 Milhões.
	Quem divida de 500 Milhões ?	Portugal.
Ordem Dos Engenheiros ,realiza exames ,a 21 de Agosto	Quando realiza Ordem Dos Engenheiros ?	A 21 de Agosto.
	Quem realiza exames a 21 de Agosto ?	Ordem Dos Engenheiros.

Tabela 3.3: Perguntas e respostas geradas a partir de factos extraídos pelo sistema.

Apesar de haver muito espaço para melhorias, seria possível gerar perguntas utilizando estes modelos. Os resultados deste caso parecem bastante optimistas mas é necessário relembrar que foram obtidos tendo em conta que a informação se encontrava sempre entre duas entidades consecutivas, e que o extractor de relações tinha a capacidade de extrair relações deste tipo.

É ainda importante destacar que é possível extrair duas perguntas de um só facto, o que permite um maior número de perguntas, que para contexto de Agentes Conversacionais é o desejado.

Embora apenas brevemente abordados, estes exemplos de utilização servem como prova

de conceito de que a cadeia poderá ser, em futuras versões, utilizada para geração de questões a partir de factos.

Capítulo 4

Experimentação

Neste capítulo são descritas as várias formas de avaliação e validação da *pipeline* previamente descrita. Atendendo ao facto de que cada tarefa é diferente, foi necessário testar cada uma individualmente e de formas diferenciadas, como se verá nas próximas secções. Para permitir uma melhor compreensão dos resultados da avaliação é importante saber as métricas utilizadas, que serão explicadas na secção que se segue. Posteriormente serão descritos os testes realizados que foram usados para determinar qual a melhor ferramenta, que use a linguagem de programação Python, para usar como base para o desenvolvimento da *pipeline* e os testes realizados para avaliar cada módulo que foi alterado ou adicionado.

4.1 Métricas de avaliação

Ao longo deste trabalho foram aplicadas diversas métricas de avaliação. Para a compreensão das mesmas serão explicadas como estas foram calculadas para cada tarefa.

Para as tarefas de Atomização, Marcação PoS e de Lematização, a métrica escolhida foi acerto por átomo(PTA, *Per token accuracy*). Esta métrica começa por marcar todas as instâncias, que neste caso são representadas por linhas de um ficheiro constituído por àtomos no caso da atomização, àtomos seguido das *tags* no caso de Marcação PoS e àtomos seguidos dos lemas no caso da lematização. Cada linha é assinalada como estando correcta se está nos resultados esperados e incorrecta caso contrário. Posteriormente divide-se o número de instâncias correctas pelo número de instâncias esperadas. Isto traduz-se na seguinte fórmula:

$PTA = \frac{ICA}{IE} * 100$

Em que *ICA* é o número de instâncias correctamente assinaladas e *IE* corresponde ao número de instâncias esperado.

Observemos o exemplo da figura 4.1. O *ICA* será neste caso dez, o *IE* será quinze. Calculando o resultado de acerto por átomo obtemos:

$PTA = \frac{10}{15} * 100 = 66,67\%$

Para a avaliação do reconhecimento de entidades mencionadas, a métrica utilizada foi a descrita na secção 2.3.2. Relembrando, estas tratam-se de Precisão, Abrangência e Medida-F.

```
Brinquei com a pulseira dela, e ela deu-me a mão.
```

Esperado	Obtido	Correcto
Brinquei	Brinquei	1
com	com	1
a	a	1
pulseira	pulseira	1
de	dela	0
ela	,	1
,	e	1
e	ela	1
ela	deu-me	0
a	a	1
mim	mão	1
deu	.	1
a		Na
mão		Na
.		Na

Figura 4.1: Exemplo de assinalação de átomos como correctos ou incorrectos.

$$\text{Precisão (P)} = \frac{(EntidadesEsperadas) \cap (EntidadesObtidas)}{EntidadesObtidas}$$

$$\text{Abrangência (A)} = \frac{(EntidadesEsperadas) \cap (EntidadesObtidas)}{EntidadesEsperadas}$$

$$\text{Medida-F} = 2 * \frac{(P*A)}{(P+A)}$$

É ainda importante referir e explicar a avaliação realizada recorrendo à competição IBERLEAF. O IBERLEF (Iberian Languages Evaluation Forum) [1] é uma avaliação conjunta que visa melhor os resultados de processamento de linguagem natural, no domínio das linguagens de Ibéricas. De entre as várias tarefas disponíveis, optou-se pala participação em duas. A primeira trata-se de uma tarefa de REM, em que o objectivo era encontrar e classificar entidades de categorias pré-definidas, e a segunda trata-se de uma tarefa de extracção de informação. Nesta última, o NLPyPort foi o único participante. A participação nesta competição resultou no artigo *NLPyPort:Named Entity Recognition with CRF and Rule-Based Relation Extraction* (Ferreira, Gonçalo Oliveira e Rodrigues, 2019b).

Tendo em conta tudo isto serão agora ser apresentados os resultados dos diversos testes realizados.

4.2 Escolha de ferramentas PLN

Numa fase inicial, grande parte do trabalho consistiu em testar as ferramentas já existentes — OpenNLP, NLPPort, SpaCy e NLTK, analisadas no capítulo 2 — de modo a perceber quais estariam mais aptas a ser adaptadas e integradas numa *pipeline* para a língua portuguesa em Python. Como referido no capítulo 2, o **Bosque** é um corpo português com anotações morfossintácticas revistas manualmente e, por isso, amplamente utilizado

[1] https://iberlef.sepln.org

no treino e avaliação de ferramentas para o processamento computacional do português. Por este motivo foi escolhido para testar as ferramentas. Para assegurar também que este apresenta os resultados mais correctos possíveis, foi escolhido o **tagset** mais estendido, descrito na secção anterior, uma vez que este obriga a uma classificação mais aprofundada do que um *tagset* mais superficial, que vai de acordo com a anotação realizada no Bosque. Por exemplo, um verbo irá ser considerado com uma forma extensa como "v-fin" em vez de apenas ser marcado com "v", adicionando assim mais informação à marcação.

Assim, o primeiro passo foi testar as ferramentas como se encontram de raiz, isto é, foram usados os modelos que estas ferramentas disponibilizam para o processamento do português. Para além de definir a língua alvo como sendo o português, não foi realizada nenhuma configuração adicional ou treino de novos modelos. Com estes testes foi possível determinar quais as ferramentas mais adequadas para cada tarefa.

De forma a avaliar estas ferramentas, foram realizados dois testes. O teste de **atomização** consistiu em fornecer ao programa frases separadas por linhas, usar cada ferramenta para realizar a *atomização*, e depois comparar os resultados obtidos com os resultados esperados sobre o Bosque. O objectivo deste teste foi avaliar a capacidade de dividir o texto correctamente em *átomos*, para determinar que alterações poderiam ser necessárias realizar.

Para calcular os resultados deste teste utilizou-se a métrica acerto por átomo descrita na secção anterior.

No primeiro teste, foram obtidos os resultados na tabela 4.1.

Ferramentas	**atomização**
OpenNLP	75%
NLPPort	99%
SpaCy	81%
NLTK	83%

Tabela 4.1: Percentagem de *átomos* correctos obtidos

Como podemos ver, o NLPPort apresenta resultados significativamente melhores do que os restantes *toolkits*, pelo que podemos assumir que as camadas acrescentadas ao OpenNLP, que apresenta os piores resultados de *atomização* nesta comparação, tiveram um forte impacto. A razão para esta diferença deve-se ao facto da existência de processamento prévio à atomização, baseado em listas, que separa os *átomos* que estão comprimidos, como é o caso dos **clíticos** e **contracções**, o que leva a melhores resultados pois são obtidos os *átomos* esperados. Parte desta diferença deve-se ainda à realização de uma segunda passagem dos átomos, depois de utilizado o módulo de REM que junta entidades num só átomo (E.g "Inês_Ferreira"), o que se encontra de acordo com a anotação do Bosque e consequentemente são obtidos mais átomos esperados.

SpaCy e NLTK apresentam resultados semelhantes, estando no entanto distantes do NLPPort.

O teste seguinte foi a avaliação da marcação PoS de cada ferramenta. O objectivo deste teste é determinar se as ferramentas conseguem, tendo à partida os *átomos* correctos, fazer a marcação PoS adequada para a palavra.

De forma a testar justamente todas as ferramentas, foi fornecida a mesma sequência de átomos, um por linha, para serem marcados. Os átomos referidos foram também obtidos a partir do Bosque e estarão, por isso, correctamente definidos. Depois desta

marcação, foi realizada a comparação entre os valores esperados (conforme a marcação PoS do **Bosque**) e os obtidos. Os resultados podem ser vistos na tabela 4.2. A métrica "Frases completamente correctas" mede a percentagem de frases, constituídas por átomos e respectivas *tags* PoS, que estão na íntegra iguais às esperadas.

