Ana Cláudia Fernandes Costa

Automatização de tarefas de Marketing

Ana Cláudia Fernandes Costa

Automatização de tarefas de Marketing

Relatório de Estágio apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Matemática e Aplicações, realizada sob a orientação científica do Doutor António Ferreira Pereira, Professor Auxiliar do Departamento de Matemática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Agostinho Miguel Mendes Agra
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Professora Doutora Maria Adelaide da Cruz Cerveira
Professora Auxiliar da Universidade de Trás-Os-Montes e Alto Douro

Professor Doutor António Ferreira Pereira
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Agradeço em primeiro lugar ao meu orientador António Pereira pois sem a ajuda dele este relatório não seria possível e em segundo lugar à ProdCent LDA por me ter acolhido para estágio. Por fim, gostaria de agradecer a todas as pessoas que acreditaram em mim e me deram de alguma forma apoio para eu terminar mais uma etapa da minha vida.

Palavras-chave Desenvolvimento guiado por testes, teste de software, tratamento de emails devolvidos, marketing, automatização, tratamento de dados

Resumo Ao vivermos num mundo cada vez mais tecnológico e competitivo é necessário inovar usando novas tecnologias. Este relatório de estágio retrata os testes associados à criação de uma nova ferramenta assim como a automatização de tarefas manuais através de programação.
Em particular são analisados vários algoritmos para determinar a similaridade entre strings (Rateliff e Obershelp, Damerau – Levenshtein, Jaro – Winkler) e estudada a sua aplicabilidade no contexto do problema de identificação de nomes similares numa lista de nomes de empresas.

keywords Test driven development, software testing, treatement of returned emails, marketing, automation, data processing

Abstract As we live in an increasingly technological and competitive world it is necessary to innovate using new technologies. This report describe the tests associated with creating a new tool as well as automation of manual tasks with programming.
In particular, several algorithms are analyzed to determine the similarity between strings (Rateliff and Obershelp, Damerau - Levenshtein, Jaro - Winkler) and studied their applicability in the context of the problem of identifying similar names in a list of company names.

Conteúdo

Conteúdo i

Lista de Figuras iii

Lista de Tabelas v

1 Introdução 1

2 Integração na equipa de testes 3

2.1 Desenvolvimento guiado por testes 4

2.2 Inserção rápida por email 6

2.3 Inserção de dados na base de dados de Marketing 11

2.4 Atualização de emails na base de dados de Marketing 14

3 Recolha de dados de empresas 25

3.1 Webscrapping 25

3.2 Pesquisa de dados em dois portais de empresas portuguesas 28

3.3 Algoritmos para a comparação de strings 29

3.3.1 Algoritmo de Rateliff e Obershelp 29

3.3.2 Algoritmo de Hamming 30

3.3.3 Distância de Levenshtein 30

3.3.4 Algoritmo de Damerau - Levenshtein 31

3.3.5 Algoritmo de Jaro 32

3.3.6 Algoritmo de Jaro - Winkler 33

3.4 Aplicação do algoritmo de Rateliff e Obershelp 34

3.5 Conclusões 36

4 Classificação automática de emails 39

5 Conclusão 47

Bibliografia 49

Lista de Figuras

2.1 Esquema dos passos do *Test Driven Development* 5
2.2 Fluxograma do FASTINSERT por email 7
2.3 Template usado para o FASTINSERT por email 9
2.4 Exemplo da criação de um utilizador através do FASTINSERT 10
2.5 Representação da base de dados 13

3.1 Representação dos conjuntos de empresas 36

4.1 Exemplo do ficheiro de input 40
4.2 Ilustração do Autómato 41

Lista de Tabelas

2.1 Exemplos de rotas de teste 3
2.2 Descrição dos erros devolvidos perante falhas 16
2.3 Comportamento de cada campo da tabela email 17
2.4 Comportamento das restantes tabelas 17
2.5 Comportamento do programa para o tratamento automático de emails na folha BLACKLISTED 18
2.6 Comportamento do programa para o tratamento automático de emails na folha ERROR 19
2.7 Comportamento do programa para o tratamento de emails automático na folha MISSPELLED 20
2.8 Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte I 21
2.9 Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte II 22
2.10 Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte III 23
2.11 Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte IV 24

3.1 Cálculo da distância de Levenshtein entre 'Sendim' e 'Serpins' 31
3.2 Resultados da aplicação do algoritmo Rateliff e Obershelp 37

Capítulo 1

Introdução

Este relatório surgiu do estágio proposto pela empresa ProdCent no âmbito do estágio final do Mestrado em Matemática e Aplicações.

A ProdCent é uma empresa situada na incubadora de empresas da universidade de Aveiro, em que os principais objetivos são a promoção internacional e a gestão da interação comercial entre empresas. Para isso é utilizado o portal (www.centroproduto.com) onde cada empresa pode expor os seus produtos / serviços.

Para assegurar a qualidade dos dados do portal é efetuada uma triagem das empresas que ficam públicas, ou seja, o perfil de uma empresa só é tornado visível ao público após ter passado por uma verificação dos dados presentes na 'página' da empresa. A missão da empresa é facilitar o relacionamento global entre empresas auxiliando os seus clientes a encontrar oportunidades de negócios a nível mundial, eliminando barreiras geográficas, cronológicas e linguísticas.

A equipa da empresa está dividida em três subequipas principais:

- a equipa de programadores que efetua toda a parte de programação relativa ao portal e suas funcionalidades;
- a equipa de testes que como o nome sugere efetua testes às novas funcionalidades / ferramentas antes de serem disponibilizadas ao utilizador final no portal de negócios;
- a equipa de *marketing* que trata da parte de promoção do portal, angariação de novos utilizadores e promoção das empresas existentes no portal através de campanhas de angariação de fornecedores ou clientes para as empresas.

O estágio teve como principal objetivo a otimização e eliminação de tarefas rotineiras realizadas pelos colaboradores da equipa de *marketing* através do uso de programação.

No próximo capítulo é retratada a integração na equipa de testes do portal, a abordagem usada pela empresa na documentação das funcionalidades implementadas no portal e alguns modelos de escrita de testes existentes.

No terceiro capítulo é descrita a forma como eram extraídos os dados de empresas de vários *websites* e a automatização da procura de número de identificação fiscal e de outros dados de empresas em dois *websites* com uma vasta base de dados de empresas portuguesas.

No quarto capítulo é relatada a forma pela qual foi feita a automatização da classificação de *emails* devolvidos derivados dos *emails* de *marketing* enviados pela empresa.

No quinto capítulo são apresentadas algumas conclusões do trabalho realizado neste estágio.

Capítulo 2

Integração na equipa de testes

O desenvolvimento de *software* é uma tarefa complexa e as empresas recorrem a vários métodos para desenvolver o seu *software*. No caso da ProdCent, o método utilizado consiste em descrever o mais detalhadamente possível o resultado esperado de uma funcionalidade antes de começar a escrever a parte do código correspondente. Após a escrita do comportamento da funcionalidade e sua implementação, são realizados testes antes da alteração (ou funcionalidade) chegar efetivamente ao portal, para assegurar o bom funcionamento do mesmo.

Para realizar esses testes são usadas 'cópias' do portal designadas por instâncias, nas quais se implementam as alterações a serem testadas. A equipa de testes usa essas instâncias e um conjunto de rotas para testar o desempenho das novas funcionalidades e as consequências das alterações efetuadas nos dados existentes e na introdução de novos dados.

Cada rota é um pequeno teste para a verificação de um acontecimento esperado ao fim de um conjunto de passos (ou ações). Na tabela 2.1 descrevem-se dois exemplos de rotas de testes que podem ser efetuadas quando um utilizador se regista.

Tabela 2.1: Exemplos de rotas de teste

id	Ação	Resultado esperado
1	- Aceder ao portal - Clicar em *Comece Já* - Preencher todos os campos obrigatórios - Clicar em *Comece já*	- Email enviado para o email inserido para confirmação do email.
2	- Aceder ao portal - Clicar em *Comece Já* - Deixar pelo menos um campo obrigatório por preencher - Clicar em *Comece já*	- Botão *Comece já* inativo - Alerta(s) de campo(s) obrigatório(s) não preenchido(s)

Imaginemos que a segunda rota não gera o resultado esperado e ao clicar no botão *Comece já* o utilizador '*email@domínio.pt*' seria registado sem o campo obrigatório do nome de utilizador preenchido. Neste caso a rota número 2 seria classificada como falha. Esta falha e os seus detalhes seriam anotados numa folha de excel juntamente com outros erros encontrados ao testar aquela instância. A folha de excel seria partilhada com a equipa de testes, a equipa da programação do portal e o CEO da empresa.

Após a realização dos testes seriam avaliados os erros encontrados e decidido quais necessitariam de uma correção urgente antes de realizar o *upgrade* para a instância seguinte. Em seguida, os elementos da equipa de programação corrigiriam os erros e voltar-se-ia a testar o que tinha sido classificado como erro para confirmar se estava corrigido. Só após ter sido verificado que todos os erros estavam corrigidos é que se efetuava a atualização para a instância seguinte.

A estratégia apresentada pela empresa para testar programas foi elaborar a documentação dos programas antes de começar a programação. Para fazer a documentação era necessário estudar quais os possíveis casos de entrada de dados (*inputs*) e qual seria a resposta do programa (*outputs*) a cada um desses *inputs*. Esta estratégia tem algo em comum com a abordagem de desenvolvimento guiado por testes, descrita na secção seguinte.

Durante o estágio, a abordagem sugerida pela empresa foi aplicada em três projetos: FASTINSERT (inserção rápida) por *email*; inserção de informação na base de dados de *marketing*; atualização de *emails* na base de dados de *marketing* através da interação do utilizador.

2.1 Desenvolvimento guiado por testes

A técnica de desenvolvimento de *software*, desenvolvimento guiado por testes, mais conhecida por *Test Driven Development* [2, 3], mais à frente designada por TDD foi criada por Kent Beck com o objetivo de criar testes que inspirem mais confiança, bem como encorajar à programação menos complexa.

Segundo Beck, a técnica TDD pode resumir-se a um processo repetitivo de 5 passos:

1. escrever um novo teste;
2. executar todos os casos de testes e verificar que o novo teste falhou;
3. escrever o mínimo de código que faça o teste passar;
4. refazer todos os testes e verificar que todos passam;
5. modificar o código de forma a ser mais fácil de compreender e eliminar código duplicado.

Na figura 2.1 pode-se observar o ciclo associado ao uso da técnica TDD.

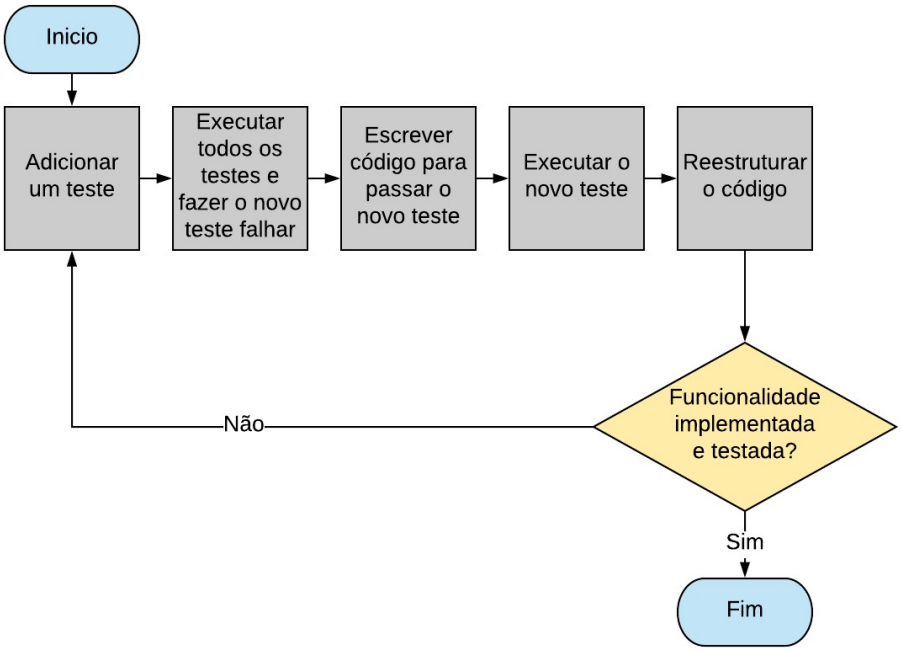


Figura 2.1: Esquema dos passos do *Test Driven Development*

À medida que mais partes do programa ficam prontas, mais testes são acrescentados, até que no final da programação se obtêm testes de regressão que asseguram uma maior segurança em futuras alterações ao *software*. Os testes de regressão são um conjunto de testes que servem para assegurar que a parte do *software* anteriormente implementada não regrediu e ficou com erros.

As maiores vantagens de criar estes testes de regressão são poderem ser executados automaticamente e permitir verificar se as alterações efetuadas fizeram de alguma forma o programa regredir, quando algum dos testes falha. Embora a técnica TDD forneça um teste de regressão do *software*, isto não significa que baste para cobrir todos os casos possíveis. Por exemplo, através de programação, não é possível testar código que tenha uma dependência de um acontecimento aleatório.

Um dos maiores benefícios ao aplicar esta técnica é ter a programação toda documentada. Uma outra vantagem da técnica TDD é que, por ser necessário escrever os testes em primeiro lugar, o programador terá de saber diferenciar entre a funcionalidade que tem de implementar e o teste para o qual a implementação não pode falhar. Por exemplo, uma funcionalidade poderia ser dividir dois números e um teste pelo qual tem de passar é não permitir a divisão $\frac{a}{b}$ com $a \in \mathbb{N}$ e $b = 0$. Na secção seguinte, é a apresentada a funcionalidade de criar utilizadores e empresas através de *email* e um teste seria não deixar o utilizador criar catálogos sem ter uma língua associada.