Ferramentas	Marcação PoS	Frases completamente corretas
OpenNLP	88%	35%
NLPPort	88%	35%
SpaCy	41%	0%
NLTK	60%	2%

Tabela 4.2: Comparação de marcações PoS correctamente atribuídas

Como seria de esperar, o OpenNLP e NLPPort têm o mesmo resultado, quer nas *tags* PoS, quer nas frases. Isto acontece pois ambos são treinados da mesma forma e com o mesmo modelo (entropia máxima) e as regras usadas pelo NLPPort para a *atomização* não são aqui aplicadas. Podemos ainda verificar que o SpaCy apresenta um resultado muito inferior aos resultados obtidos pelas restantes ferramentas em termos de marcação das *tags* PoS. Isto pode acontecer por diversos motivos, como por exemplo o modelo utilizado ou o corpo utilizado para o treinar ser diferente do corpo de teste. De notar que o modelo do SpaCy foi treinado com o corpo **MacMorpho**.

Tendo em conta o desempenho menos bom para as tarefas base do SpaCy, considerou-se o NLTK como principal candidato em Python. É ainda de notar que o Spacy executa todo o funcionamento de uma vez, ou seja, existe uma *pipeline* pré definida e a mesma é chamada e realiza todos os passos definidos; mesmo que so se pretenda obter, por exemplo, os átomos irá ser também corrido o módulo de marcação PoS. Isto torna mais difícil realizar alterações ao SpaCy porque é sempre necessário alterar código interno, o que não acontece com o NLTK.

Os testes posteriores apenas foram realizados tendo em conta esta ferramenta Python e as duas alternativas em Java.

Para estas ferramentas foi depois testado o desempenho da *atomização* seguida da marcação PoS, como ocorreria normalmente ao ser usada a *pipeline*, com o objectivo de determinar como estas se comportariam em ambiente normal. Para isso, foram fornecidas frases de entrada que depois foram transformadas em *átomos* pela ferramenta, e posteriormente anotadas, também por esta. Os resultados obtidos podem ser vistos na tabela 4.3.

Ferramentas	Marcação PoS	Frases completamente corretas
OpenNLP	75%	9%
NLPPort	87%	24%
NLTK	58%	2%

Tabela 4.3: Comparação de marcações PoS após *atomização* correctamente atribuídas

Ao contrário do que acontece com a marcação PoS, quando feita de forma isolada, podemos verificar que os resultados são mais baixos para o OpenNLP do que para o NLPPort. Isto explica-se porque este processo é realizado em cadeia e existe uma propagação dos erros do atomizador, o que afecta os resultados da marcação PoS. Ou seja, os erros na *atomização* vão influenciar a marcação de PoS, por esta também se basear na sequência de átomos. Confirmamos ainda que o NLTK apresenta resultados inferiores.

4.3 Alteração do PoS Tagger

De forma a melhor compreender como se poderia melhorar a tarefa de marcação PoS, começou-se por estudar mais atentamente a ferramenta. Este estudo permitiu observar que NLTK utilizava para o treino da marcação PoS o corpus **Floresta sintáctica** completo, enquanto todas as outras ferramentas apenas usam o **Bosque**. Apesar da maior dimensão da Floresta Sintáctica, a sua anotação pode não ser a mais adequada, porque, com excepção precisamente do Bosque, foi produzida automaticamente e não foi manualmente revista. Para além disso os testes realizados estão a usar o Bosque como anotação de referência, o que faz com que as ferramentas que o usem para treino tenham expectavelmente resultados melhores. Por esse motivo o corpus de treino da tarefa de marcação PoS do NLTK foi alterado para utilizar o Bosque. A tabela 4.4 mostra os resultados obtidos após realizar esta alteração.

Ferramenta	Atomização	Marcação PoS	Frases completamente corretas
NLTK Original	83%	60%	2%
NLTK (Bosque)	83%	85%	27%

Tabela 4.4: *Átomos* e marcações PoS correctamente atribuídas após alteração dos dados de treino

Os resultados obtidos pelo *atomizador* mantiveram-se, o que seria de esperar uma vez que apenas se alterou o treino da marcação PoS, e os resultados PoS, como previsto, foram melhorados.

O passo seguinte foi verificar desempenho da marcação de PoS quando utilizado sobre o resultado da *atomização*. A tabela 4.5 mostra os resultados obtidos.

Ferramenta	Marcação PoS	Frases completamente corretas
NLTK Original	58%	2%
NLTK (Bosque)	79%	14%

Tabela 4.5: Marcações PoS após *atomização* correctamente atribuídas e após alteração dos dados de treino

Podemos verificar que houve uma melhoria também neste teste. Estes resultados são esperados porque utilizando **Bosque** para treino todos os *toolkits* ficam em pé de igualdade, uma vez que utilizam para treino os mesmo dados.

4.4 Alteração do Atomizador

O módulo que se identificou à partida como um forte concorrente a ser melhorado foi o Atomizador. Como pode ser visto pelos resultados anteriores existe para este um grande espaço para melhorias.

De forma a resolver os problemas relacionados com clíticos e contracções foram utilizadas listas de ocorrências que continham para estes as formas estendidas correspondentes. Depois de realizar a *atomização* em si, as ocorrências de clíticos ou contracções foram substituídas pelas suas formas estendidas.

A ferramenta NLPPort já faz esta etapa de pré-processamento com base em listas XML, como as demonstradas na secção 3.1, criadas precisamente para esse fim e disponibilizadas

no repositório GitHub da ferramenta. Devido a este formato ser compatível com o Python, foi possível reutilizar estas regras e implementar com relativa facilidade esta etapa. Após esta adaptação voltaram-se a fazer os testes e obtiveram-se os resultados na tabela 4.6.

Ferramenta	Atomização	Marcação PoS	Frases completamente corretas
NLTK Original	83%	60%	2%
NLTK (atomizador melhorado)	90%	85%	27%

Tabela 4.6: *Átomos* e de marcações PoS correctamente atribuídas após processamento pelo módulo de pré-processamento de palavras

Como seria de esperar, os resultados para a *atomização* foram melhorados sem qualquer alteração na marcação PoS quando considerada individualmente, uma vez que para esta são fornecidos os *átomos* correctos de forma a testar isoladamente marcação. Porém, quando testados em conjunto, os valores para a marcação PoS sobre o output do *Atomizador* foram consideravelmente melhorados, em comparação com os valores iniciais (tabela 4.5), como se pode ver na tabela 4.7.

Ferramenta	Marcação PoS	Frases completamente corretas
NLTK Original	58%	2%
NLTK (atomizador melhorado)	82%	18%

Tabela 4.7: Marcações PoS após *atomização* correctamente atribuídas e após processamento pelo módulo de pré-processamento de palavras

A adição posterior do módulo de REM permitiu utilizar uma arquitectura semelhante à do NLPPort, em que o Atomizador é alimentado uma segunda vez depois serem encontradas as entidades. Isto permite uma melhor resolução dos átomos o que leva a uma melhoria dos resultados, como pode ser visto na tabela 4.8.

Ferramenta	Atomização	Marcação PoS	Frases completamente corretas
NLTK Original	83%	60%	2%
NLTK (após REM)	93%	85%	27%

Tabela 4.8: *Átomos* e de marcações PoS correctamente atribuídas após processamento pelo módulo de pré-processamento de palavras, utilizando uma segunda passagem pelo atomizador

Por sua vez, esta melhoria na atomização, leva mais uma vez a uma melhoria nos resultados da marcação PoS quando comparados com os iniciais, quando esta está a ser alimentada pelo mesmo. Com estas alterações, foram obtidas os resultados da tabela 4.9.

Ferramenta	Marcação PoS	Frases completamente corretas
NLTK Original	58%	2%
NLTK (após REM	84%	24%

Tabela 4.9: Marcações PoS após *atomização* correctamente atribuídas após processamento pelo módulo de pré-processamento de átomos, utilizando uma segunda passagem pelo atomizador

4.5 Adição de um Lematizador

Como descrito anteriormente, o NLTK não possui de raiz um lematizador para língua portuguesa. Dada a amplitude de aplicações do mesmo, entendeu-se que um deveria ser integrado. De forma a avaliar o desempenho do novo lematizador, e tendo em conta que este se trata de uma reprodução parcial do lematizador inserido na *pipeline* NLPPort, os resultados de ambas as *pipelines* foram comparados com as anotações presentes no Bosque e foram obtidos os resultados da tabela 4.10.

Ferramentas	**Lemas**
NLPPort	98.7%
NLPyPort	98.62%

Tabela 4.10: Comparação entre lemas obtidos e lemas do Bosque, para ambas as cadeias.

Como se pode ver, os resultados foram bastante semelhantes, tendo apenas uma diferença de 0.08%. As diferenças entre as *pipelines* devem-se principalmente a diferenças de PoS Tagging, elemento necessário para decidir como deve ser lematizado um átomo. Um exemplo de como a marcação PoS pode afectar o lematizador foi visto na secção 3.3.

Embora os resultados não sejam 100% idênticos, considera-se que este desempenho é satisfatório pois por vezes é debatível qual o lema correto, o que dependerá também do tipo de aplicação. Por exemplo, no Bosque podemos ver a palavra "jogadoras" ser lematizada para "jogadora", enquanto que ambas as *pipelines* lematizam também o género, ficando por isso a palavra "jogador".

Considerou-se por este motivo que o lematizador terá sido correctamente implementado e que se encontra pronto para uso.

4.6 Adição do REM-CRF

O módulo seguinte acrescentado completamente de raiz foi o REM. Vários testes foram realizados de forma a validar o novo módulo de REM, sendo que inicialmente foi comparado com os resultados de Pires (2017) e mais tarde foram criadas e testadas novas versões para utilização na tarefa de avaliação conjunta do IberLEF 2019. Iremos abordar cada um destes resultados na secção respectiva.

4.6.1 CRF base e validação com HAREM

De forma a comparar o novo módulo de REM com o módulo para o mesmo fim incluído no NLTK, utilizou-se como base os resultados apresentados por Pires (2017). Este utiliza a colecção HAREM, constituída pelo Primeiro HAREM (Santos et al., 2006) e pelo Segundo HAREM (Freitas et al., 2010b). O tipo de validação escolhido para este teste foi a validação cruzada em 10 partes (em inglês 10 fold cross validation), onde os dados são divididos em 10 partes, treinados em 9 e testados com o restante, sendo que se testa uma vez com cada uma das partes e no fim se calcula a média de todas as partes. Um exemplo deste teste realizado apenas com 4 partes pode ser visto na figura 4.2).

As mesma validação em 10 partes utilizada por Pires (2017) foi utilizada para todos os testes de forma a garantir que a divisão aleatória não iria beneficiar nem prejudicar os resultados.

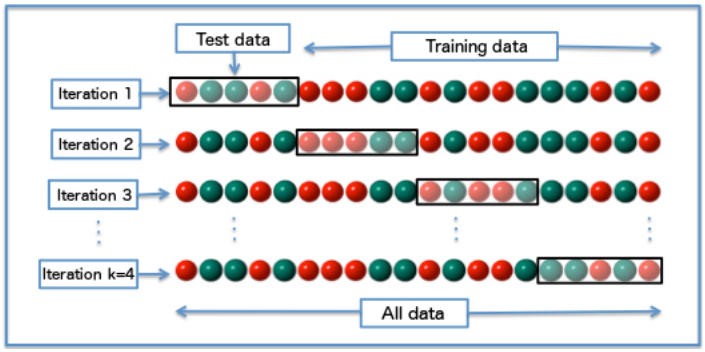


Figura 4.2: Exemplo do funcionamento do teste para 4 folds, (https://en.wikipedia.org/wiki/Cross-validation(*statistics*)).

Os resultados obtidos eram superiores aos obtidos por Pires (2017) utilizando o NLTK, como se pode ver pela seguinte tabela:

Sistema	**Precisão(%)**	**Revocação(%)**	**F-1(%)**
NLTK	30.58	31.38	30.97
CRF	57.05±2.45	46.56±1.29	51.28±1.49

Tabela 4.11: Comparação da Precisão, Revocação e Medioda F-1 do NLTK REM e CRF REM

Tentativas de melhoria do CRF

O CRF utiliza para aprendizagem dos pesos um algoritmo de gradiente descendente chamado de **L-BFGS** — Limited-Memory BroydenFletcherGoldfarbShanno . Este utiliza dois parâmetros para a regulação do método L-BFGS(Sha e Pereira, 2003), nomeadamente L1 e L2. Estes parâmetros que devem ser usados e testados para melhorar a sua performance.

No teste anterior foram utilizados parâmetros apurados por Lopes, Teixeira e Gonçalo Oliveira (2019). Decidiu-se posteriormente testar como a variabilidade destes parâmetros afectaria o sistema. Para isso, os valores foram alterados e foram treinados em 70% da colecção HAREM, e testados nos restantes 30%. No mapa de calor representado na figura 4.3 podemos ver como a variação dos mesmos afecta os resultados obtidos pelo sistema.

Como pode ser visto, os melhores valores foram obtidos quando ambos L1 e L2 eram mais baixos. Porém, escolhendo os melhores resultados com L1 igual a 2^{-6} e L2 igual a 2^{-4} e realizando depois o mesmo teste de 10-folds, os resultados obtidos pioraram em relação aos anteriores, o que pode ser visto na figura 4.12. Isto sugere que houve algum tipo de overfitting — em que o estudo dos parâmetros garante o melhor resultado mas só para o caso em questão, estando por isso muito ajustado para este — e que valores mais altos podem levar a melhores resultados em casos reais. Tendo em conta que não houve melhorias, foram mantidos os valores anteriores.

4.6.2 IBERLEF

A avaliação IBERLEF (Collovini et al., 2019) surgiu como uma hipótese de validação do sistema. Esta avaliação consistia em identificar num texto entidades, de acordo com um

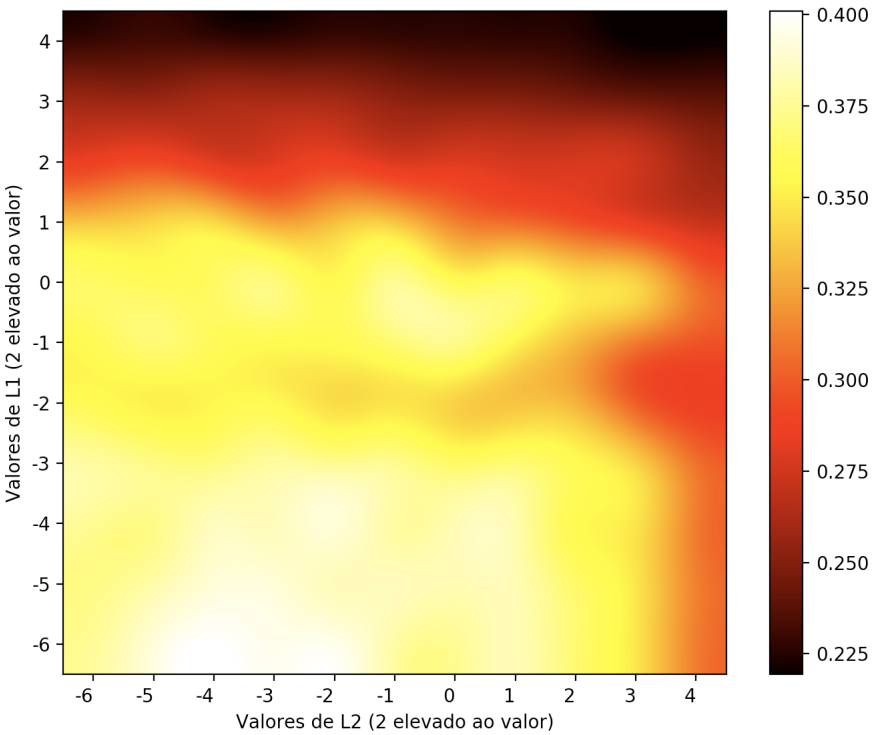


Figura 4.3: Mapa de calor ilustrativo da Medida F1 obtida para cada variação dos parâmetros L1 e L2.

Sistema	**Precisão(%)**	**Revocação(%)**	**F-1(%)**
CRF	57.05±2.45	46.56±1.29	51.28±1.49
CRF com L1 e L2 melhorados	55.92±2.02	46.97±1.26	51.06±1.44

Tabela 4.12: Comparação da Precisão, Revocação e Medida F-1 do CRF usado e CRF com parâmetros melhorados.

conjunto de categorias de entidades pré-definido (e mais reduzido que o considerado no sistema base) constituído por **PER** — Pessoa, do inglês *Person*, **ORG** — Organização, do inglês *Organization*, **PLC** — Local, do inglês *Place*, **TME** — tempo, do inglês *Time* e **VAL**— Valor, do inglês *Value*. Assim, de forma a participar na competição foi criado um pequeno módulo que fazia a conversão das *tags* obtidas no sistema inicial para as da competição. O sistema foi depois enviado e os organizadores realizaram os testes. É de notar que os participantes não sabiam que textos seriam usados para avaliar os sistemas. Os resultados obtidos pelo sistema podem ser vistos na seguinte tabela:

Os resultados aparentaram estar muito àquem dos esperados pelo que se decidiu fazer uma análise das razões do problema. Depois de receber os resultados soube-se que foram utilizados os corpus SIGARRA(Pires, 2017), para as *tags* gerais, e o HAREM para os valores (VAL).