De acordo com um estudo [13] efetuado na universidade de Reykjavík, os programadores referiram que usando esta técnica foi mais fácil detetar falhas inseridas após uma alteração

ao sistema, o que leva a uma maior qualidade do código e a um menor tempo a fazer *debug*. Uma desvantagem da aplicação desta técnica é o aumento do tempo de desenvolvimento de uma funcionalidade devido ao facto de ser necessário escrever todos os testes.

2.2 Inserção rápida por email

A inserção rápida por *email* (FASTINSERT) era uma funcionalidade que já existia no portal com o objetivo de a equipa de *marketing* criar utilizadores e empresas no portal centroproduto.com, auxiliando dessa forma as empresas a registarem-se no portal de negócios. Só eram registadas desta forma empresas que pedissem ajuda à ProdCent para efetuar o registo no portal. Normalmente este pedido de registo era efetuado por *email*. Esta função só estava disponível para os utilizadores comerciais que tinham mais permissões que um utilizador normal.

O processo de inserção de uma empresa no portal através da equipa de *marketing* passava por: inserir os dados do utilizador e os dados da empresa; associar a sua conta à empresa criada para poder criar a primeira publicação (*post*) de divulgação da empresa no portal; colocar empresa criada em contacto com a empresa beneficiária da campanha por mensagem privada no portal; e, por vezes, também era necessário inserir linhas de catálogo e produtos na página da empresa criada.

Este processo obrigava a muitos cliques ao navegar entre vários ecrãs, sendo um deles o ecrã de *emails* que continha as informações fornecidas pelas empresas. Por forma a aliviar o esforço exigido à equipa de *marketing* imposto através do processo originalmente implementado, surgiu a ideia de realizar a tarefa de inserção rápida por *email*.

A programação desta nova funcionalidade foi dividida em interpretação e extração da mensagem do *email* e execução de ações de acordo com os conteúdos extraídos. Na parte da interpretação e extração a programação seguiu o procedimento ilustrado na Figura 2.2.

Neste projeto participei no desenvolvimento da documentação do programa na parte da interpretação do *email*, tendo elaborado a documentação relativa aos erros de sintaxe devolvidos ao utilizador.

Primeiramente, foi necessário definir: quais as ações que seriam possíveis de realizar, quais os campos que estariam disponíveis para serem preenchidos através do *email* e como é que os dados seriam inseridos no *email* (qual a estrutura do *email*).

As ações possíveis de realizar eram a criação de utilizador, empresa, publicações, itens, linhas de catálogo e a personalização da mensagem entre a nova empresa e a empresa que beneficiava da campanha. Uma campanha no portal é uma ação efetuada através de *posts* no portal ou através de *emails* com o objetivo de angariar empresas para o portal. O tema dos *emails* e *posts* costumava ser ou procura de fornecedores e clientes para uma dada empresa ou *posts* alusivos ao facto da ProdCent ter contribuído para melhorar o negócio de outra

empresa.

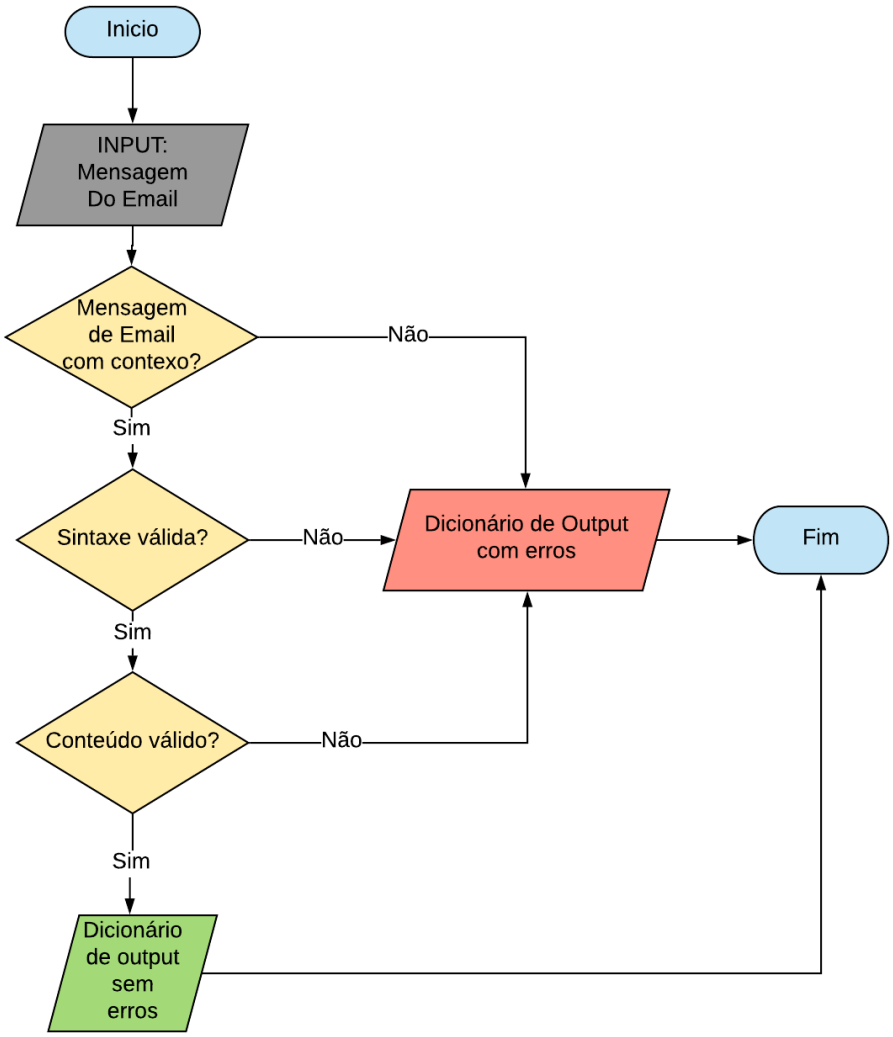


Figura 2.2: Fluxograma do FASTINSERT por email

A abordagem usada no processo de escrita do comportamento deste programa foi semelhante à abordagem de escrever testes de Cockburn em 'Writing effective use cases' [1].

Segundo Cockburn para se poder escrever casos de testes efetivos é necessário seguir os seguintes passos:

1. definir os objetivos e comportamento do sistema;

2. definir os cenários de maior sucesso para os objetivos definidos no passo anterior e garantir que o sistema está de acordo com a finalidade para a qual o utilizador final quer usar o programa;

3. completar os cenários criados anteriormente com todas as possíveis falhas que podem acontecer antes de escrever como o sistema deve lidar com elas;

4. escrever como é suposto o sistema lidar com cada falha. Por vezes, nesta parte é onde se descobre um novo objetivo que deve ser suportado pelo sistema e caso isso aconteça deve-se voltar ao ponto 1 e rever também os casos escritos até este passo.

Nesta funcionalidade os objetivos eram que a partir de uma mensagem de um *email* enviado para um endereço predefinido fosse possível: criar utilizadores, empresas, *posts*, itens, catálogos e escolher a mensagem que colocaria em contacto as duas empresas. O comportamento do sistema era receber a mensagem do *email* com o formato presente na Figura 2.3 e devolver um dicionário com a informação recolhida.

Neste programa existiam 6 cenários de maior sucesso:

1. criar utilizador;

2. criar utilizador e empresa;

3. criar utilizador, empresa e *post* (publicação no portal);

4. criar utilizador, empresa e item;

5. criar utilizador, empresa e catálogo;

6. criar utilizador, empresa e definir a mensagem partilhada entre a empresa dona da campanha e a empresa registada.

Para criar o utilizador era necessário criar um *email* com o assunto 'FASTINSERT()' e com o modelo apresentado na Figura 2.3, com o campo *operations* preenchido com 'USER' e os campos do *user* e o campo *campaing.id* preenchidos com os dados referentes ao novo utilizador e o campo *campaign.id* com a campanha em que está interessado.

Para criar:

- utilizador e empresa preenchiam-se os campos referentes ao *user* (utilizador), ao *actor* (empresa), os campos do *campaign.id* e o *operations* com 'USER, ACTOR';

- utilizador, empresa e *post* era necessário preencher o *campaign.id*, todos os campos referentes ao *user*, *actor* e *post* e preencher o campo *operations* com 'USER, ACTOR, POST'

- utilizador, empresa e item era necessário preencher o *campaign.id*, todos os campos referentes ao *user*, *actor* e *item* e preencher o campo *operations* com 'USER, ACTOR, ITEM';

- utilizador, empresas e catálogo era necessário preencher o *campaign.id*, todos os campos referentes ao *user*, *actor* e *catalog* (catálogo) e preencher o campo *operations* com 'USER, ACTOR, CATALOG';
- utilizador, empresa e definir a mensagem partilhada entre a empresa dona da campanha e a empresa registada era necessário preencher o *campaign.id*, todos campos referentes ao *user*, *actor* e *shared_message* (mensagem partilhada) e preencher o campo *operations* com 'USER, ACTOR, CAMPRESP'.

```
<email_content>
<operations>   </operations>
<campaign.id>  </campaign.id>

<user.first_name>   </user.first_name>
<user.last_name>   </user.last_name>
<user.email>  </user.email>
<user.language>   </user.language>
<user.dial_code>   </user.dial_code>
<user.phone>  </user.phone>

<actor.actor_legal_name>  </actor.actor_legal_name>
<actor.actor_short_name>  </actor.actor_short_name>
<actor.country>  </actor.country>
<actor.nif>  </actor.nif>
<actor.business_simple_code>  </actor.business_simple_code>
<actor.cep>  </actor.cep>
<actor.email>  </actor.email>

<catalog>
<catalog.description> </catalog.description>
<catalog.lang>  </catalog.lang>
<catalog.name>  </catalog.name>
</catalog>

<item>
<item.lang>   </item.lang>
<item.usage>  </item.usage>
<item.category>   </item.category>
</item>

<post.title>  </post.title>
<post.subtitle>  </post.subtitle>
<post.text>  </post.text>
<post.keyword>  </post.keyword>

<shared_message.briefuser>  </shared_message.briefuser>
</email_content>

XXXXXX
```

Figura 2.3: Template usado para o FASTINSERT por email

Por exemplo para criar um utilizador para a campanha número 5 era necessário enviar um *email* com a mensagem presente na Figura 2.4.

```
<email_content>
<operations>USER</operations>
<campaing.id>5</campaing.id>
<user.first_name>Ana</user.first_name>
<user.last_name>Costa</user.last_name>
<user.emaile>ana.costa@dominio.cc</user.email>
<user.language>pt</user.language>
<user.dial_code>PT</user.dial_code>
<user.phone>960000000</user.phone>
</email_content>

REENCAMINHAMENTO DO EMAIL ENVIADO PELO
UTILIZADOR DANDO AUTORIZAÇÃO DO SEU REGISTO
```

Figura 2.4: Exemplo da criação de um utilizador através do FASTINSERT

As falhas foram divididas em 3 tipos: as falhas de sintaxe, as falhas de requerimentos e as falhas de validação. Nas falhas de sintaxe incluem-se:

- não escrever uma ou as duas tags delimitadoras da informação (email_content);
- escrever corretamente as tags delimitadoras mas eliminar o início ou o fim de uma outra tag, por exemplo eliminar '<user.first_name>' ou '</user.first_name>';
- inserir '/' numa tag de início;
- eliminar '/' numa tag de fim;
- eliminar o conteúdo e uma tag de início ou numa tag de fim, por exemplo ao invés de ter '<user.first_name>' teríamos '<>';
- modificar o conteúdo de uma tag de início ou de uma tag de fim;
- eliminar o '<' ou '>' de uma tag.

Eram consideradas falhas de requisitos sempre que estavam ausentes pelo menos uma das seguintes tags:

- user.email, user.language quando USER estiver na tag operations;
- actor.actor_legal_name, actor.actor_short_name, actor.business_simple_code, actor.country quando ACTOR estiver na tag operations;
- post.text quando POST estiver na tag operations;
- catalog.name quando CATALOG estiver na tag operations;
- item.usage quando ITEM estiver na tag operations.

Algumas das falhas de validação ocorriam quando uma das seguintes condições não era satisfeita:

- user.first_name, user.last_name tinham de ter entre 0 a 30 carateres;
- user.email tinha de ter entre 6 a 75 carateres e passar por uma função já existente na empresa que o validava como *email*;
- user.language tinha de ter dois carateres e pertencer à lista de línguas disponíveis no portal;
- user.dial_code preenchido se e só se user.phone preenchido;
- user.dial_code tinha de ter dois carateres e pertencer à lista de indicativos de países (dial codes) reconhecidos pelo portal;
- actor.actor_legal_name entre 1 e 255 carateres;
- actor.actor_short_name entre 1 e 50 carateres;
- actor.country tinha de ter 2 carateres e pertencer à lista de países cujas empresas eram autorizadas a registar-se no portal;
- actor.nif válido para o país (actor.country) escolhido e ainda não pertencer a nenhuma empresa já registada no portal;
- actor.business_simple_code não era válido no país definido no actor.country;
- inserir actor.country se inserir actor.cep;
- actor.cep ser válido.

Dependendo da falha era devolvido um *email* com o erro correspondente e não era criado nada no portal. O erro descritivo de cada falha poderá observar-se na Tabela 2.2 onde os erros são devolvidos em inglês a pedido da empresa. Ao ser devolvido um *email* com a descrição do erro para o utilizador, este corrigia a mensagem do *email* de modo a que não houvesse o erro e reenviava o *email* corrigido. Caso o *email* enviado não tivesse erros os dados eram criados no portal e era enviado um *email* a informar o utilizador que os dados tinham sido inseridos no portal.