É de notar que apenas o *dataset* geral foi disponibilizado por motivos legais, pelo que apenas este pode ser usado em novas experiências, com a finalidade de compreender onde o sistema falhou e onde pode ser melhorado.

Depois de uma primeira análise, descobriu-se um erro na *pipeline* que estaria a afectar o desempenho do sistema – todos os átomos estavam a ser convertido para ter letra minús-

Corpus	Categoria	Precisão(%)	Revocação(%)	F1
Dataset Policial	PER	21.72	53.07	30.83
Dataset Clínico	PER	27.27	26.25	26.75
Dataset Geral	Total	26.08	19.78	22.50
	ORG	19.41	12.62	15.30
	PER	50.07	34.34	40.74
	PLC	42.31	20.64	27.75
	TME	8.99	11.11	9.94
	VAL	56.60	54.55	55.56

Tabela 4.13: Resultados para a tarefa de REM , por categoria de entidade, com CRF treinado na colecção HAREM (oficial).

cula, pelo que todas as características que utilizavam letra maiúscula (como a forma da palavra) não eram considerados aí. Este erro não tinha sido detectado durante a fase de testes anteriores pois nesta os átomos eram fornecidos directamente ao módulo de CRF, enquanto que para a competição tinham de atravessar toda a *pipeline*.

Depois de realizar estas correcções, obtivemos os resultados da tabela 4.14.

Corpus	Categoria	Precisão(%)	Revocação(%)	F1
Dataset Geral	Total	28.63	24.10	26.17
	ORG	24.45	17.56	20.44
	PER	55.53	38.84	45.71
	PLC	44.32	25.99	32.77
	TME	7.48	10.41	8.70
	VAL	100.00	100.00	100.00

Tabela 4.14: Resultados para a tarefa de REM, por categoria de entidade, com CRF treinado na colecção HAREM (oficial), utilizando letras maiúsculas.

Como podemos ver, os elementos que foram testados exclusivamente com o HAREM obtiveram resultados de 100%, e houve algumas melhorias nos restantes resultados.

De forma a despistar outros possíveis problemas, decidiu-se também realizar o treino do sistema com o SIGARRA em conjunto com a colecção do HAREM de forma a garantir que com os dados de treino correctos o sistema seria capaz de uma performance boa. Obviamente este teste é meramente exploratório e os resultados não poderiam ser considerados pois está a ser usado, para treino e teste, o mesmo corpus. Por outro lado, antes da avaliação não tínhamos esta informação, e nada nos teria impedido de enviar um modelo treinado em ambas colecções, ambas publicamente disponíveis. Os resultados obtidos podem ser vistos na tabela 4.15.

Corpus	Categoria	Precisão(%)	Revocação(%)	F1
Dataset Geral	Total	95.23	95.49	95.36
	ORG	90.83	93.61	92.20
	PER	95.74	95.74	95.74
	PLC	93.16	93.92	97.54
	TME	98.38	96.99	97.68
	VAL	99.06	99.06	99.06

Tabela 4.15: Resultados para a tarefa de REM, por categoria de entidade, com CRF treinado na colecção HAREM (oficial) e a colecção SIGARRA em simultâneo.

Como seria de esperar, os resultados indicam que treinando com os corpus utilizados para testar a performance do sistema seria melhor. Isto pode também indicar que, modelos gerados utilizando um corpus específico poderão não possuir a capacidade de identificação de entidades noutros corpus. É ainda de reparar que os resultados para a categoria "VAL",

que antes eram de 100%, diminuíram neste caso. Isto pode indicar que o treino do CRF com outros dados poderá piorar a sua performance.

Uma outra alteração a que se decidiu recorrer para tentar melhorar os resultados obtidos pelo CRF foi o acréscimo de características baseadas em *word embeddings*, representações vectoriais de palavras de acordo com o seu contexto em grandes colecções de texto. A utilização destas representações tem melhorado os resultados de várias tarefas e, para o português, há vários modelos deste tipo pré-treinados. Estes permitem através de técnicas de aprendizagem computacional agrupar palavras semanticamente semelhantes em **clusters**, o que pode ser uma característica importante para o CRF. Os resultados obtidos encontram-se na tabela 4.16.

Corpus	**Categoria**	**Precisão(%)**	**Revocação(%)**	**F1**
Dataset Geral	Total	23.42	25.62	24.47
	ORG	20.96	18.87	19.86
	PER	37.42	45.65	41.13
	PLC	38.54	28.27	32.62
	TME	8.78	13.32	10.58
	VAL	93.40	95.19	94.19

Tabela 4.16: Resultados para a tarefa de REM, por categoria de entidade, com CRF treinado na colecção HAREM (oficial), utilizando os "word embeddings" como uma das características.

Mais uma vez verificamos que a adição de embeddings não melhora os resultados. Isto contraria alguma da literatura (Lopes, Teixeira e Gonçalo Oliveira, 2019) mas acredita-se que existe um motivo para isso — a generalidade das categorias. Ao tratar-se de categoria muito genéricas, em que estão incluídas muitos tipos de entidades, a adição dos "word embeedings" do *fastext* não traz vantagens pois estes não são capazes de agrupar de forma significativa as entidades de cada categoria. No futuro outros modelos devem ser testados com objectivo de compreender se podem melhorar os resultados do módulo de REM.

4.6.3 Versão Final

É importante compreender as escolhas que são realizadas pela versão finalizada, que está neste momento disponível. Os parâmetros utilizados por esta versão podem ser vistos na tabela 4.4.

Parâmetro	**Valor**
Algoritmo de Treino	L-BFGS
Coeficiente L1	0,0625
Coeficiente L2	0,5
Características	Átomo,Sinal de Pontuação,Apenas caracteres ASCII Apenas caracteres minúsculos,Apenas caracteres maiúsculos Apenas caracteres alfabéticos,Apenas números,Caracteres Alfa-Numéricos Começa com letra maiúscula,Termina com a,Termina com s, Forma da palavra,Tamanho da palavra,Prefixos,Sufixos,Tags PoS,Lema
Categorias	PESSOA,TEMPO,OBRA,ORGANIZAÇÃO,ABSTRACÇÃO,LOCAL ACONTECIMENTO,VALOR,COISA,OUTRO

Figura 4.4: Valores dos parâmetros utilizados pela versão do REM final.

O CRF permite ainda observar qual o impacto das características (negativo e positivo) na sua performance, impacto este que pode ser obtido observando os pesos que o grafo atribui a cada ligação. A figura 4.5 ilustra de forma global — isto é, para todas as

categorias — quais as características que mais pesam na decisão do CRF. Por sua vez, a figura 4.6 mostra as características que mais negativamente influenciam a decisão do CRF.

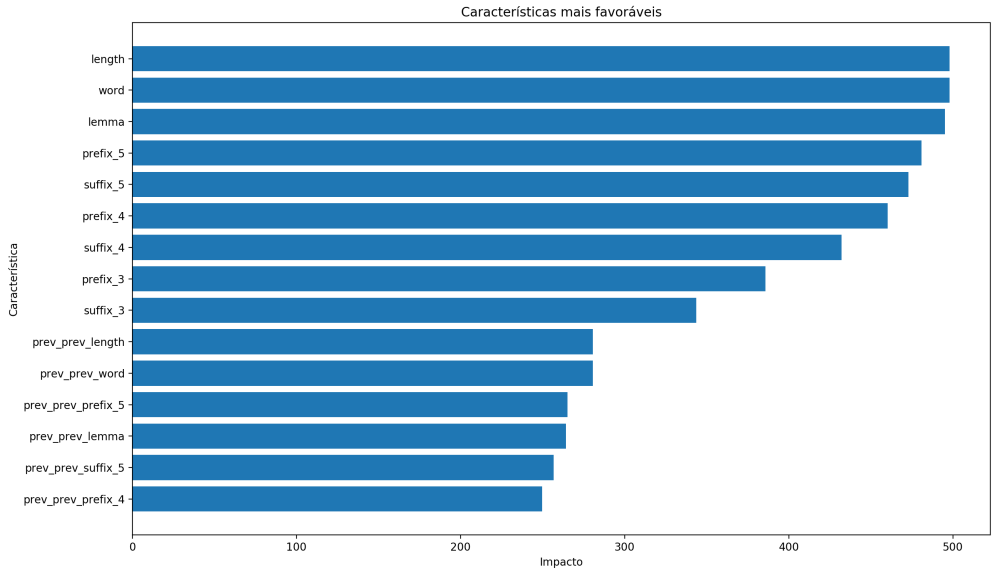


Figura 4.5: Histograma de impactos positivos de cada característica.

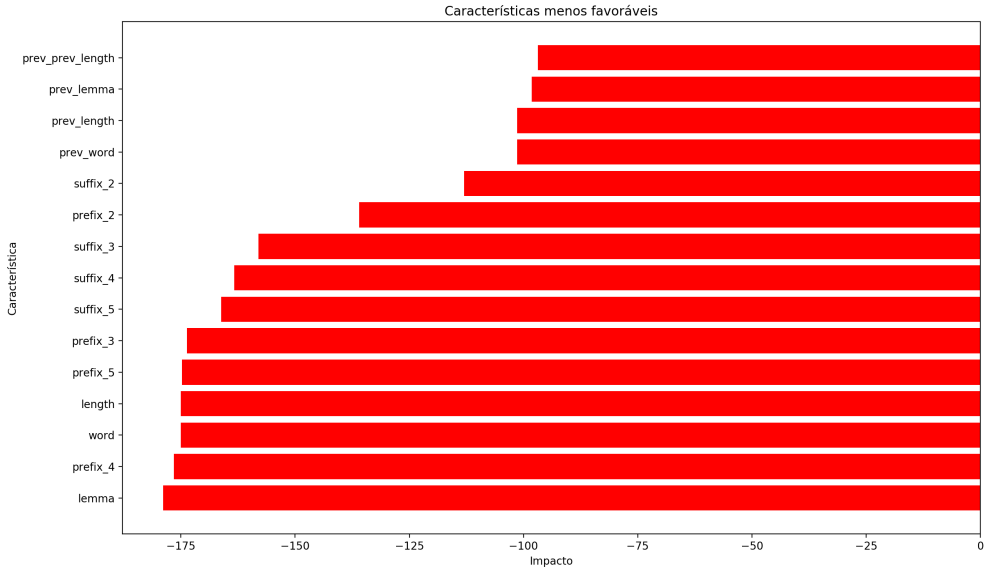


Figura 4.6: Histograma de impactos negativos de cada característica.

Como pode ser visto, existem características que são simultaneamente positivas e negativas. Isto acontece porque a mesma característica pode ajudar para uma das categorias, e pode não ser significativa ou até negativa noutra categoria.

Para compreender os resultados é ainda importante compreender que uma repetição de parte da característica significa que esta se refere a um grau superior de distância. Por outras palavras, se virmos algo a dizer “prev_length”, isto refere-se ao tamanho da

palavra anterior à palavra em teste, e se virmos algo que diga "prev_prev_length" refere-se ao tamanho da palavra duas posições antes da palavra em teste. Adicionalmente, pode também aparecer um número à frente que indica a profundidade de caracteres. Por exemplo, a expressão "prefix_5" indica que se trata do prefixo considerando uma janela de 5 letras.

Olhando para as características de topo podemos compreender até certo ponto o porquê de dadas características terem mais impacto; a característica primária é "word", a palavra em si. Isto faz sentido pois a palavra é geralmente necessária para determinar o tipo. De seguida temos o tamanho e o lema. A primeira não será surpreendente pois muitas categorias são constituídas por nomes longos (e.g., Politécnico); o lema por sua vez pode ter impacto de diversas formas. Um lema semelhante à palavra pode indicar que esta não sofreu lematização e é por isso entidade, mas pode também evidenciar melhor se uma palavra é (ou pertence) a uma entidade ao evidenciar a origem da palavra inflacionada — por exemplo, a palavra "Liga", que lematizada para "liga" evidencia que esta se trata de uma competição, mas se fosse lematizada para "ligar" mostraria que esta se tratava de um verbo e seria portanto improvável ser uma entidade.

As características seguintes são os sufixos e prefixos, de diversos graus de profundidade. Esta característica dever-se-á provavelmente à existência de palavras com origens semelhantes e que podem ser por isso mais fáceis de classificar. Um exemplo disso são as palavras "Algarve", "Alentejo" e "Alcácer", todas locais e ambas iniciadas pelo prefixo de origem árabe "AL".

É interessante destacar que em muitos casos em que estas características são as mais relevantes e com mais impacto, noutras são negativamente avaliadas. Pegando no exemplo anterior, pode ser o caso que o sistema tente avaliar grande parte das palavras iniciadas por "AL" como sendo do tipo local, podendo no entanto não ter tipo e ser de outra categoria (e.g., alpaca). Para melhora analisar o porquê destes casos podemos observar a figura 4.7, que mostra o impacto que cada característica tem na categoria.

Como seria de esperar, cada categoria tem uma série de características favoráveis, sendo que algumas delas são fáceis de compreender e outras não. Por exemplo, para a categoria **Pessoa**, podemos ver que o lema tem extrema importância. Isto seria de esperar uma vez que os nomes, quando lematizados ficam inalterados e consequentemente o CRF assume que se um átomo não foi lematizado, este deve tratar-se de uma entidade do tipo Pessoa.

4.7 Adição do módulo de Extracção de Informação

O módulo seguinte a testar foi o módulo de extracção de informação. Uma vez que a ideia deste módulo seria extrair a informação de forma aberta, seriam necessários textos ou recursos textuais assinalados desta forma. No entanto, a maioria dos recursos disponíveis com relações anotadas estão limitados a um conjunto finito de relações.

Devido a esta falta de recursos, foi desenvolvido um sistema tendo em mente a avaliação IBERLEF (Collovini et al., 2019) que permitiu que o sistema fosse treinado pelos recursos disponibilizados para a mesma, e sendo por isso avaliada neste contexto. Esta avaliação fornecia um conjunto de dados de exemplo reduzido, sendo que os participantes deveriam treinar o seu sistema com estes ou outros dados obtidos de outras fontes. Para além disso, no caso desta tarefa, seriam sempre fornecidas as entidades e o tipo das mesmas, e assumia-se que a relação a extrair encontrava-se sempre entre as duas entidades.

A tabela 4.17 permite observar os resultados obtidos na segunda task da competição IBERLEF.

Exato			**Parcial**		
Precisão (%)	**Revocação (%)**	**F1**	**Precisão (%)**	**Revocação (%)**	**F1**
73.6	71.1	72.4	76.6	74.8	75.7

Tabela 4.17: Resultados do sistema FactPyPort.

O resultado **Exato** trata casos em que houve uma correspondência exacta com as palavras assinaladas como sendo da relação e o resultado **Parcial** corresponde a casos em que apenas parte da relação foi identificada, ou quando existem átomos em excesso. Por exemplo, observemos a seguinte frase.

`... Parlamento enquanto tal, em o espírito e em o seguimento de a sua magnífica declaração, não se pronunciou a favor de o retorno de a paz em Israel e em o Próximo_Oriente.`

A relação esperada seria:

`não se pronunciou a favor de o retorno de a paz em`

e uma relação parcial seria por exemplo:

`pronunciou a favor de o`

ou, por exemplo:

`retorno de a paz em Israel e em o`

Tendo sido os únicos participantes desta competição, é difícil compreender por completo qual a performance do programa desenvolvido. No entanto, dada a complexidade do problema, os resultados aparentam ser satisfatórios, estando na casa dos 73% de F1. Estes resultados foram inesperados pois o programa criado é relativamente simples, e apenas utiliza um conjunto de 100 exemplos para gerar as regras. Decidiu-se por este motivo explorar o que estava a acontecer para que estes valores tenham sido obtidos, e se serão significativos.

Após observar os casos de teste chegou-se à conclusão que os tipos de relações consideradas na competição muitas vezes não transmitia informação útil. De facto, grande parte das relações das frases de teste (49,7%) tinha um de três tipos: “de”, “(” e “em” . Exemplos distos podem ser vistos na tabela 4.18.

Frases	**Entidade 1**	**Entidade 2**	**Relação**
... promovido por a USP (Universidade_de_São_Paulo) de a cidade. ...	USP	Universidade_de_São_Paulo	(
...o virologista Paolo_Zanotto, de a USP, que está preparando...	Paolo_Zanotto	USP	de
... beneficiava a Inglaterra em Portugal transferiram-se automaticamente para o Brasil ...	Inglaterra	Portugal	em

Tabela 4.18: Exemplos de casos das instâncias mais comuns onde a relação pretendida não transmite informação útil.