2.3 Inserção de dados na base de dados de Marketing

Com o objetivo de obter mais empresas inscritas no portal através de envio de *emails* de *marketing* eram extraídas informações sobre empresas de diversos portais. Após extrair as informações era necessário reunir todos esses dados numa base de dados. Inicialmente existia

um programa que inseria os dados de cada endereço de *email*, caso não estivesse na base de dados. Nesse programa, o maior problema era que ao fim de carregar a informação referente a um endereço de *email* de uma empresa havia muita dificuldade de atualizar e acrescentar informação relacionada com esse *email*.

Por exemplo, inseríamos um endereço de *email* 'nome@empresa.com' e o nome da empresa na base de dados. Depois encontrávamos num outro portal o mesmo endereço de *email* ('nome@empresa.com') com o nome, o nif e o cep. No programa inicial não havia forma de inserir estes novos dados por o endereço já estar na base de dados.

Os dados referentes a cada *email* que poderiam ser guardados eram:

- nome legal da empresa - actor_legal_name;
- país – country;
- código empresarial postal – cep;
- nif ou vat - simple_nif;
- international standard classification of industry – isic;
- identificador usado para identificar a atividade económica – identifier;
- código de atividade económica - simple_code;
- nome do utilizador do endereço de *email* caso não seja o *email* geral - user_name;
- portal de onde foi retirada a informação – *origin*;
- palavras chave sobre a atividade da empresa – *keywords*;
- responsabilidade associada ao dono do endereço de *email* (por exemplo pode ser CEO, compras, vendas) - *responsability*;
- número de telemóvel ou telefone – *mobile*.

A base de dados seguia a estrutura representada na Figura 2.5.

No caso deste programa, a forma usada para escrever o comportamento esperado passou por basear o comportamento nos possíveis *inputs* (ficheiro de excel) e os dados existentes na base de dados. Primeiramente, estipulou-se que só se atualizaria um campo na base de dados caso o mesmo estivesse vazio. No caso de campos da tabela *email* o programa tinha o comportamento descrito na Tabela 2.3.

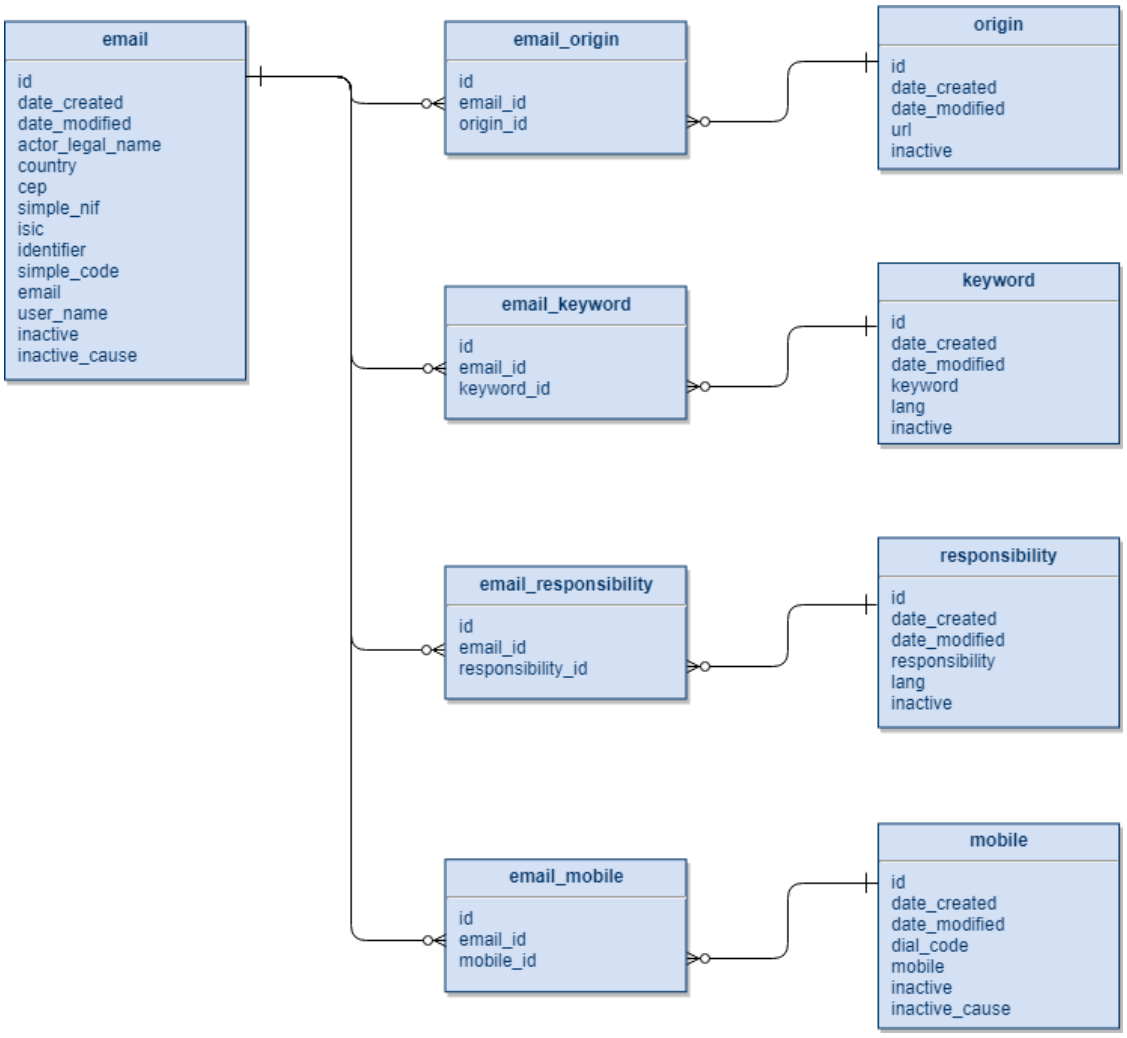


Figura 2.5: Representação da base de dados

Para as restantes tabelas, *keywords*, *origin*, *responsability* e *mobile*, o comportamento do programa seguia o descrito na tabela 2.4.

Esta foi a primeira abordagem para atualizar registos na base de dados, como melhoria ao que se usava anteriormente que permitia apenas inserir dados de endereços de *email* novos.

Como medida de segurança o programa não inseria ou alterava nada na base de dados caso houvesse algum erro não previsto.

Após alguns usos do programa detetou-se que alguns endereços de *emails* não conseguiam ser atualizados. Devido ao facto de existirem endereços repetidos na base de dados, o programa não conseguia escolher qual dos endereços era para atualizar. O programa só estava preparado para lidar com o caso de encontrar no máximo um registo do endereço a inserir na base de dados. Foi corrigido esse problema fazendo uma junção da informação dos endereços de *email* repetidos num só registo de *email*.

Também foi alterado o modo de como se armazenava as palavras chave associadas a uma empresa. Inicialmente era armazenado na variável *keyword* um texto com a descrição da empresa, por exemplo 'A sociedade tem por objecto a gestão de actividades turísticas; actividades de *marketing* e promoção de serviços para convenções e (...)', o que limitava a informação que era armazenada para cada empresa. Para conseguir-se armazenar mais informação, foi separado o texto em palavras (só as com mais de 2 carateres), guardadas na tabela *keywords* e associadas aos respetivos endereços de *email*. Para cada texto inserido no campo *keywords* no ficheiro era necessário o tratamento descrito no Algoritmo 1.

Algoritmo 1

1: O texto é separado por espaços; vírgulas; ponto e vírgulas; '/'; ponto; subtraço e guardado numa lista de palavras.
2: Para cada palavra:
   a. Remove-se a acentuação
   b. Substitui-se todos os carateres que não são letras ou números por espaços e insere-se cada palavra na base de dados.

Cada palavra extraída do texto era inserida na tabela *keywords* caso ainda não existisse nessa tabela e era associada ao endereço de *email* caso ainda não estivesse associada.

O processo para as tabelas *origin*, *responsability* e *mobile* ficou análogo ao processo efetuado para a tabela *keywords* excepto o tratamento do texto.

Uma melhoria necessária era que quando o programa encontrasse entradas repetidas nas tabelas *keyword*, *responsability*, *origin* ou *mobile* associasse o endereço de *email* a uma delas ao invés de retornar erro e não inserir nenhuma informação relativa ao *email*.

2.4 Atualização de emails na base de dados de Marketing

Devido à crescente devolução de *emails* por falha de entrega era necessário 'eliminar' esses endereços na base de dados para diminuir a quantidade de *emails* devolvidos e também para não gastar a quota de *emails* a enviar com *emails* que não chegaram ao destino. Para esses endereços de *email* não voltarem a ser inseridos na base de dados ao invés de eliminar os endereços, os registos eram colocados como inativos.

Para isso pegou-se no resultado do programa mencionado mais à frente no capítulo 4 e criou-se um programa para ajudar o utilizador a inativar os *emails* blacklisted e a decidir o que fazer com os *emails* em que não se identificava o tipo de erros que eram resposta automática e os que provavelmente estavam mal escritos.

Para ser mais fácil a utilização, este programa é composto por uma interface gráfica onde existia uma aba para colocar os *emails* a serem inativos de forma automática (classificados como blacklist) e outra aba onde o utilizador tinha de escolher manualmente o que fazer (error, reply, auto-reply e misspelled).

Este programa centrou-se nas variáveis *email_old* e *email_new* que seriam os endereços de email antigos que se pretendia ou não desativar e os *emails* novos a inserir na base de dados com os dados antigos. Para cada decisão era avaliado se os endereços de email estavam na base de dados e se estavam inativos. Caso o *email_old* estivesse inativo e aparecesse na lista para desativar, o registo do *email_old* não era alterado na base de dados. Caso o *email_novo* estivesse na base de dados, eram adicionadas ao registo das informações do *email_new* as informações que faltassem do *email_old*.

Para a aba automática o programa tinha o comportamento descrito na tabela 2.5, ou seja, o *email* só era colocado inativo se fosse único e estivesse inativo na base de dados.

Para os *emails* classificados como erro teriamos as opções: eliminar e ignorar. O comportamento do programa nesta aba seguia como a Tabela 2.6.

Para os *emails* classificados como misspelled teríamos a opções: eliminar, renomear e ignorar. O comportamento do programa para esta aba seguia como a Tabela 2.7.

Para os *emails* classificados como *reply* e *auto-reply* tínhamos as opções: atualizar e inativar; atualizar e manter activo; remover; guardar para mais tarde; ignorar. O comportamento do programa para estes casos está descrito nas tabelas 2.8 a 2.11.

Como em alguns casos ao inativar um *email* ficar-se-ia sem um endereço de *email* para uma dada empresa, efetuou-se um pequeno estudo sobre quais os endereços de *email* mais comuns nas empresas portuguesas. Nesse estudo verificou-se que na maior parte das empresas portuguesas o endereço de *email* era 'geral@*domínio.país*' ou 'info@*domínio.país*' ou '*domínio*@*domínio.país*'. Uma melhoria a este programa seria gerar automaticamente novos endereços de *email* para substituir os que foram inativados por serem blacklist, pegando no que estivesse à esquerda do domínio do endereço de *email* e substituir por '*domínio*', 'geral' ou por 'info', já que são os mais comuns entre empresas que tem domínios próprios para o endereço de *email*.

Para esta automatização seria necessário saber como detetar que um *email* pertencia a um domínio de uma empresa e não a domínio público como por exemplo '*gmail.com*'. Algumas empresas por não terem um domínio próprio usam servidores de *email* públicos como o gmail mas colocam o nome da empresa do lado esquerdo do '@'. Caso esses endereços não fossem filtrados, os casos em que o *email* acaba, por exemplo, em '@gmail.com' poderíamos reescrever o *email* '*empresa*@gmail.com' (que iria ser inactivado) por 'geral@gmail.com'.

Tabela 2.2: Descrição dos erros devolvidos perante falhas

Falha	Erro devolvido
Não escrever a tag delimitadoras de início da informação (email_content)	COMPILATION_slicing: Could not find starting tag '<email_content>'
Não escrever a tag delimitadoras de fim da informação (email_content)	COMPILATION_slicing: Could not find ending tag '</email_content>'
Escrever corretamente as tags delimitadoras mas eliminar o início de uma outra tag, por exemplo eliminar '<user.first_name>'	COMPILATION_heap: FAILURE: Trying to close with tag <*ending tag*> without any previously opened tag
Escrever corretamente as tags delimitadoras, mas não escrever a tag 'operations' ou deixar essa tag vazia	operations: Missing required tag 'operations'
Escrever corretamente as tags delimitadoras e escrever um conteúdo inválido na tag 'operations'	Erro com as operações em falta para a tag ser válida
Escrever corretamente as tags delimitadoras, mas não escrever a tag 'campaing.id' ou deixar essa tag vazia	campaign.id: Missing required tag 'campaign.id'
Escrever corretamente as tags delimitadoras mas eliminar o fim de uma outra tag	COMPILATION_heap: Tag <*starting tag*> does not belong inside <*next starting tag*>
Inserir '/' numa tag de início	COMPILATION_heap: FAILURE: Trying to close with tag <*changed tag*> without any previously opened tag
Eliminar '/' numa tag de fim	COMPILATION_heap: Tag <*next starting tag*> does not belong inside <*changed ending tag*>
Eliminar o conteúdo e uma tag de início ou numa tag de fim	COMPILATION_heap: FAILURE: Trying to close with tag <*next ending tag*> without any previously opened tag
Modificar o conteúdo de uma tag de início ou de uma tag de fim	COMPILATION_heap: Tag <*next starting tag*> does not belong inside <*previous starting tag*>
Eliminar o '<' ou '>' de uma tag	COMPILATION_heap: FAILURE: Trying to close with tag <*next ending tag*> which is different from the last opened tag <*starting tag changed*>
Não inserir user.email	Missing required tag 'user.email'

Tabela 2.3: Comportamento de cada campo da tabela email

Dados a inserir?	Base de Dados?	Ação
Sim	Não	Atualizar campo e atualizar data de modificação (date_modified)
Sim	Sim	Não atualizar campo
Não	Não	Não atualizar campo
Não	Sim	Não atualizar campo