O resultado que seria de esperar para este caso era que um sistema que realizasse algum tipo de overfitting que levasse a que mais relações dos tipos acima indicados fossem

identificados, naturalmente teria um bom desempenho.

De facto, podemos ver que, em parte, isto é verdade para o sistema desenvolvido. Observemos a tabela 4.19.

	Relação	**Ocorrências**
Esperados	de	43 (28.8%)
	(	18 (12.1%)
	em	13 (8.7%)
Resultados	de	45 (30%)
	(	19 (12.8%)
	em	20 (13.4%)

Tabela 4.19: Relações mais frequentes esperadas e obtidas pelo sistema.

Como podemos ver, o número de relações identificadas pelo sistema que corresponde às anteriormente referidas constitui 56,2% das relações, um aumento de 6,5% em relação às existentes nos dados de teste. Assim, podemos compreender que, embora a performance do sistema tenha sido boa em outros casos, ouve algum overfitting. Exemplos desta situação podem ser observados na tabela 4.20.

Frases	**Relação esperada**	**Relação obtida**
...de Nova_Iorque, Rudy_Giuliani, ... foi presidente de a Câmara ...	presidente de	de
...registo, Martin_SCHULZ (PSE, DE), líder de o PSE, recordou que, em a...	lider de	(
...Helena_Roseta ...em as próximas eleições intercalares para a Câmara_de_Lisboa.	eleições para	em

Tabela 4.20: Exemplos de casos onde as relações mais comuns foram erradamente identificadas .

Porém, tendo em conta a natureza do sistema, mais dados de treino levaria provavelmente a que menos relações fossem identificadas com estes elementos. Isto porque estas relações correspondem às regras mais gerais, e mais casos de treino adicionariam regras mais específicas, e consequentemente poderia haver mais variabilidade na identificação.

Por último, é importante reforçar que, embora o sistema tenha ainda assim uma performance satisfatória, o tipo de relações identificadas transmite geralmente pouca informação. Um exemplo de relações consideradas pode ser visto na seguinte frase:

A reunião começou anteontem em o campus de a UnB (Universidade_de_Brasília).

Entidade 1: “UnB”

Entidade 2: “Universidade_de_Brasília”

Relação esperada: “(”

A interpretação desta relação não é clara e, desta forma, pouco acrescenta ao conhecimento que se tem sobre qualquer uma das entidades ou sobre a relação entre elas. Considerou-se por este motivo que o sistema, embora aplicável para a avaliação IberLEF, não seria o mais indicado para uma *pipeline* que se quer genérica e útil para as mais variadas aplicações que procurem processar texto em português. No caso concreto deste módulo, seria importante formalizar as extracções com vista uma interpretação mais fácil,

logo à partida, por humanos.

4.8 Discussão

Ao longo das últimas secções foram foram apresentados os resultados de várias experiências com vista à validação da nova *pipeline*, incluindo módulos alterados, criados de raiz ou integrados.

Inicialmente foram estudados os *toolkits* identificados como relevantes. Após este estudo concluiu-se que o NLPPort apresenta os melhores resultados para a *atomização* e marcação PoS para a língua portuguesa. Isto seria de esperar uma vez que é o único toolkit que usa listas e regras criadas manualmente e especificamente para o português. No entanto, o NLPPort usa a linguagem de programação Java. Como referido anteriormente, era desejável que fosse criada uma ferramenta em Python, uma vez que a popularidade da língua tem vindo a crescer e o número de ferramentas para o português que a usam é reduzido, tendo havido desde o início deste projecto pessoas que manifestaram interesse em utilizar uma ferramenta nesta linguagem caso ela fosse criada. Assim, de forma a conseguir montar esta nova ferramenta, foram realizadas diversas alterações e ajustes à ferramenta NLTK utilizando para isto recursos já existentes nas outras ferramentas que permitiram aproximar os resultados desta aos *toolkits* em Java que melhores resultados apresentam.

Realizaram-se diversas alterações, começando por alterar o recurso usado para treino, passando a ser utilizado o **Bosque** do *PoS Tagger*, passando depois para a melhoria da tarefa de *atomização* por adição de etapas de processamento extra. Estes dois elementos interligam-se, por isso faz sentido observar quais os valores deste em conjunto, vistos na tabela 4.21.

Ferramenta	Atomização	Marcação PoS	Marcação PoS após atomização
Valores iniciais	83%	60%	58%
Treino do glspos com o Bosque	83%	85% (+25%)	79% (+21%)
Melhorias na atomização	90% (+7%)	85%	82% (+3%)
Realimentação após REM	93% (+3%)	85%	84% (+2%)

Tabela 4.21: Tabela de comparação dos resultados obtidos depois de cada alteração.

Estes resultados foram satisfatórios por permitir aproximar os valores obtidos pelo NLTK aos valores obtidos pelo melhor *toolkit* testado, o NLPPort, sendo que a versão de realimentação após REM foi a versão final alcançada. As melhorias que foram conseguidas até ao momento foram realizadas nos passos iniciais da *pipeline*, o que permite que seja minimizada a propagação de erros.

Posteriormente, foram adicionados os módulos de lematização (inicialmente inexistente no NLTK), que obteve resultados mais uma vez próximos dos do NLPPort; foi adicionado de raiz o módulo de REM que apresenta melhores resultados que o módulo base do NLTK, para além de ser mais simples de treinar. Finalmente foi criado um extractor de factos baseado em sequências de marcações PoS.

Na tabela 4.22 podemos ver todos os valores iniciais do NLTK para os módulos existentes, e todos os resultados obtidos com a nova cadeia montada, NLPyPort. É ainda de notar que o termo desempenho se refere aqui à métrica correspondentemente usada para cada tarefa.

Os resultados foram satisfatórios num todo, melhorando sempre sobre os resultados

Módulo	Desempenho inicial	Desempenho obtido	Corpus
Atomizador	83%	93% (+10%)	Bosque
PoS Tagger	60%	85% (+25%)	Bosque
Atomizador e PoS	58%	84% (+26%)	Bosque
Lematizador	Não Existente	98%	Bosque
REM	31%	51% (+20%)	Colecção HAREM
Extractor de Factos	Não Existente	73%	Sigarra e HAREM parcial

Tabela 4.22: Tabela de comparação dos resultados iniciais e resultados obtidos pela *pipeline* NLPyPort.

base do NLTK e acrescentando ainda mais elementos a esta ferramenta.

A *pipeline* resultante pode ser usada como base para um vasto leque de aplicações no âmbito do processamento computacional do português, e existem já diversas pessoas para quem esta já foi útil, pelo que consideramos que a criação da mesma foi uma mais valia que valeu o investimento de tempo e a sua expansão com adição de mais módulos seria benéfica.