Tabela 2.4: Comportamento das restantes tabelas

Dados a inserir?	Tem uma ligação com a tabela email?	Existe na base de dados?	Ações
Sim	Sim	Sim/não	Não atualiza nenhuma tabela
Sim	Não	Sim	Cria a ligação na tabela email_campo
Sim	Não	Não	Cria uma entrada na tabela correspondente ao campo; criar ligação na tabela email
Não	Sim/não	Sim/não	Não atualiza nenhuma tabela

Tabela 2.5: Comportamento do programa para o tratamento automático de emails na folha BLACKLISTED

Endereço preenchido?	Endereço antigo está na BD?	É único?	Endereço antigo está inativo?
NÃO	N/A	N/A	N/A
Reação: Escrever erro na folha de output do XLSX			
SIM	NÃO	N/A	N/A
Reação: Escrever erro na folha de output do XLSX			
SIM	SIM	NÃO	N/A
Reação: Escrever erro na folha de output do XLSX			
SIM	SIM	SIM	NÃO
Reação: Atualizar email.inactive para True, email.inactive_cause para 'BOUNCED error description' e email.date_modified para a data atual.			
SIM	SIM	SIM	SIM
Reação: Não modificar nenhuma tabela na base de dados			

Tabela 2.6: Comportamento do programa para o tratamento automático de emails na folha ERROR

Endereço preenchido no XLSX?	Endereço antigo está na BD?	É único na BD?	Endereço antigo está ativo na BD?	Dar escolha ao utilizador
NÃO	N/A	N/A	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	NÃO	N/A	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	SIM	NÃO	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	SIM	SIM	NÃO	SIM
Escolha do utilizador: Eliminar				
Reação: Atualizar email.inactive para True e email.inactive_cause para 'BOUNCED error description'				
SIM	SIM	SIM	NÃO	SIM
Escolha do utilizador: Ignorar				
Reação: Não atualizar nenhuma tabela				
SIM	SIM	SIM	SIM	NÃO
Escolha do utilizador: N/A				
Reação: Não atualizar nenhuma tabela				

Tabela 2.7: Comportamento do programa para o tratamento de emails automático na folha MISSPELLED

Endereço preenchido no XLSX?	Endereço está na BD?	É único na BD?	Endereço está ativo na BD?	Dar escolha ao utilizador
NÃO	N/A	N/A	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	NÃO	N/A	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	SIM	NÃO	N/A	NÃO
Escolha do utilizador: N/A				
Reação: Escrever erro na folha de output do XLSX				
SIM	SIM	SIM	NÃO	SIM
Escolha do utilizador: Eliminar				
Reação: Atualizar email.inactive para True e email.inactive_cause para 'BOUNCED error description '				
SIM	SIM	SIM	NÃO	SIM
Escolha do utilizador: Escrever um novo email e alterar o antigo				
Reação: Atualizar email.inactive para True e email.inactive_cause para 'BOUNCED MISSPELLED'; criar/atualizar o email_written com a informação extraída do email antigo				
SIM	SIM	SIM	NÃO	SIM
Escolha do utilizador: Ignorar				
Reação: Não atualizar nenhuma tabela				
SIM	SIM	SIM	SIM	NÃO
Escolha do utilizador: N/A				
Reação: Não atualizar nenhuma tabela				

Tabela 2.8: Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte I

endereço antigo no XLSX?	novo endereço no XSLX?	endereço antigo na base de dados?	endereço antigo é único?	endereço antigo está inativo?	novo endereço na base de dados?	email novo está inativo?
Sim	Não	Não	N/A	N/A	N/A	N/A
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						
Sim	Não	Sim	Não	N/A	N/A	N/A
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						
Sim	Não	Sim	Sim	Não	N/A	N/A
Escolha do utilizador: UPDATE AND REMOVE OLD or UPDATE AND KEEP OLD						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'UPDATE'						
Sim	Não	Sim	Sim	Não	N/A	N/A
Escolha do utilizador: REMOVE						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'REMOVE'						
Sim	Não	Sim	Sim	Não	N/A	N/A
Escolha do utilizador: IGNORE						
Reação: Não altera a base de dados.						
Sim	Não	Sim	Sim	Não	N/A	N/A
Escolha do utilizador: SAVE THIS EMAIL FOR LATER						
Reação: Salva o corpo da mensagem, o endereço antigo e o novo endereço numa folha de cálculo para ver mais tarde.						
Sim	Não	Sim	Sim	Sim	N/A	N/A
Escolha do utilizador: N/A						
Reação: Não altera a base de dados.						
Sim	Sim	Não	N/A	N/A	N/A	N/A
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						
Sim	Sim	Sim	Não	N/A	N/A	N/A
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						
Sim	Sim	Sim	Sim	Não	Não	N/A
Escolha do utilizador: UPDATE AND KEEP OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						

Tabela 2.9: Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte II

endereço antigo no XLSX?	novo endereço no XSLX?	endereço antigo na base de dados?	endereço antigo é único?	endereço antigo está inativo?	novo endereço na base de dados?	email novo está inativo?
Sim	Sim	Sim	Sim	Não	Não	N/A
Escolha do utilizador: UPDATE AND REMOVE OLD						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'UPDATE'; extrai a informação do endereço antigo e cria o novo endereço com a informação extraída do antigo.						
Sim	Sim	Sim	Sim	Não	Não	N/A
Escolha do utilizador: REMOVE						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'REMOVE'						
Sim	Sim	Sim	Sim	Não	Não	N/A
Escolha do utilizador: IGNORE						
Reação: Não altera a base de dados.						
Sim	Sim	Sim	Sim	Não	Não	N/A
Escolha do utilizador: SAVE THIS EMAIL FOR LATER						
Reação: Salva o corpo da mensagem, o endereço antigo e o novo endereço numa folha de cálculo para ver mais tarde.						
Sim	Sim	Sim	Sim	Não	Sim	Não
Escolha do utilizador: UPDATE AND KEEP OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						
Sim	Sim	Sim	Sim	Não	Sim	Não
Escolha do utilizador: UPDATE AND REMOVE OLD						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'UPDATE'; extrai a informação do endereço antigo e atualiza o novo endereço com a informação extraída do endereço antigo						
Sim	Sim	Sim	Sim	Não	Sim	Não
Escolha do utilizador: REMOVE						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'REMOVE'						
Sim	Sim	Sim	Sim	Não	Sim	Não
Escolha do utilizador: IGNORE						
Reação: Não altera a base de dados.						
Sim	Sim	Sim	Sim	Não	Sim	Não
Escolha do utilizador: SAVE THIS EMAIL FOR LATER						
Reação: Salva o corpo da mensagem, o endereço antigo e o novo endereço numa folha de cálculo para ver mais tarde.						

Tabela 2.10: Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte III

endereço antigo no XLSX?	novo endereço no XSLX?	endereço antigo na base de dados?	endereço antigo é único?	endereço antigo está inativo?	novo endereço na base de dados?	email novo está inativo?
Sim	Sim	Sim	Sim	Não	Sim	Sim
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						
Sim	Sim	Sim	Sim	Sim	Não	N/A
Escolha do utilizador: UPDATE AND KEEP OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						
Sim	Sim	Sim	Sim	Sim	Não	N/A
Escolha do utilizador: UPDATE AND REMOVE OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						
Sim	Sim	Sim	Sim	Sim	Não	N/A
Escolha do utilizador: REMOVE						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'REMOVE'						
Sim	Sim	Sim	Sim	Sim	Não	N/A
Escolha do utilizador: IGNORE						
Reação: Não altera a base de dados.						
Sim	Sim	Sim	Sim	Sim	Não	N/A
Escolha do utilizador: SAVE THIS EMAIL FOR LATER						
Reação: Salva o corpo da mensagem, o endereço antigo e o novo endereço numa folha de cálculo para ver mais tarde.						
Sim	Sim	Sim	Sim	Sim	Sim	Não
Escolha do utilizador: UPDATE AND KEEP OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						
Sim	Sim	Sim	Sim	Sim	Sim	Não
Escolha do utilizador: UPDATE AND REMOVE OLD						
Reação: Extrai a informação do endereço antigo e cria o endereço novo com a informação extraída do antigo						
Sim	Sim	Sim	Sim	Sim	Sim	Não
Escolha do utilizador: REMOVE						
Reação: Atualiza o campo inactive do endereço antigo para Verdadeiro e o campo inactive_cause para 'REMOVE'						
Sim	Sim	Sim	Sim	Sim	Sim	Não
Escolha do utilizador: IGNORE						
Reação: Não altera a base de dados.						

Tabela 2.11: Comportamento do programa para o tratamento de emails automático na folha REPLIES and AUTO REPLIES parte IV

endereço antigo no XLSX?	novo endereço no XSLX?	endereço antigo na base de dados?	endereço antigo é único?	endereço antigo está inativo?	novo endereço na base de dados?	email novo está inativo?
Sim	Sim	Sim	Sim	Sim	Sim	Não
Escolha do utilizador: SAVE THIS EMAIL FOR LATER						
Reação: Salva o corpo da mensagem, o endereço antigo e o novo endereço numa folha de cálculo para ver mais tarde.						
Sim	Sim	Sim	Sim	Sim	Sim	Sim
Escolha do utilizador: N/A						
Reação: Escrever erro no XLSX e não altera a base de dados.						

Capítulo 3

Recolha de dados de empresas

Com o objetivo de aumentar a base de dados de *marketing*, eram extraídas informações de diversos portais que contivessem endereços de *email* e mais alguns dados de empresas, como por exemplo o nome e o tipo de atividade da empresa. Em cada portal a extração de dados era dividida em duas ou três partes, dependendo da nacionalidade das empresas:

1. selecionar os *urls* que continham os dados das empresas presentes no portal;
2. extrair a informação de cada empresa contida em cada *url* selecionado na primeira parte;
3. quando a empresa era portuguesa, pesquisar a informação em falta no portal (por exemplo o código de atividade da empresa) noutros dois *websites* de empresas portuguesas para completar a informação já extraída.

As duas primeiras partes de extração de dados de um portal são retratadas na secção seguinte.

3.1 Webscrapping

Em cada portal foram primeiramente selecionados os *urls* onde estariam as informações de cada empresa. Posteriormente foi efetuada a extração das informações de cada empresa percorrendo cada *url* de um dado portal. Estas informações eram guardadas num ficheiro de Excel. No caso de existirem dados de mais do que uma empresa num dado *url*, era necessário isolar o HTML que continha a informação de cada uma para depois conseguir extrair a informação separadamente.

Com o objetivo de obter mais facilmente o *url* referente a cada empresa, era primeiramente analisada a estrutura do website. Por exemplo, um dado portal disponibilizava uma página com uma lista de todas as empresas registadas, na qual era possível filtrar as que possuíam endereço de *email*. Essa página foi encontrada após analisar o mapa do portal (*sitemap*). A utilização deste filtro permitiu reduzir o número de resultados de 479 181 empresas para 126

209 empresas com endereços de *email*, tornando o processo de extração de informação mais eficiente ao remover dos resultados empresas sem interesse para a base de dados da ProdCent, ou seja, empresas sem endereço de *email*.

As informações recolhidas eram:

- *email*;
- nome legal da empresa;
- palavras chave;
- país;
- telemóvel ou telefone;
- nome do utilizador do *email*;
- cargo ocupado pelo utilizador do *email*;
- código de atividade económica (cae);
- nif (número de identificação fiscal) ou vat (*value-added tax*);
- código de endereço postal (cep).

Para extrair os dados dos portais foram usadas usadas as bibliotecas *requests*[16], *selenium*[15] e *BeautifulSoup*[17]. As primeiras duas bibliotecas serviam para aceder ao HTML da página enquanto a terceira servia para navegar no HTML da página.

A biblioteca *requests* serve para fazer pedidos a um servidor com objetivo de obter uma resposta. Fazem parte dessa resposta o cabeçalho, o HTML e a codificação da página web. Para a extração dos dados, a parte da resposta que interessava era o HTML da página.

Através do HTML conseguia-se ter acesso a todo o texto contido na página, salvo raras exceções em que existia texto oculto através do uso de *javascript*. O *javascript* era utilizado com a finalidade de proteger algumas informações e obrigar o utilizador a clicar para poder ter acesso às mesmas. Nesses casos era usada a biblioteca *Selenium* para extrair a informação da página.

As tags (nomes de elementos XML) relativas ao website da empresa ou endereço de *email* possuíam na maior parte dos casos o seguinte formato:

‘Website’.

A biblioteca *Selenium* serve para aceder todas as funcionalidades de um navegador comum, incluindo simular o comportamento humano ao utilizar um website. O navegador utilizado poderá ser o *Google Chrome*, *Firefox*, *Opera*.

Enquanto a biblioteca *requests* fornece o código HTML, o *Selenium* consegue interagir com a página, desbloqueando blocos de informação HTML só acessíveis por clique.

A biblioteca *BeautifulSoup* faz a organização e interpretação do código HTML, o que permite navegar no mesmo através das tags e atributos. Nesta biblioteca uma das funções mais utilizadas foi a função *find* que recebia como argumentos o nome da tag (exemplos: *div*, *a*, *p*, *span*, *td*, *tr*) e os respetivos atributos.

Por exemplo, em algumas páginas era possível extrair a morada facilmente por a mesma se encontrar dentro de uma tag em que os atributos eram únicos. Ao ter a morada dentro de uma tag <p id =“address”> era possível usar a função *find* com os argumentos ‘*p*’ no lugar do nome da tag e para atributos (*attrs*) colocar o seguinte dicionário { ‘id’: ‘address’ }.

Outra função muito utilizada foi o *find_all* para extrair os *urls* onde estavam as informações das empresas. Normalmente estes *urls* encontravam-se no atributo *href* de uma tag *a* e era necessário isolar as tags *a* que importavam para que não extraíssemos outras tags *a* que levassem por exemplo para a página de perguntas frequentes do portal (FAQ).