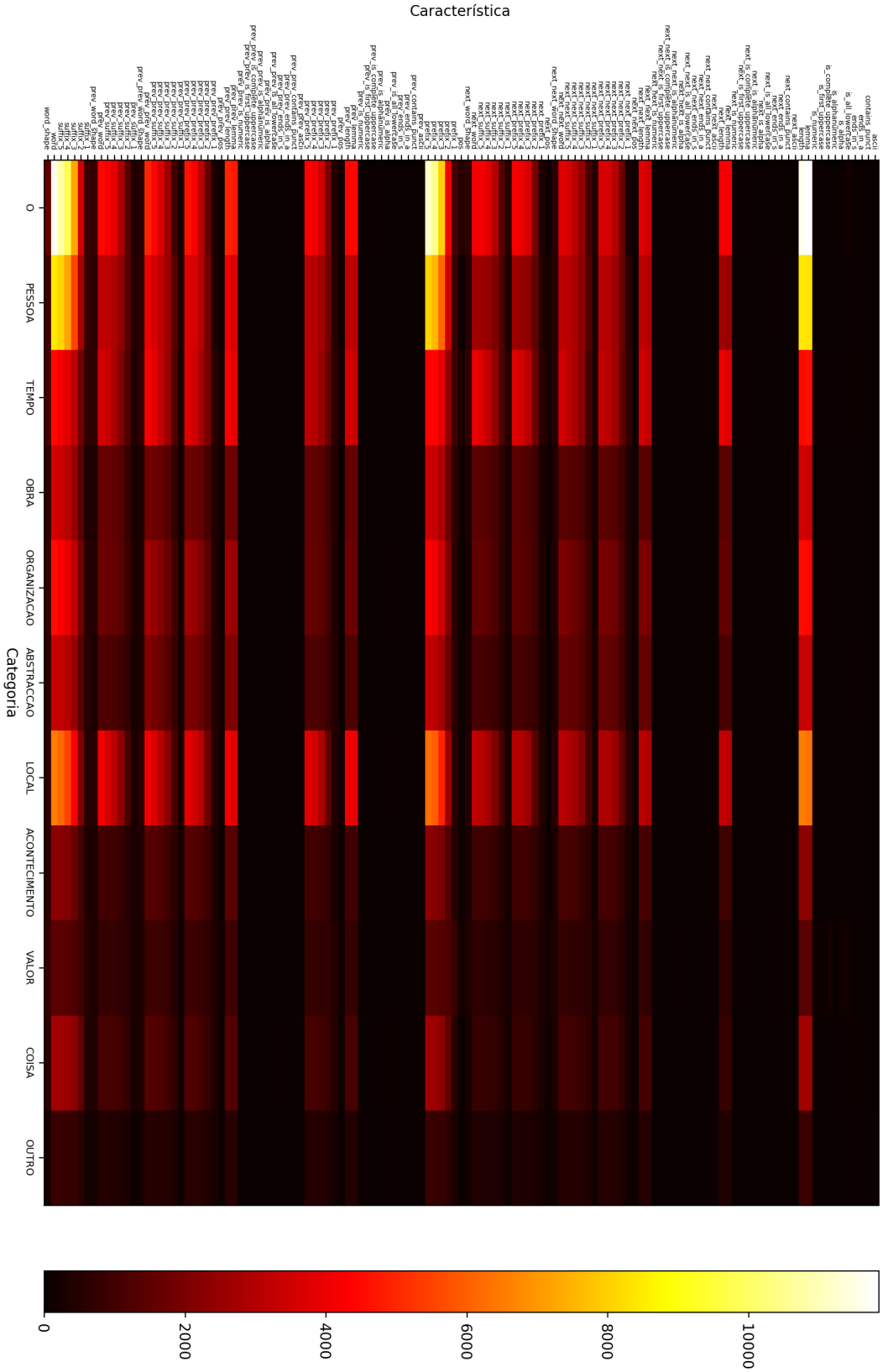


Figura 4.7: Histograma de impactos positivos e negativos de cada característica em cada categoria.

Capítulo 5

Conclusão

Nesta secção é feita uma revisão de todo o trabalho realizado, as contribuições resultantes do mesmo e ainda o trabalho a realizar no futuro.

De forma a alcançar os objectivos propostos para este trabalho realizou-se um estudo acerca das ferramentas, aplicações e recursos já existentes. Foi aprofundado o conhecimento relativo ao funcionamento de uma *pipeline* de Processamento de Linguagem Natural (PLN), tendo para isso cada constituinte da mesma sido estudado em detalhe. Com este estudo pretendeu-se adquirir a familiaridade necessária para a montagem de uma nova *pipeline* de PLN, que possa ser usada por qualquer pessoa noutros trabalhos para, por exemplo, extrair características linguísticas acerca de palavra que podem posteriormente ser exploradas por sistemas inteligentes.

O trabalho realizado começou pelo estudo das ferramentas existentes na língua Python, para posterior adaptação destas com o objectivo de melhorar a mesma e alcançar resultados do mesmo nível de ferramentas semelhantes noutras línguas. Adicionalmente este estudo permitiu começar a visualização e aprendizagem do funcionamento de uma cadeia PLN.Após análise, concluiu-se que diversos módulos poderiam ser melhorados ou acrescentados de raiz.

Depois de compreender qual a melhor ferramenta base, começou-se por realizar melhorias nos módulos de atomização e marcação Part-of-Speech (PoS), recorrendo a recursos já existentes noutras cadeias PLN ou recursos considerados padrões de ouro. Estando estas adaptações concluídas, foi adicionado o próximo módulo que se considerou ser crítico para muitas aplicações, o módulo de lematização, que segue uma arquitectura hierárquica de regras para tornar uma palavra no seu lema, forma como esta apareceria no dicionário.

Foi depois acrescentado um módulo de Reconhecimento de Entidades Mencionadas (REM) que permitiu melhorias significantes sobre aquelas obtidas pelo NLTK base, e que entre outras coisas permite um mais fácil treino com novos modelos.

Por último adicionou-se um módulo de extracção de factos e um módulo de geração de perguntas que, embora até ao momento apenas sirvam como prova de conceito, conseguem em dados domínios ter uma performance relativamente satisfatória.

O sistema criado oferece uma alternativa mais fácil de usar ao NLTK, que apresenta melhores resultados que este e, por estes motivos, trata-se de uma opção viável para os utilizadores da linguagem Python que pretendam processar texto na língua portuguesa. Espera-se ainda que o trabalho nesta *pipeline* continue a ser desenvolvido para completar a mesma com a adição de módulos ou melhoria dos módulos já implementados.

O trabalho realizado resultou numa cadeia de PLN em Python para a língua portuguesa, disponibilizada ao público para utilização de forma aberta, e pode ser obtida em https://github.com/jdportugal/NLPyPort.

Adicionalmente, dois artigos foram escritos. O primeiro artigo, *Improving NLTK for Processing Portuguese* (Ferreira, Gonçalo Oliveira e Rodrigues, 2019a), apresenta a *pipeline* com os módulos iniciais e foi apresentado na conferência SLATE [1]. O segundo artigo, *NLPyPort: Named Entity Recognition with CRF and Rule-Based Relation Extraction*Ferreira, Gonçalo Oliveira e Rodrigues, 2019b, relata os resultados da tarefa partilhada IBERLEF(Collovini et al., 2019), destacando os módulos de REM e o extractor de factos criado para a mesma.

Embora a cadeia de PLN seja já capaz de realizar a maioria das tarefas de processamento desejadas de uma cadeia deste tipo, muitos módulos poderão ainda a ser acrescentados com vista a obter uma cadeia mais completa, ou melhorar os resultados já obtidos. Seria interessante acrescentar um módulo de análise de dependências, que poderia vir a permitir extracção de triplos de factos que não se encontram entre duas entidades.

Seria também benéfico continuar o trabalho nos dois últimos módulos de extracção de informação e geração de questões, que embora tenham sido criados para demonstrar prova de conceito mostraram potencial e podem ser úteis para gerar perguntas, elemento utilizado por aplicações avançadas como Agentes conversacionais. De facto, a dificuldade de testes relativos à geração de perguntas deve-se ao facto de esta se encontrar entre entidades, sendo que a relação consegue no entanto ser extraída sem a mesma. No entanto, de modo a formar triplos de informação, é necessário que existam entidades, sendo que caso fosse acrescentado o módulo referido anteriormente (análise de dependências) estas poderiam ser substituídas por sintagmas nominais.

Para terminar, relembram-se os três objectivos propostos como resultado da presente tese:

- Estudo e selecção do conjunto de ferramentas a ser aplicado na nova cadeia de processamento da língua portuguesa
- Criação e disponibilização de forma aberta uma nova cadeia de PLN, o **NLPyPort**, que apresenta resultados superiores aos actualmente disponibilizados pela ferramenta NLTK e que são comparáveis outras *pipelines*.
- Aplicação da cadeia desenvolvida na extracção de factos e geração de questões, mostrando que isto seria possível e deve vir a ser mais trabalhado no futuro.

Após estudo da das ferramentas existentes, uma nova ferramenta foi montada que, melhora os resultados das ferramentas originais e permite por isso melhor processar o texto em português, utilizando a linguagem de programação Python. Sendo que a ideia era que esta funcionasse de forma aberta, foi disponibilizada livremente no GitHub. Como resultado disto, várias pessoas começaram já a utilizar a *pipeline* para outros projectos, das quais algumas sabemos a aplicação, nomeadamente a sua utilização para para reconhecimento de humor e agentes conversacionais. Considera-se por este motivo que o trabalho realizado foi um sucesso e todos os objectivos foram compridos, e tratando-se por isso de um contributo significativo e útil para o processamento da língua portuguesa.

[1] http://slate-conf.org/2019/home

Bibliografia

Ali, Husam, Yllias Chali e Sadid A Hasan (2010). "Automation of Question Generation from Sentences". Em: *Proceedings of QG2010: The Third Workshop on Question Generation*, pp. 58–67.

Bird, Steven e Edward Loper (2004). "NLTK: The Natural Language Toolkit". Em: *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, p. 31.

Branco, António e João Ricardo Silva (2004). "Evaluating Solutions for the Rapid Development of State-of-the-Art POS Taggers for Portuguese." Em: *LREC*.

Brill, Eric (1992). "A simple rule-based part of speech tagger". Em: *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics, pp. 152–155.

Chiu, Jason PC e Eric Nichols (2016). "Named entity recognition with bidirectional LSTM-CNNs". Em: *Transactions of the Association for Computational Linguistics* 4, pp. 357–370.