Como alguns dados dos portais poderiam estar incorretos, era sempre necessário fazer uma validação para não estragar os dados já existentes na base de dados de *marketing*. Para evitar danificar os dados, realizava-se a validação do *email*, telemóvel, cae, nif e código-postal. No caso do portal conter empresas de vários países, extraía-se o país e a morada para poder associar o código de duas letras a cada país. Por exemplo: Portugal tem como código PT e Espanha ES.

Quando havia empresas de países diferentes, extraía-se a morada para mais tarde ser possível extrair o código postal usando uma expressão regular dependente do país da empresa. Para portugal os códigos postais são da forma XXXX-XXX, mas, por exemplo, em espanha já são XXXXX onde X é um algarismo de 0 a 9.

Para validar o endereço de *email* foi usada a função *ISEMAIL()* do google sheets. Para os casos de Portugal e Espanha foram usadas expressões regulares para validar o nif, cae, código postal e telemóvel. Para extrair o código postal da morada de uma empresa portuguesa era usada a seguinte expressão ‘$\backslash d\{4\} - \backslash d\{3\}$’ . Por exemplo, na string ‘IEUA, Edifício 1 Campus de Santiago 3810-193 Aveiro’ era extraído como cep a substring ‘3810-193’. Na expressão regex o ‘$\backslash d$’ representa um dígito de 0-9 e a parte à frente do ‘$\backslash d$’ entre chavetas significa quantos vezes esse tipo de caracter é extraído.

Para validar o nif e telemóvel de Espanha eram consideradas válidas todas as *strings* que tivessem tamanho igual a 9. No caso da validação do telemóvel eram retirados o indicativo do país e os restantes carateres que não fossem dígitos antes de verificar o tamanho. Por vezes os números tinham o indicativo junto ao número de telefone ou telemóvel mas apresentado de formas diferentes. Por exemplo, para Portugal poderiam existir números com o indicativo da forma ‘00351’ ou ‘+351’ ou ‘351’.

O cae tinha de ter 5 dígitos caso pertencesse a Portugal e 4 dígitos caso pertencesse a Espanha. Para empresas portuguesas era usado com referência o CAE - Classificação Portuguesa das Atividades Económicas e para empresas espanholas era utilizado o CNAE - Clasificación Nacional de Actividades Económicas. Em empresas de outros países não era extraído o código de atividade por essa informação não estar no conjunto de informações disponibilizadas. Para as empresas que tinham código de atividade era necessário identificar qual era norma daquele código. Para isso, tinha de ser inserido um identificador, que no caso

de Portugal era 'cae_Rev3_pt' e para Espanha era 'cnae_2009_es'.

3.2 Pesquisa de dados em dois portais de empresas portuguesas

Como em alguns portais a única informação disponível era o nome e o endereço de *email* da empresa, era necessário arranjar uma forma de conseguir mais dados. Após o extrair a informação dos portais referidos anteriormente passava-se a mesma por um programa que pesquisava os NIF (número de identificação fiscal), cae (código de atividade empresarial) e cep (código de endereço postal) das empresas portuguesas em dois portais de empresas distintos tendo como base de pesquisa o nome ou, caso se soubesse, o nif da empresa. Estes dados serviam para facilitar a seleção das empresas para envio de *email* para várias campanhas de *marketing*. As campanhas serviam para angariar empresas para o portal centroproduto.com usando para isso o envio de *emails* em massa procurando fornecedores ou clientes para um dado produto ou serviço.

O primeiro passo consistiu em descobrir o padrão das empresas não encontradas pelo programa existente numa amostra de nomes de empresas. Existiam empresas em que o nome legal tinha a seguir às siglas 'lda' e 's.a.' o nome das várias lojas da empresa. Por exemplo 'Tintas 2000' estava registada na base de dados como 'Tintas 2000 S.A. - loja de Aveiro'. Ao pesquisar esse nome nos portais não era possível encontrar um resultado com o nome exatamente igual.

Decidiu-se implementar a seguinte melhoria: retirar ao nome da empresa procurada e aos resultados da pesquisa, as siglas 's.a.', 'lda.' e as suas variações juntamente com tudo o que viesse à frente dessa sigla.

Com a melhoria implementada foram encontradas mais algumas empresas. No entanto, continuavam a existir empresas que não apareciam na lista de resultados com o nome legal exatamente igual ao nome legal das empresas procuradas. Por vezes, a diferença era apenas em um caracter (espaço, ponto, vírgula, hífen).

Para solucionar esse problema foi implementada uma função de similaridade que retornava um valor entre 0 e 1, consoante as os nomes das empresas fossem mais parecidos (próximos de 1) ou fossem muito diferentes (próximos de 0).

A função de similaridade implementada utiliza a função 'sequencematcher', disponível em python, que tem por base o algoritmo de Rateliff e Obershelp. Existem outros algoritmos para a comparação de *strings* tais como: a distância de Levenshtein, distância de Damerau – Levenshtein, a distância de Hamming e a distância de Jaro – Winkler.

Os maiores problemas no uso do algoritmo de Rateliff e Obershelp nesta tarefa ocorrem quando o nome da empresa a procurar é grande e os resultados o apresentam de forma abreviada, e vice-versa, assim como quando existem nomes semelhantes pertencentes a empresas

diferentes.

Por exemplo, ao pesquisar 'Eurotamega-Sociedade Comercial de Importações e Exportações de Equipamentos e Serviços do Tamega, Lda.' era devolvido nos resultados 'Eurotamega-Sociedade Comercial de Importações e Exportações de Equipamentos e Serviços do Tameg'. Ao olho humano conseguimos ver que são exatamente a mesma empresa, mas para o computador são *strings* completamente diferentes. Com o uso da função de similaridade obteríamos a resposta que as empresas eram muito similares, com 0,99 de similaridade.

Mesmo com estas alterações, ainda havia empresas que escapavam à pesquisa porque eram pesquisadas com um nome mais curto do que aquele que existia nos *websites*.

Por exemplo, 'Garfos e Letras' existia nos *websites* como 'Garfos e Letras - Livraria Temática de Gastronomia Lda' e como só se aceitavam resultados com mais de 0.85 de similaridade estas empresas também não eram 'encontradas'.

Com base neste problema, a função de similaridade foi atualizada de modo a truncar a *string* maior para o tamanho da mais pequena e penalizando a sua similaridade em 0,1.

3.3 Algoritmos para a comparação de strings

3.3.1 Algoritmo de Rateliff e Obershelp

Rateliff e Obershelp desenvolveram um algoritmo de reconhecimento de padrões [4] que proporcionou uma forma de comparar duas *strings* sem haver a necessidade da existência de uma correspondência exata de caracteres. O algoritmo analisa duas *strings* e encontra a maior *substring* de carateres comum. Essa *substring* é usada como ponto de referência. De seguida, o par formado pelas *substrings* à esquerda do ponto de referência e o par formado pelas *substrings* à direita do ponto de referência são colocadas numa pilha para análise. Este procedimento é repetido para todos os pares de *strings* presentes na pilha até que não reste nenhum para analisar. O valor devolvido pelo algoritmo é o dobro do número de carateres encontrados em comum dividido pelo número total de caracteres nas duas *strings*.

Por exemplo, imaginemos que temos na base de dados a empresa 'olba comercio de materiais de tinta, lda' e nos resultados temos 'olba - comercio de materiais tinta, lda'. Primeiramente é detetado ' comercio de materiais ' como sendo a maior *substring* comum aos nomes das empresas, somando 46 à pontuação atual. Em seguinda os pares de *substrings* à esquerda ('olba' e 'olba -') e à direita ('de tinta, lda' e 'tinta, lda') são adicionados à pilha para posterior análise. Ao analisar o par 'de tinta, lda' e 'tinta, lda' o algoritmo deteta 'tinta, lda' como a maior *substring* em comum, acrescentando mais 20 à pontuação, ficando num total de 66. É atribuído a pontuação de zero às *strings* 'de ' e '' que não são acrescentadas à pilhas por não terem carateres em comum. De seguida, é analisado o par 'olba' e 'olba -' que tem como maior *substring* comum 'olba'. Mais uma vez é atribuído a pontuação de zero às *strings* '' e ' -' por não existirem carateres em comum. Não restando mais pares na

pilha, a pontuação final de 74 é dividida pela soma do número de carateres das duas *strings* (79 caracteres) obtendo-se 0.94 de similaridade entre as *strings*. Com 94% de similaridade pode-se concluir que os nomes são muito semelhantes, mesmo não sendo exatamente iguais.

Como o algoritmo efetua muitas comparações para determinar a maior *substring* comum a duas *strings*, é expectável que demore muito tempo para calcular a similaridade entre duas *strings* muito longas. No pior dos casos, as *strings* não têm nenhum caracter em comum. Nesse caso são efetuadas N*M comparações, em que N é o tamanho da primeira *string* (número de caracteres) e M o tamanho da segunda *string*.

3.3.2 Algoritmo de Hamming

O algoritmo de Hamming [5] consiste em calcular o número mínimo de substituições necessárias para transformar uma *string* noutra. O problema deste algoritmo é que só permite a comparação de *strings* do mesmo tamanho, o que seria um inconveniente para a comparação de nomes de empresas, uma vez que raramente os nomes a comparar têm o mesmo tamanho. Mesmo assim, poderia ser usado para saber o número mínimo de substituições necessárias para substituir um nome no outro quando os nomes a comparar tivessem o mesmo tamanho.

3.3.3 Distância de Levenshtein

A distância de Levenshetein [6] consiste no mínimo de alterações necessárias para transformar uma *string* noutra. Matematicamente, a distância de Levenshtein entre duas *strings* a e b denotada por $\mathrm{L}_{(a,b)}(|a|,|b|)$ pode ser calculada através da seguinte fórmula [18]:

$$\mathrm{L}_{(a,b)}(i,j) = \begin{cases} \max\{i,j\} & \text{se } \min\{i,j\} = 0 \\ \min\{\, \mathrm{L}_{(a,b)}(i-1,j)+1, & \\ \qquad \mathrm{L}_{(a,b)}(i,j-1)+1, & \text{caso contrário.} \\ \qquad \mathrm{L}_{(a,b)}(i-1,j-1)+1_{a_i \neq b_i}\} & \end{cases} \tag{3.1}$$

onde $1_{(a_i \neq b_j)}$ é a função indicadora que é igual a zero se $a_i = b_j$ e igual a 1 caso contrário, $\mathrm{L}_{(a,b)}(i,j)$ é a distância entre os primeiros i carateres de a e os primeiros j carateres de b.

É de reparar que a primeira condição do mínimo corresponde à eliminação do caracter i na string b, a segunda condição à inserção, a terceira condição à correspondência ou substituição dos carateres dependendo se os carateres são iguais ou diferentes.

Usando a distância de Levenshtein irá verificar-se que serão necessárias no mínimo 4 alterações para transformar 'Associação Humanitária dos Bombeiros Voluntários de Sendim' em 'Associação Humanitária dos Bombeiros Voluntários de Serpins'. Para simplificar o processo vai-se analisar a parte de 'Serpins' e 'Sendim' pois o resto dos nomes são exatamente iguais. Para facilitar os cálculos irá usar-se a Tabela 3.1. Esta tabela é construída usando a fórmula (3.1) onde as linhas estão indexadas desde $i = 0$ até $i = |a|$ e as colunas estão indexadas desde

$j = 0$ até $j = |b|$.

Tabela 3.1: Cálculo da distância de Levenshtein entre 'Sendim' e 'Serpins'

		S	E	N	D	I	M
	0	1	2	3	4	5	6
S	1	0	1	2	3	4	5
E	2	1	0	1	2	3	4
R	3	2	1	1	2	3	4
P	4	3	2	2	2	3	4
I	5	4	3	3	3	2	3
N	6	5	4	3	4	3	3
S	7	6	5	5	4	4	4

Observando a tabela 3.1 é possível saber quais as alterações necessárias para transformar SENDIM em SERPINS. A primeira alteração é a substituição de 'R' por 'N', a segunda alteração também é uma substituição do 'P' por 'D', a terceira alteração é a substituição da letra 'N' para a letra 'M' e a última alteração é a remoção da letra 'S'.

3.3.4 Algoritmo de Damerau - Levenshtein

Mais tarde Federick Damerau acrescentou a possibilidade de troca de carateres adjacentes. A razão pela qual Damerau adicionou esta possibilidade foi porque, segundo ele mais de 80% dos erros humanos de escrita são inserção, eliminação, substituição e troca de carateres. A distância de Damerau- Levenshtein [7, 8] entre duas *strings* a e b ($\mathrm{DL}_{(a,b)}(|a|,|b|)$) é obtida através da seguinte fórmula [19, 21]:

$$\mathrm{DL}_{(a,b)}(i,j) = \begin{cases} \max\{i,j\} & \text{se } \min\{i,j\} = 0 \\ \min\{\, \mathrm{DL}_{(a,b)}(i-2,j-2)+1, & \\ \quad \mathrm{DL}_{(a,b)}(i-1,j)+1, & \text{se } i,j>1 \text{ e } a_i = b_{j-1} \text{ e } \ a_{i-1} = b_j \\ \quad \mathrm{DL}_{(a,b)}(i,j-1)+1, & \\ \quad \mathrm{DL}_{(a,b)}(i-1,j-1)+1_{a_i \neq b_i}\} & \\ \min\{\, \mathrm{DL}_{(a,b)}(i-1,j)+1, & \\ \quad \mathrm{DL}_{(a,b)}(i,j-1)+1, & \text{caso contrário.} \\ \quad \mathrm{DL}_{(a,b)}(i-1,j-1)+1_{a_i \neq b_i}\} & \end{cases} \tag{3.2}$$

onde $1_{(a_i \neq b_j)}$ é a função indicadora que é igual a zero se $a_i = b_j$ e igual a 1 caso contrário, $\mathrm{DL}_{a,b}(i,j)$ é a distância de Damerau-Levenshtein entre os primeiros i carateres de a e os

primeiros j carateres de b.

Apesar de adicionar a possibilidade de haver troca de carateres adjacentes, para o caso que estamos a estudar este algoritmo não acrescenta nada relevante comparativamente com o algoritmo de Levenshtein. De facto, não foi observada qualquer troca de ordem dos carateres entre os nomes das empresas procuradas e os nomes das empresas encontradas, quando os nomes se referiam à mesma empresa.

3.3.5 Algoritmo de Jaro

Em 1989 foi desenvolvido outro algoritmo por Matthew Jaro [9, 10] que também tinha como objetivo verificar se duas *strings* eram parecidas ou não. Mais tarde William Winker alterou o algoritmo, pois acreditava que a similaridade entre duas *strings* com um grande prefixo comum deveria ser mais elevada do que as que fossem diferentes logo nos primeiros símbolos.

O algoritmo (também conhecido como distância) de Jaro pode ser representado através da seguinte fórmula [20]:

$$\mathrm{J}_{(a,b)} = \begin{cases} \frac{1}{3} \cdot \left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m} \right) & m > 0 \\ 0 & m = 0 \end{cases} \tag{3.3}$$

onde $|a|$ e $|b|$ são o tamanho das strings a e b, respetivamente, m é o número de carateres coincidentes e t é o número de transposições.

Um carater é considerado coincidente se aparece na *string* a e na *string* b e a distância entre as posições desse carater em cada uma das *strings* não é maior do que

$$\left\lfloor \frac{\max(|a|, |b|)}{2} - 1 \right\rfloor .$$

Para cada par $\{\alpha, \beta\}$ de carateres coincidentes que não apareçam na mesma ordem nas duas strings incrementa-se um na variável t, ou seja, admite-se que existe uma troca.

Por exemplo, comparando as palavras 'PRECEBER' e 'PERCEBER' conseguimos ver que todas as letras são coincidentes ($m = 8$), mas a letra 'R' aparece na primeira palavra na segunda posição enquanto que na segunda palavra aparece na terceira posição da sequência de correspondência, logo $t = 1$.

De seguida, iremos comparar partes de dois nomes de empresas já usados como exemplos anteriormente: 'SENDIM' e 'SERPINS'. Existem 3 carateres coincidentes ($m = 3$). Como os carateres coincidentes, 'S', 'E' e 'I' aparece na mesma ordem em ambos os nomes ('S' em primeiro, 'E' em segundo e 'I' em terceiro) então $t = 0$. Podemos concluir que a distância de

Jaro entre 'SENDIM' e 'SERPINS' é aproximadamente 0.64,

$$\mathrm{J}_{(a,b)} = \frac{1}{3} \cdot \left(\frac{3}{6} + \frac{3}{7} + \frac{3-0}{3} \right) \simeq 0.64 \ .$$

Caso os nomes fossem só 'SERPINS' e 'SENDIM' podíamos concluir que não corresponderiam à mesma empresa, mas como têm a parte de 'Associação Humanitária dos Bombeiros Voluntários de ', a distância de Jaro mostra que os nomes das empresas são muito parecidos, ou seja, a distância de Jaro fica próximo de 1 o que dá a entender que os nomes correspondem à mesma empresa,

$$\mathrm{J}_{(a,b)} = \frac{1}{3} \cdot \left(\frac{56}{58} + \frac{56}{59} + \frac{56-1}{56} \right) \simeq 0.96 \ .$$

Como se pode observar a distância de Jaro entre 'Associação Humanitária de Bombeiros Voluntários de Serpins' e 'Associação Humanitária dos Bombeiros Voluntários de Sendim' é de 0.96, o que dá erradamente a entender que os nomes pertencem à mesma empresa.

3.3.6 Algoritmo de Jaro - Winkler

Como já foi referido anteriormente Winkler [11] modificou a distância de Jaro de modo a que duas *strings* que fossem muito parecidas nos primeiros carateres a comparação dessas *strings* obteria uma maior pontuação.

A fórmula [20] de Jaro-Winkler é a seguinte:

$$\mathrm{JW}_{(a,b)} = \mathrm{J}_{(a,b)} + l \cdot p \cdot (1 - \mathrm{J}_{(a,b)}) \tag{3.4}$$

onde $\mathrm{J}_{(a,b)}$ é a distância de Jaro, l é o número de carateres coincidentes no início das duas *strings* e $p \in [0, 1]$ é um coeficiente.

Ao analisar a fórmula da distância de Jaro-Winkler vemos que quando $l \cdot p = 1$ temos que $\mathrm{JW}_{(a,b)} = \mathrm{J}_{(a,b)} + 1 \cdot (1 - \mathrm{J}_{(a,b)}) = 1$. Então se $l \cdot p = 1$ a função de Jaro-Winkler mostra-nos que as duas *strings* são exatamente iguais nos primeiros l carateres.

Pode-se concluir que, para conseguirmos estabelecer que uma *string* é igual à outra se os primeiros l carateres forem iguais em ambas as *strings*, temos de colocar o coeficiente p de tal modo que $l \cdot p = 1$, ou seja, p deverá ser $p = \frac{1}{l}$. Por exemplo, se quisermos considerar que dois nomes pertencem à mesma organização caso tenham os primeiros 4 carateres iguais devemos estabelecer $p = 0.25$.

Para este problema, o p talvez deva ser $3/2$ do inverso do tamanho do menor nome a analisar, caso o nome tenha mais do que 10 carateres. Caso o nome tenha menos de 10 carateres, o p deve ser o inverso do tamanho do menor nome em análise.

Voltando ao exemplo anterior, temos que o menor nome tem 58 carateres, pelo que

$$p = \frac{3}{2} \cdot \frac{1}{58} \simeq 0.03 \ .$$

Então,

$$\begin{aligned} \mathrm{JW}_{(a,b)} &= \mathrm{J}_{(a,b)} + l \cdot p \cdot \left(1 - \mathrm{J}_{(a,b)}\right) \\ &= 1.02 \end{aligned}$$

Como podemos observar neste exemplo, ao contrário das outras distâncias aqui referidas, a distância de Jaro-Winkler pode ser maior do que 1, caso o número de carateres iniciais coincidentes seja maior que o número estipulado para cálculo do coeficiente p.

3.4 Aplicação do algoritmo de Rateliff e Obershelp

Inicialmente o conjunto de testes fornecido continha os nomes de 283 empresas, sendo 138 repetidos, diferindo apenas na parte final pela localização da loja.

Quando se aplicou somente o algoritmo de Rateliff e Obershelp na comparação dos nomes das empresas conseguiram-se encontrar 238 pares com pontuação superior a 0.85, sendo que somente 230 desses pares estavam corretos. Das correspondências incorretas a similaridade estava compreendida entre 0.85 e 0.9523.

Após implementar a penalização de 0.1 na similaridade, caso um nome tivesse comprimento superior a 10% do comprimento do outro, obtiveram-se 248 pares de empresas corretos e 12 pares incorretos num conjunto de 260 pares de empresas. As correspondências incorretas encontravam-se num intervalo de similaridade entre 0.85 e 0.952. Nesse mesmo intervalo também existiam 54 pares corretos, logo não compensa remover as empresas encontradas neste intervalo de similaridade.

Posteriormente foi facultado um conjunto com 85 801 empresas em que foram encontradas 78 887 empresas com pontuação maior a 0.85 usando como método de comparação a similaridade de Rateliff e Obershelp com penalização, das quais só 78 757 estavam corretas. Esta verificação inicialmente foi efetuada à mão, não só neste conjunto, mas como em todos os outros durante o estágio.

Ao juntar os conjuntos verificados à mão obteve-se um conjunto com 83 029 pares de nomes de empresas encontrados em que apenas 82 811 são empresas que tiveram a correspondência correta na procura no website.

Para cada par de nomes de empresas do conjunto de empresas referido anteriormente, simplificaram-se e depois calculou-se a similaridade entre eles usando o algoritmo de Rateliff e Obershelp de três formas.

A abordagem utilizada para simplificar os nomes das empresas foi: converter todos os carateres para minúsculas; remover acentuação; substituir '&' por 'e'; substituir 'ª' por 'a';

substituir ‘sucrs’ por ‘sucessores’; retirar o que está à frente de ‘lda’, ‘ limitada’, ‘s a ’, ‘ s.a’, ‘ sa ’; remover ‘sa’, ‘lda’, ‘unipessoal’, ‘unipessoa’, ‘sa’, ‘ca’; substituir ‘,’ por ‘–’.

As três maneiras de calcular a similidaridade foram: usando o algoritmo de Rateliff e Obershelp, a similaridade de Rateliff e Obershelp aplicando a penalização anteriormente referida e a similaridade entre os nomes usando o Algoritmo 2.

Algoritmo 2

1: Separar ambos os nomes por ‘–’ e guardar as partes de cada nome;
2: Calcular a similaridade da primeira parte do nome original com a primeira parte do nome resultado da pesquisa usando o algoritmo de Rateliff e Obershelp e guardar na variável X;
3: SE o nome original tiver sido dividido em mais do que uma parte ENTÃO calcular a similaridade da segunda parte do nome original com a primeira parte do nome do resultado usando o algoritmo de Rateliff e Obershelp e guardar na variável Y.
4: Retornar o maior valor entre X e Y.

Nos resultados obtivemos os seguintes conjuntos representados na Figura 3.1:

- empresas com similaridade igual a 1 usando o algoritmo Rateliff e Obershelp (conjunto A);
- empresas com similaridade 1 obtidas através do Algoritmo 2 (conjunto B);
- empresas obtidas por utilização do algoritmo de Rateliff e Obershelp com penalização (conjunto C);
- empresas corretas (conjunto D).

Na tabela 3.2 são apresentados os resultados da aplicação do algoritmo Rateliff e Obershelp. Apesar do uso do algoritmo com penalização adicionar mais resultados corretos também adiciona empresas que estão incorretas. Ao se misturarem as similaridades dos resultados com penalidades com os resultados sem penalidades fica difícil definir um intervalo em que haja uma forte probabilidade das empresas estarem corretas.

O conjunto C continha empresas corretas (3 140) e incorretas (62) e maior parte do nome legal das empresas era do tipo ‘*<nome da empresa> – <designação da responsabilidade + nome da loja>*’ e outras do tipo ‘*<sigla> – <nome legal> – <designação da responsabilidade + nome da loja >*’. Criou-se o Algoritmo 2 para comparar a primeira parte do nome legal das empresas procuradas com a primeira parte do nome legal das empresas encontradas e comparar a segunda parte de nome a procurar com a primeira parte do nome encontrado e retornar a similaridade mais alta. Caso a similaridade seja igual a 1 então é altamente provável que a empresa procurada e a empresa encontrada sejam a mesma. Isto pode concluir-se do facto que dos 80 464 elementos do conjunto B em que 14 466 não são exatamente iguais (ou seja, não pertencem ao conjunto A) somente um par de empresas é que não pertence à mesma empresa.

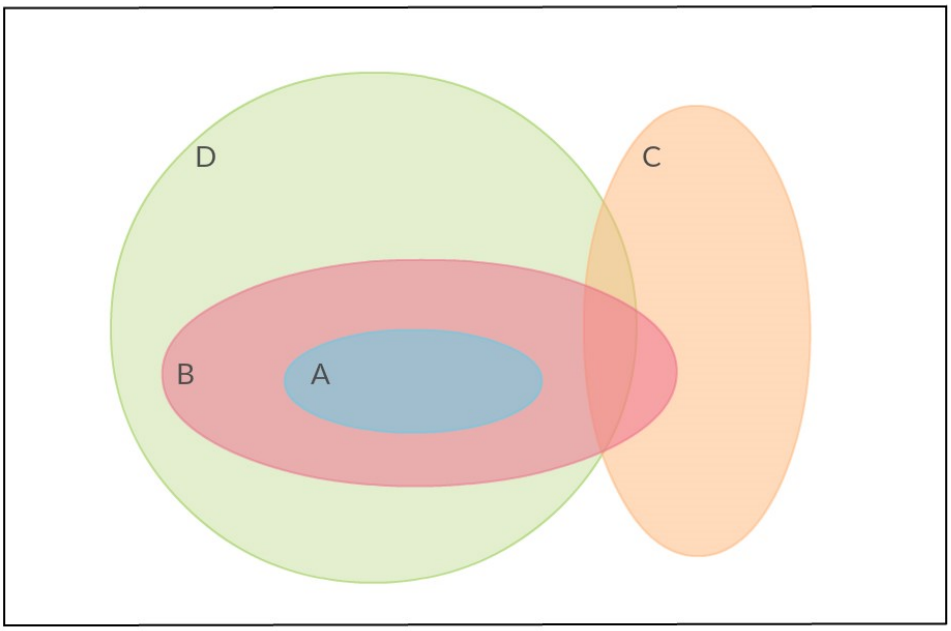


Figura 3.1: Representação dos conjuntos de empresas

Usando o Algoritmo 2 para decidir se a empresa procurada e a empresa encontrada são a mesma empresa só acarretaria num erro de $6.9 \times 10^{-5}\%$, enquanto que usando o algoritmo de Rateliff e Obershelp apesar de encontrar mais empresas obteríamos um erro de 0.2 %.

A forma de simplificar e comparar os nomes poderia ser ainda melhorada para poder identificar automaticamente como corretas mais algumas das restantes 2348 empresas corretas. Na parte da simplificação dos nomes poder-se-ia: não eliminar 'ca' dos nomes; eliminar 'crl' do fim dos nomes legais e 'unip'; substituir 'ca' e 'c^{a}' por 'companhia'. Na parte da comparação poder-se-ia melhorar retirando os '-' de ambos os nomes ao verificar se o par de nomes legais pertencem à mesma empresa.