Cho, Han-Cheol et al. (2013). "Named entity recognition with multiple segment representations". Em: *Information Processing & Management* 49.4, pp. 954–965.

Collovini, Sandra et al. (2019). "Portuguese Named Entity Recognition and Relation Extraction Tasks at IberLEF 2019". Em: *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2019)*. CEUR Workshop Proceedings. CEUR-WS.org.

Diéguez, Daniel, Ricardo Rodrigues e Paulo Gomes (2011). "Using CBR for Portuguese Question Generation". Em: *Proceedings of the 15th Portuguese Conference on Artificial Intelligence*, pp. 328–341.

Etzioni, Oren et al. (2011b). "Open Information Extraction: The Second Generation". Em: *Proceedings of 22nd International Joint Conference on Artificial Intelligence*. IJCAI 2011. Barcelona, Spain: IJCAI/AAAI, pp. 3–10.

Etzioni, Oren et al. (2011a). "Open Information Extraction: The Second Generation." Em: *IJCAI*. Vol. 11, pp. 3–10.

Ferreira, João, Hugo Gonçalo Oliveira e Ricardo Rodrigues (2019a). "Improving NLTK for Processing Portuguese". Em: *Symposium on Languages, Applications and Technologies (SLATE 2019)*. In press.

— (2019b). "NLPyPort: Named Entity Recognition with CRF and Rule-Based Relation Extraction". Em: *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2019), co-located with 35th Conference of the Spanish Society for Natural Language Processing (SEPLN 2019)*. CEUR Workshop Proceedings. Bilbao, Spain: CEUR-WS.org, pp. 468–477.

Fonseca, Erick Rocha e João Luís G Rosa (2013). "Mac-Morpho Revisited: Towards Robust Part-Of-Ppeech Tagging". Em: *Proceedings of the 9th Brazilian symposium in information and human language technology*.

Fonseca, Erick Rocha et al. (2016). "Visão Geral da Aaliação de Similaridade Semântica e Inferência Textual". Em: *Linguamática* 8.2, pp. 3–13.

Forney, G David (1973). "The Viterbi Algorithm". Em: *Proceedings of the IEEE* 61.3, pp. 268–278.

Freitas, Claudia, Paulo Rocha e Eckhard Bick (2008). "Um mundo novo na Floresta Sintá (c) tica–o treebank do Português". Em: *Calidoscópio* 6.3, pp. 142–148.

Freitas, Cláudia et al. (2010a). "Second HAREM: Advancing the State of the Art of Named Entity Recognition in Portuguese". Em: *quot; In Nicoletta Calzolari; Khalid Choukri; Bente Maegaard; Joseph Mariani; Jan Odijk; Stelios Piperidis; Mike Rosner; Daniel Tapias (ed) Proceedings of the International Conference on Language Resources and Evaluation (LREC 2010)(Valletta 17-23 May de 2010) European Language Resources Association*. European Language Resources Association.

Freitas, Cláudia et al. (2010b). "Second HAREM: advancing the state of the art of named entity recognition in Portuguese". Em: *Proceedings of 7th International Conference on Language Resources and Evaluation*. LREC 2010. La Valleta, Malta: ELRA.

Gatt, Albert e Emiel Krahmer (2018). "Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation". Em: *Journal of Artificial Intelligence Research* 61, pp. 65–170.

Hajič, Jan et al. (2009). "The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages". Em: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pp. 1–18.

Hearst, Marti A (1992). "Automatic Acquisition of Hyponyms from Large Text Corpora". Em: *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, pp. 539–545.

Joulin, Armand et al. (2016). "Fasttext. zip: Compressing text classification models". Em: *arXiv preprint arXiv:1612.03651*.

Jurafsky, Dan e James H Martin (2009). *Speech & Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2nd. Pearson International Edition.

Kalady, Saidalavi, Ajeesh Elikkottil e Rajarshi Das (2010). "Natural Language Question Generation Using Syntax and Keywords". Em: *Proceedings of QG2010: The Third Workshop on Question Generation*. Vol. 2. questiongeneration. org, pp. 5–14.

Li, Xin e Dan Roth (2006). "Learning Question Classifiers: The Role of Semantic Information". Em: *Natural Language Engineering* 12.3, pp. 229–249.

Lopes, Fábio, César Teixeira e Hugo Gonçalo Oliveira (2019). "Named Entity Recognition in Portuguese Neurology Text using CRF". Em: *Proceedings of 19th EPIA Conference on Artificial Intelligence*, In press.

Maia, R et al. (2006). "An HMM-Based Brazilian Portuguese Speech Synthesizer and Its Characteristics". Em: *Journal of Communication and Information Systems* 21.2.

Manning, Christopher et al. (2014). "The Stanford CoreNLP Natural Language Processing Toolkit". Em: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55–60.

Nadeau, David e Satoshi Sekine (2007). "A survey of named entity recognition and classification". Em: *Lingvisticae Investigationes* 30.1, pp. 3–26.

Nadkarni, Prakash M, Lucila Ohno-Machado e Wendy W Chapman (2011). "Natural language processing: an introduction". Em: *Journal of the American Medical Informatics Association* 18.5, pp. 544–551.

Nothman, Joel, James R Curran e Tara Murphy (2008). "Transforming Wikipedia into Named Entity Training Data". Em: *Proceedings of the Australasian Language Technology Association Workshop 2008*, pp. 124–132.

Pantel, Patrick e Marco Pennacchiotti (2006). "Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations". Em: *Proceedings of the 21st Interna-*

tional Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, pp. 113–120.

Pinto, Alexandre, Hugo Gonçalo Oliveira e Ana Oliveira Alves (2016). "Comparing the Performance of Different NLP Toolkits in Formal and Social Media Text". Em: *OASIcs-OpenAccess Series in Informatics*. Vol. 51. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Pires, André Ricardo Oliveira (2017). "Named Entity Extraction from Portuguese Web Text". Tese de mestrado. Faculdade de Engenharia da Universidade do Porto.

Ranchhod, Ehsabete, Cristina Mota e Jorge Baptista (1999). "A computational lexicon of Portuguese for automatic text parsing". Em: *SIGLEX99: Standardizing Lexical Resources*.

Ratnaparkhi, Adwait (1996). "A Maximum Entropy Model for Part-of-Speech Tagging". Em: *Conference on Empirical Methods in Natural Language Processing*.

Ritter, Alan, Oren Etzioni, Sam Clark et al. (2012). "Open Domain Event Extraction from Twitter". Em: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 1104–1112.

Rodrigues, Ricardo, Hugo Gonçalo Oliveira e Paulo Gomes (2018). "NLPPort: A Pipeline for Portuguese NLP (Short Paper)". Em: *7th Symposium on Languages, Applications and Technologies (SLATE 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Santos, Diana (1992). "Natural Language and Knowledge Representation". Em: *Proceedings of the ERCIM Workshop on Theoretical and Experimental Aspects of Knowledge Representation*. Pisa, pp. 195–197.

Santos, Diana e Eckhard Bick (2000). "Providing Internet access to Portuguese corpora: the AC/DC project". Em: *Proceedings of 2nd International Conference on Language Resources and Evaluation*. LREC 2000, pp. 205–210.

Santos, Diana et al. (2006). "HAREM: An Advanced NER Evaluation Contest for Portuguese". Em: *Proceedings of 5th International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: ELRA.

Sha, Fei e Fernando Pereira (2003). "Shallow parsing with conditional random fields". Em: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, pp. 134–141.

Silveira, Natália (2008). "Towards a Framework for Question Generation". Em: *Proceedings of the Workshop on the Question Generation Shared Task and Evaluation Challenge*.

Snow, Rion, Daniel Jurafsky e Andrew Y Ng (2005). "Learning syntactic patterns for automatic hypernym discovery". Em: *Advances in neural information processing systems*, pp. 1297–1304.

Sripada, Somayajulu, Ehud Reiter e Ian Davy (2003). "SUMTIME-MOUSAM: Configurable Marine Weather Forecast Generator". Em: *Expert Update* 6.3, pp. 4–10.

Taba, Leonardo Sameshima et al. (2013). "Extração automática de relações semânticas a partir de textos escritos em português do Brasil". Em:

Xu, Kelvin et al. (2015). "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". Em: *International conference on machine learning*, pp. 2048–2057.

Yao, Xuchen e Yi Zhang (2010). "Question Generation with Minimal Recursion Semantics". Em: *Proceedings of QG2010: The Third Workshop on Question Generation*. Citeseer, pp. 68–75.

Zhou, Qingyu et al. (2017). "Neural Question Generation from Text: A Preliminary Study". Em: *National CCF Conference on Natural Language Processing and Chinese Computing*. Springer, pp. 662–671.