3.5 Conclusões

O algoritmo de Rateliff e Obershelp, utilizado para obter os resultados apresentados na secção anterior poderá não ser o mais apropriado. De igual modo os algoritmos de Leveinshtein e de Damerau-Leveinshtein apresentam algumas desvantagens.

Por exemplo, a empresa 'Associação Humanitária dos Bombeiros Voluntários de Serpins' obviamente não corresponde à empresa 'Associação Humanitária dos Bombeiros Voluntários de Sendim', mas a similaridade entre as strings é de 0.94 aplicando o algoritmo de Rateliff e Obershelp, o que erradamente dá a entender que os nomes poderão ser iguais a menos de erros ortográficos.

Tabela 3.2: Resultados da aplicação do algoritmo Rateliff e Obershelp

Condição	Número de empresas
Empresas corretas	82811
Empresas com similaridade maior que 0.85 com uso de penalização	79671
Empresas encontradas unicamente por uso da penalidade no algoritmo de Rateliff e Obershelp	3202

Usando a distância de Leveinshtein e de Damerau-Leveinshtein são necessárias no mínimo 4 alterações para transformar o primeiro nome no segundo, três substituições de letras e uma remoção. Desta forma saberíamos que as *strings* não são iguais, mas mesmo assim seria necessário dar pesos diferentes à inserção, substituição e remoção.

A inserção e remoção deveriam ter um peso menor do que a substituição, pois ao haver muitas substituições de carateres de uma *string* para a outra é provável que os nomes das empresas sejam diferentes. Já a inserção e remoção não deveriam ter muito peso pois em grande parte dos casos o que muda do nome de empresa procurado para o nome de empresa encontrado é a designação legal que num lugar está abreviada e no outro não.

Entre os algoritmos apresentados, aquele que teria provavelmente melhores resultados práticos seria o algoritmo de Jaro-Winkler. Este permite ter um controlo de quantos carateres iniciais consideraríamos necessários para haver uma maior confiança de que o nome procurado e o nome encontrado pertencem ou não à mesma empresa, através do coeficiente p. Ao mesmo tempo, se existisse um erro nos primeiros carateres devolveria a similaridade entre os nomes, dando maior importância aos primeiros $1/p$ carateres. Estes $1/p$ carateres podem ser os carateres antes do '-' no nome da empresa, para que o nome legal seja mais importante que a designação da área de atividade da empresa.

Pode-se observar no exemplo usado para apresentar este algoritmo que apesar da distância obtida ser maior do que 1, os nomes são diferentes. Uma solução para estes nomes de organizações que são iguais no início, mas no fim diferem no nome da localidade, poderia passar por inverter a ordem dos carateres na *string* e assim aparecer a parte que poderia diferir ao início da *string* e desta forma o algoritmo poderia funcionar também nestes casos. As organizações mencionadas anteriormente poderiam ser: de nomes das associações, câmaras municipais, freguesias.

Para automatizar a verificação de que os nomes pertencem à mesma empresa, dever-se-ia implementar o Algoritmo 2 guardando sempre o número de empresas encontradas e o número de empresas consideradas corretas. Se o número de empresas consideradas corretas for muito

inferior ao número de empresas encontradas, tal poderá significar que existem mais algumas empresas corretas que foram rejeitadas. Caso isso venha a acontecer dever-se-ia analisar a causa desse acontecimento e caso seja possível melhorar o algoritmo.

Capítulo 4

Classificação automática de emails

Com o objetivo de ter mais empresas registadas no portal, a equipa de *marketing* faz campanhas via *email*, com conteúdo alusivo a empresas que melhoraram o seu negócio com a ajuda do uso do portal, ou com pedido de fornecedores ou clientes para um determinado produto. Por causas externas à empresa e como os *emails* são enviados em massa, existiam muitos *emails* em que a entrega falhava e retornavam um *email* de '*delivery status notification*'. Para tratar desses *emails* decidiu-se elaborar um programa que conseguisse classificar se a entrega do *email* tinha falhado devido à caixa de correio estar cheia, ou por a conta estar inativa, ou por a conta para a qual o *email* foi enviado não existir ou por outras razões.

O principal objetivo do programa seria detetar corretamente os três primeiros tipos de falha para que esses endereços de *email* não fossem novamente usados pela equipa de *marketing* e não houvesse desperdício de cota de envio de *emails*.

As fontes dos *emails* a serem analisadas são concatenadas e escritas num ficheiro .mbox através de uma ferramenta da Google chamada *Google Takeout* e depois convertidas para um ficheiro de texto (.txt). Após detetar que a Google inseria um cabeçalho antes do início da fonte de cada *email*, que era constituído por uma string que começa por 'From: ' e continha na mesma linha '@xxx', conseguiu-se uma forma de delimitar os vários *emails* presentes no ficheiro .txt. A Figura 4.1 ilustra um exemplo resumido de um ficheiro de input, em que na primeira linha aparece a *string* que separa os *emails* (cabeçalho do *email*). Numa versão inicial do programa, o processamento de cada *email* foi efetuado de acordo com o Algoritmo 3.

No passo 1 retiram-se as linhas com mais de 50 carateres, por se acreditar que eram linhas com informação encriptada sem valor. No passo 2 extraía-se a data e a linha delimitadora do *email* para ser possível encontrar a fonte do *email*, caso houvesse um erro ao interpretar. Ao isolar a fonte do *email* era mais fácil encontrar o que estava errado e corrigir.

Para retirar o corpo do *email* ou seja a mensagem (passo 3) era usado um autómato, representado na Figura 4.2, que percorria a fonte do *email*. O texto só era guardado como corpo da mensagem enquanto o autómato estivesse no estado 1. O estado 2 foi meramente

criado para guardar mensagens que chegavam em formato HTML, para serem usadas mais tarde, caso fosse necessário.

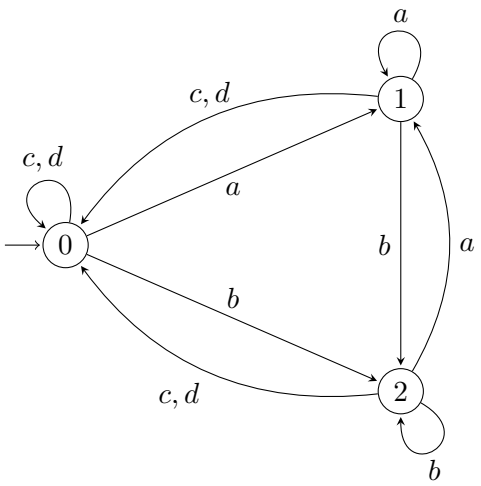
```
From 303078955022705520485@xxx Wed Apr 11 00:06:37 +0000 2018

(fonte do primeiro email)

From 270552048530307891O2@xxx Wed Apr 11 00:06:37 +0000 2018
X-GM-THRID: AAAAAAAAAAAAAAAAAA
X-Gmail-Labels: Archived, EXEMPLOS
MIME-Version: 1.0
Message-ID: (omited info)
Date: Wed, 11 Apr 2018 00:06:37 +0000
Subject: Assunto
From: =?UTF-8?B?QW5hIENsw6F1ZGlh?= <remetente@remetente_dominio>
To: <destinatario@destinatario_dominio>
Content-Type: multipart/alternative; boundary="001a113a2d28a9853c05698769ff"

--001a113a2d28a9853c05698769ff
Content-Type: text/plain; charset="UTF-8"; format=flowed; delsp=yes
Content-Transfer-Encoding: base64

RXhlbXBsbyBkZSBtZW5zYWdlbTogb2zDoQ==
--001a113a2d28a9853c05698769ff
Content-Type: text/html; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

<div dir=3D"auto">Exemplo de mensagem: ol=C3=A1</div>
--001a113a2d28a9853c05698769ff--
```

Figura 4.1: Exemplo do ficheiro de input

Algoritmo 3

Para cada email:

1: Eliminar linhas com informação 'encriptada';
2: Procurar e guardar a data e a linha delimitadora do email;
3: Procurar e guardar o corpo do email;
4: Procurar e guardar o remetente e o destinatário;
5: Procurar e guardar erros de falha de entrega.
6: Classificar o email e guardar a informação num ficheiro de OUTPUT



Legenda:

a - se a linha continha 'content-type: text-plain'

b - se a linha continha 'content-type: text/html'

c - se a linha continha 'content-type:' mas não era a condição a nem b

d- se a linha continha 'from' e '@xxx'

Figura 4.2: Ilustração do Autómato

Para retirar os endereços do remetente e do destinatário era percorrida novamente a fonte do *email* até encontrar a linha começada por 'From:'. Os endereços eram extraídos com a ajuda de uma expressão regular do python. A expressão utilizada era

$$[\backslash w\backslash.-] + @[\backslash w\backslash.-] + \backslash.\backslash w+$$

indicando, essencialmente, que o endereço começava com uma palavra seguida de '@', seguida de outra palavra, seguida do carater '.', terminando com uma palavra. Em seguida, filtravam-se os endereços de *email* que corresponderiam a mensagens enviadas automaticamente por servidores de *email*. Os endereços que eram considerados como sendo de servidores eram os que verificavam uma destas condições:

- as palavras 'hosting', 'server', 'spam' estivessem no endereço;
- o endereço tivesse mais que 48 carateres;
- as palavras '@mail.gmail'ou '@xxx' estivesse no domínio do endereço ('@' + parte à direita do '@');
- o domínio do endereço fosse igual ao domínio da conta de onde foram extraídos os *emails*;
- o domínio do *email* tivesse mais que 32 carateres;
- as palavras 'mailer-daemon', 'postmaster@', '.jpg', '.png' estivessem no nome do endereço (parte à esquerda do @ + @);
- o nome do endereço tivesse mais que 32 carateres.

O processo para a procura do destinatário era semelhante só que ao invés de procurar a linha que começasse por 'From:' procurava-se a linha que começasse por 'To:'.

Para saber se existia algum erro de falha de entrega classificável por *Simple Mail Transfer Protocol* (SMTP) *Enhanced Status Codes Registry*[12] eram extraídos da fonte do *email*, com o auxílio de uma expressão regular, todas as strings do tipo X.X.XX ou X.X.X em que X é um dígito de 0 a 9.

Para um *email* ser considerado blacklisted (ou seja para eliminar da base de dados), deveria ter um dos seguintes erros (blacklist_errors): 'X.1.1' (Bad destination mailbox address); 'X.1.3' (Bad destination mailbox address syntax); 'X.1.6' (Destination mailbox has moved, No forwarding address); 'X.2.1' (Mailbox disabled, not accepting messages); 'X.2.2' (Mailbox full); 'X.2.3' (Message length exceeds administrative limit); 'X.3.1' (Mail system full), em que X era um dígito de 0 a 9.

Para a classificação dos *emails* era utilizado o algoritmo 4.

Algoritmo 4

ENTRADA: error_codes(códigos de erro encontrados); fonte do email; blacklist_errors (erros predefinidos para eliminar)
SAÍDA: category (classificação)

SE $|error\_codes| > 0$ E $\forall x \in error\_codes : x \notin blacklist\_errors$:
| $category = ERROR$
MAIS SE $error\_codes| > 0$ E $\exists x \in error\_codes : x \in blacklist\_errors$:
| $category = BLACKLISTED$
MAIS SE $'sender'$+*domínio do utilizador estiverem no início de uma linha da fonte do email*:
| $category = SENT$
MAIS SE $'forwardmessage'$ *estivesse na fonte do email*:
| $category = FORWARDED$
MAIS SE $'auto\text{-}submitted : auto\text{-}replied'$ *ou*
$'precedence : auto\_reply'$ *estivessem na fonte do email*:
| $category = AUTO - REPLY$
MAIS SE $'content\text{-}type : text/plain'$ *estivesse na fonte do email*:
| $category = REPLY$
SENÃO:
| $category = UNKNOW$

Após a classificação, os dados relativos a cada *email* (cabeçalho, data, corpo da mensagem, endereço do remetente) eram colocados num ficheiro de output excel e separados por quatro folhas (BLACKLIST, ERROR, REPLY e AUTO-REPLY) de acordo com a categoria em que foram classificados. As informações dos *emails* das restantes categorias FORWARD e SENT eram ignoradas, pois não eram consideradas importantes.

Após uma análise mais cuidada do ficheiro de saída do programa foi verificado que existiam alguns *emails* com o corpo da mensagem vazio. Devido a esse facto não ser comum e ao repetir-

se em vários *emails*, foi-se verificar a fonte de cada um. Constatou-se que esses *emails* ao invés de terem 'Content-Transfer-Encoding: quoted-printable' que retorna um texto próximo do normal (à exceção dos caracteres que não eram letras sem acentuação, números e alguma pontuação), tinham 'Content-Transfer-Encoding: base64' que eram mensagens codificadas na codificação base64 e eram apresentadas em sequências de carateres sem espaço. Sendo assim concluiu-se que parte da informação retirada como informação encriptada era a mensagem do *email* codificada na base64.

O novo programa passou a usar a biblioteca *email* [14] pertencente ao python para interpretar os cabeçalhos do *email* tais como o 'From:' e 'To:' uma vez que com o programa anterior por vezes não se obtinham os *emails* corretos do remetente e destinatário para *emails* classificados como REPLY ou AUTO-REPLY.

Do algoritmo usado no programa anterior (Algoritmo 3) retirou-se o passo 1 e modificou-se a forma como se efetuavam alguns dos outros passos, conforme descrito no algoritmo 5.

Algoritmo 5

Para cada email:

1: Procurar e guardar a data e linha delimitadora do email;
2: Procurar e guardar o corpo do email;
3: Procurar e guardar o remetente e o destinatário;
4: Procurar e guardar erros de falha de entrega;
5: Classificar o email e guardar a informação num ficheiro de OUTPUT.

O passo 1 continuou a ser efetuado da mesma forma que o passo 2 do Algoritmo 3.

A nova forma de retirar a mensagem do *email* (passo 2) foi usando essencialmente as funções *is_multipart* e *get_payload*. A função *is_multipart* verificava se o *email* estava no formato message/rfc822 e, nesse caso usava as funções *is_multipart* e *get_payload* para extrair o primeiro 'Content-type', fosse ele do tipo text/plain ou text/html. Caso o *email* não estivesse no formato message/rfc822 fazia-se uso unicamente da função *get_payload* para extrair o corpo da mensagem do *email*. Posteriormente esse conteúdo era descodificado segundo o *charset* extraído, fazendo uso da função *get_content_charset* também pertencente à biblioteca *email*.

No passo 3 passou-se a aceder aos cabeçalhos 'FROM:' e 'TO:' do *email* para retirar os endereços do remetente e do destinatário, respetivamente. Continuou-se a extrair os endereços com a mesma expressão regular usada anteriormente, mas a forma de filtrar os endereços mudou. Por alguns endereços serem muito extensos e não pertencerem a endereços de servidores de *emails*, retiraram-se todas as restrições referentes aos tamanhos dos endereços e apostou-se em detetar mais *strings* que estivessem num endereço de servidores de *email*.

Desta forma os endereços que são filtrados e não são guardados passaram a ser os endereços que continham:

- 'hosting', 'server', 'spam' ou 'noreply@cisco.com';

- '@mail.gmail.com', '@xxx', 'hes.trendmicro.eu' ou 'hostgator.com' no domínio;
- 'daemon', 'mailer-daemon@', 'postmaster@', 'microsoftexchange', 'orgextensionproperties', 'postmaster.winx', '.jpg' ou '.png' no nome.

A forma de extrair os códigos de erro [12] na falha da entrega mudou para só considerar erro um código que fosse da forma Y.X.X ou Y.X.XX em que Y é 4 ou 5 e X é um algarismo de 0 a 9.

O modo de classificação dos *emails* também foi alterado. Um *email* é classificado como:

- BLACKLISTED se o remetente foi filtrado (pertence a *emails* de servidores que enviam relatório de falha de entrega), e se um dos códigos erros encontrados está lista dos blacklist ou se o corpo do *email* contiver uma das seguintes strings: 'endereço não encontrado', 'mailbox unavailable', 'address not found', 'no such user', 'over quota', 'mailbox is full', 'no mailbox here by that name', 'caixa de entrada do destinatário cheia', 'recipient not found', 'mailbox size limit exceeded', 'quota exceeded', 'recipient inbox full', 'user unknown', 'no such person at this address', 'unknown user', 'this address no longer accepts mail', 'not enough disk quota', 'no such recipient here', 'mail address is administratively disabled', 'mailbox full', 'recipients inbox is full', 'we do not accept mail to this address', 'user's mailfolder is full', 'user does not exist' ou 'recipient does not exist';
- SPAM se o remetente foi filtrado e o *email* não for classificado como BLACKLISTED e se o corpo do *email* contiver uma das seguintes strings: '`https://support.google.com/a/answer/168383`', 'spam detected in your mail', 'high probability of spam', 'JunkMail rejected', 'Sender denied', 'message looks like spam or phish to me', 'the recipient's *email* system refused to accept a connection from your *email* system';
- MISSPELLED se o remetente foi filtrado e o *email* não for classificado como BLACKLISTED ou SPAM e se o corpo do *email* contiver uma das seguintes strings: '`https://support.google.com/mail/answer/7720`', 'address may be misspelled or may not exist', 'you might have spelled or formatted the group name incorrectly the account or domain may not exist, they may be blacklisted, or missing the proper dns entries' ou 'https://support.google.com/mail/answer/69585';
- ERROR os *emails* em que o remetente for filtrado e não foi classificado como BLACKLISTED, SPAM ou MISSPELLED;
- SENT se o remetente é o endereço inserido no início do programa;
- FORWARD se o *email* não é classificado como SENT e o remetente tem o mesmo domínio que o endereço inserido no início do programa, ou seja, se o *email* é enviado por uma pessoa da mesma empresa.

- AUTO-REPLY se na fonte do *email* contém 'auto-submitted: auto-replied' ou 'precedence: auto_reply'
- REPLY se o *email* não for classificado como nenhuma das categorias anteriores.

Antes de escrever o corpo da mensagem do *email* no ficheiro excel eram retirados todos os sinais de igual do início do texto para que ao abrir o ficheiro excel não desse erro ao interpretar aquele pedaço de texto como fórmula por essa célula começar por um igual.

Após classificados, os dados do *email* são armazenados num ficheiro excel e separados por seis folhas (BLACKLIST, ERROR, REPLY, AUTO-REPLY, SPAM e MISSPELLED). Os *emails* das restantes categorias (FORWARD e SENT) eram ignoradas, pois não eram consideradas importantes.

O tempo de execução aumentou relativamente ao programa anterior, mas a eficácia e correção também aumentaram, uma vez que foi possível extrair e analisar mais mensagens dos *emails*. A análise de 1000 *emails* passou de uma média de 30 seg para uma média de 2min e 30seg.

O tempo de execução aumentou devido ao número de linhas a processar por *email* ser maior, por já não se remover nenhuma parte da fonte do *email*. Para reduzir esse tempo era necessário remover da fonte do *email* a parte dos anexos que ocupavam muitas linhas.

Supondo que se receberia em média 1000 *emails* deste tipo para classificar por mês e que uma pessoa perderia em média 2 min a analisar o conteúdo do *email* para ver se o destinatário era para eliminar da base de dados ou se era uma resposta automática com alguma informação útil, podemos concluir que este novo programa poupa em média por mês 33h 17 min e 30 seg de trabaho rotineiro a uma pessoa da equipa de *marketing*, o qual pode ser utilizado para responder mais rapidamente aos clientes.

Capítulo 5

Conclusão

O presente estágio possibilitou aplicar conhecimentos obtidos ao longo do curso e ao mesmo tempo permitiu o contacto com o 'mundo do trabalho'.

Ao longo deste estágio houve vários projetos relacionados com tarefas realizadas manualmente pela equipa de *marketing*. As tarefas da equipa de *marketing* eram:

1. angariar e validar informação de empresas;
2. inserir informação extraída na base de dados;
3. extrair *emails* de empresas selecionadas na base de dados a partir de palavras-chave e códigos de atividades económicas;
4. envio de *emails* em massa para empresas;
5. remover *emails* em que a entrega falhou ou em que o utilizador do *email* pediu para remover da base de dados.

Na angariação de informações de empresas foi tentado automatizar a extração para portais diferentes. Tal mostrou-se um problema difícil de solucionar uma vez que cada portal organizava a informação dentro do HTML de uma forma diferente. Nesta tarefa, a parte da validação de dados poderá ser melhorada automatizando a parte de validação de dados, excepto as *keywords* e o *email*. O *email* continuaria a ser validado usando a função da google sheets. Outra melhoria seria gerar o ficheiro de output já com os cabeçalhos do ficheiro modelo usado para importar as informações para a base de dados.

No programa de inserção de dados poderia ser implementada a possibilidade de reescrever informação já existente.

Também poderia ser implementada a forma de armazenar os códigos de atividade económica, criando uma tabela relacional de códigos, para que cada empresa pudesse ter mais do que um código associado.

Ao inserir os dados é necessário melhorar a forma usada para inserir as *keywords*, uma vez que, quando existem palavras-chave negativas, como por exemplo bebidas não alcoólicas, estas são inseridas na base de dados como 'alcoolicas', 'nao', 'bebidas'. Deste modo é possível que ao extrair da base de dados empresas que vendam bebidas alcoólicas através das *keywords*: alcoólicas e bebidas, sejam extraídas erradamente empresas que, por exemplo, só vendam sumos.

No problema de remover da base de dados alguns *emails* ainda se poderia automatizar a parte de ir buscar as fontes dos *emails* acedendo através de programação a *labels* de várias contas do gmail, já que os *emails* para classificar pertenciam a várias contas diferentes. Nessa implementação a saída seriam os *emails* já classificados em ERROR, BLACKLISTED, REPLY, AUTO-REPLY e as entradas seriam os endereços de *email*, as respectivas *labels* e as respetivas palavras passe.

No que respeita à parte dos testes ao portal poder-se-ia implementar testes automáticos com recurso ao python e ao Selenium para testar automaticamente partes do portal que não sofressem alteração num período de médio a longo prazo.

Outra melhoria que poderia ser implementada seria no envio dos *emails* de *marketing* em massa. Para o envio de *emails*, a equipa de *marketing* tinha de preparar um *email* para cada campanha alterando só alguns detalhes como por exemplo, os detalhes do produto / serviço anunciado. Para além de ter de criar diferentes *emails*, era necessário coloca-los em contas de *emails* distintas. Para reduzir o tempo despendido, poder-se-ia automatizar o envio de *emails* através programação usando um modelo como base. O programa teria de receber o modelo com campos definidos para alterar, credenciais da conta de onde o *email* seria enviado, as contas dos destinatários e os campos personalizados para cada conta.

Neste estágio foi possível desenvolver capacidades de trabalho em equipa e adquirir conhecimentos nas áreas de *marketing* e de desenvolvimento de *software*. A licenciatura e a parte curricular do mestrado em Matemática foram essenciais neste estágio ao aplicar as capacidades adquiras na análise de problemas e suas especificações, sua abstração e desenvolvimento de algoritmos.

Bibliografia

[1] A. Cockburn, "Writing effective use cases", Boston: Addison-Wesley, 2001.

[2] D. Janzen e H. Saiedian, "Test-Driven Development: Concepts, Taxonomy, and Future Direction", *IEEE Computer Society*, pp. 43-50, 2005.

[3] K. Beck, "Test-driven development: by example", Addison-Wesley Professional, 2002.

[4] J. W. Ratcliff e D. Metzener, "Pattern Matching: The Gestalt Approach", Dr. Dobb's Journal, p. 46, 1988.

[5] R. W. Hamming, "Error Detecting and Error Correcting Codes", Bell System Technical Journal, vol. 29, nº 2, pp. 147-160, 1950.

[6] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", Doklady Akademii Nauk, vol. 163, nº 4, pp. 845-848, 1965.

[7] F. J. Damerau, "A technique for computer detection and correction of spelling errors", Communications of the ACM, vol. 7, nº 3, pp. 171-176, March 1964.

[8] I. Setiadi, "Damerau-Levenshtein Algorithm and Bayes Theorem for Spell Checker Optimization", Bandung Institute of Technology, Report number: X, Dezembro 2013. [Online]. Available: `https://www.researchgate.net/publication/268334497_Damerau-Levenshtein_Algorithm_and_Bayes_Theorem_for_Spell_Checker_Optimization`. [Acedido em 11 Maio 2018].

[9] M. A. Jaro, "Advances in record-linkage methodology as applied to the 1985 census of Tampa, Florida", Journal of the American Statistical Association, vol. 84, nº 406, p. 414–420., 1989.

[10] D. Șuteu, "Jaro distance", rosettacode.org, 25 Fevereiro 2018. [Online]. Available: `https://rosettacode.org/wiki/Jaro_distance`. [Acedido em 1 Maio 2018].

[11] W. E. Winkler, "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage", Proceedings of the Section on Survey Research Methods. American Statistical Association, p. 354–359, 1990.

[12] C. Newman, "Simple Mail Transfer Protocol (SMTP) Enhanced Status Codes Registry", 22 Junho 2017. [Online]. Available: https://www.iana.org/assignments/smtp-enhanced-status-codes/smtp-enhanced-status-codes.xhtml. [Acedido em 11 Maio 2018].

[13] R. Aguilar, "Using Test-Driven Development to Improve Software Development Practices", School of Computer Science, REYKJAVIK UNIVERSITY, 2016. [Online]. Available: https://skemman.is/bitstream/1946/26193/1/tdd_research_study_2016_raquelita.pdf. [Acedido em 1 Maio 2018].

[14] M. Cowles. "The Python Standard Library. Internet Data Handling", 2018 [Online]. Available: https://docs.python.org/3.5/library/email-examples.html. [Acedido em 1 Maio 2018].

[15] Baiju Muthukadan. "Selenium with Python", 2018 [Online]. Available: https://selenium-python.readthedocs.io/index.html. [Acedido em 1 Maio 2018].

[16] Kenneth Reitz. "Requests: HTTP for Humans", 2018 [Online]. Available: http://docs.python-requests.org/en/master/. [Acedido em 1 Maio 2018].

[17] Leonard Richardson. "Beautiful Soup Documentation", 2018 [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#. [Acedido em 1 Maio 2018].

[18] Alexander Bogomolny. "Distance Between Strings" [Online]. Available: http://www.cut-the-knot.org/do_you_know/Strings.shtml [Acedido em 25 Novembro 2018].

[19] Heikki Hyyrö. "A bit-vector algorithm for computing Levenshtein and Damerau edit distances", Nordic Journal of Computing, v.10 n.1, p.29-39, 2003 [Online]. Available: http://www.academia.edu/download/39402556/psc02.pdf [Acedido em 25 Novembro 2018].

[20] Ilya Ilyankou. "Comparasion of Jaro-Winkler and Ractliff/Obershelp algorithms in spell check", IB Extended Essay Computer Science, 2014 [Online]. Available: https://ilyankou.files.wordpress.com/2015/06/ib-extended-essay.pdf [Acedido em 17 Maio 2018].

[21] Santiago Marco-Sola. "Efficient Aproximate String Matching Techniques for Sequance Alignment", Universitat Politécnica de Catalunya, 2016 [Online]. Available: https://www.tdx.cat/bitstream/handle/10803/460835/TSMS1de1.pdf?sequence=1&fbclid=IwAR0vKDdSfXbREaaCs9e2ZUQ5AV8oYTB5ZD1lMC9dS5j9ciBpkpMNlWHdVVE [Acedido em 08 Janeiro 2019].