INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Protótipo Plataforma de Escrita

ANA RITA DIAS NOGUEIRA
Julho de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Protótipo Plataforma de Escrita

Ana Rita Dias Nogueira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Nuno Escudeiro

Porto, Julho de 2016

Resumo

Desde que existe, o Homem sentiu a necessidade de deixar a sua marca no mundo. Com a invenção da escrita, este passou a poder registar os seus pensamentos e ideias em algo mais duradouro do que a sua memória.

Com o evoluir do mundo, foram aparecendo novas formas de poder deixar registos escritos. Com o aparecimento de novas tecnologias e a aproximação das pessoas *online*, também a forma de registar os pensamentos mudou. Qualquer pessoa pode escrever e partilhar *online*, para que outras possam ler.

Neste momento já existem várias plataformas, cujo objetivo é facilitar a partilha de histórias criadas por pessoas que desejam partilhá-las com o mundo. No entanto, muitos destes escritores são amadores e por isso podem não ter os conhecimentos necessários para conseguir caracterizar as suas histórias adequadamente.

Estas plataformas oferecem funcionalidades de pesquisa e recomendação rudimentares.

A aplicação de técnicas de *data mining* e *text mining* podem ajudar os escritores a caracterizar as suas histórias de uma forma automática, eficiente e sem exigir conhecimentos literários. Com esta automação podemos obter dados que podem ser também utilizados para recomendar histórias e disponibilizar funcionalidades de pesquisa e recomendação, por exemplo, por personagem, género literário ou ainda por outras características.

O objetivo desta dissertação é implementar estas tecnologias numa plataforma para escrita, partilha e leitura de histórias dos mais variados géneros literários, por parte de escritores amadores e seus fãs, para que a sua vida seja facilitada, quer ao nível da escrita, com o preenchimento automático de atributos descritores nas histórias — como o género literário ou a lista de personagens— quer ao nível da sugestão de conteúdos aos utilizadores.

Palavras-chave: *Machine learning*; *natural language processing* (NLP); inteligência artificial; processamento de texto; *Text Mining*; *Data Mining*.

Abstract

Since Man exists, he felt the need to leave his mark on the world. With the invention of writing, Man was able to record his thoughts and ideas into something more lasting than memory.

As the world evolved, new ways of leaving written records began to appear. With the emergence of new technologies and the approximation of people online, the way of recording thoughts also changed. Anyone can write and share online, so that others can read.

At the moment there are already multiple platforms, whose goal is to facilitate the sharing of stories created by people who want to share them with the world. However, many of these writers are amateurs and they may not have the necessary knowledge to be able to characterize their stories appropriately.

These platforms offer rudimentary search and recommendation features.

The application of data mining and text mining can help writers to characterize their stories automatically, efficiently and without requiring literary knowledge. With this automation we can obtain data that can also be used to recommend stories and search and recommendation features like, character, literary genre or other characteristics.

The goal of this dissertation is to implement these technologies on a platform for writing, sharing and reading stories of various literary genres, by amateur writers and their fans, so that their life is facilitated, both in terms of writing, with the auto-completion of attribute descriptors in the stories—like the literary genre or the list of characters—and in terms of the suggestion of content to users.

Keywords: Machine learning; natural language processing (NLP); artificial intelligence; text processing; *Text Mining*; *Data Mining*

Agradecimentos

Gostaria em primeiro lugar de agradecer ao professor Nuno Escudeiro pelo seu apoio no desenvolvimento deste projeto, contribuindo com ideias, incentivos e orientando-me. Também gostaria de agradecer ao meu colega Filipe Couto, sem o qual esta ideia nunca teria saído do papel.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

LDA	Latent Dirichlet Allocation
TF-DIF	Term frequency-inverse document frequency
CBOW	Continuous Bag of Words
LSVM	Linear Support Vector Machine
SQL	Structured Query Language
NoSQL	Not Only SQL
XML	eXtensible Markup Language
NLP	Natural Language Processing
NER	Named Entity Recognition

Lista de Símbolos

∑	Somatório

1 Introdução

Este capítulo tem por objetivo fazer um breve resumo desta dissertação e do projeto documentado por esta.

1.1 Enquadramento

Este documento destina-se a documentar todo o desenvolvimento do projeto realizado pela aluna Ana Rita Dias Nogueira (nº. 1110494) na unidade curricular Tese / Dissertação / Estágio do Mestrado no ramo de Sistemas Computacionais do departamento de Engenharia Informática do Instituto Superior de Engenharia do Porto (ISEP) no ano letivo 2015/2016.

Este projeto foi desenvolvido em parceria com o meu colega Filipe Couto, estando este responsável pelo desenvolvimento de parte das componentes da plataforma, nomeadamente o *website,* sendo que esta componente apenas será mencionada ao longo deste documento, mas não aprofundada.

1.2 Apresentação do Projeto

Na área da escrita criativa (invenção e publicação de textos literários, utilizando a imaginação como fonte de inspiração) já existem vários *websites* bastante conhecidos e frequentados, quer por escritores, quer por leitores, dos mais variados géneros literários (Piotr Kowalczyk 2016)(Rhiannon Tuffield 2016).

Muitas destas plataformas não fornecem automação na criação de histórias, isto é, ao criar uma nova história, o utilizador terá que, para além de escrever a história propriamente dita, também preencher um pequeno formulário com, por exemplo, um resumo, idioma, entre outras. Este facto, que pode ser um pouco entediante para alguns escritores. No entanto pode ser facilmente contornado, se existir, algum processo automático que preencha os campos adicionais de caracterização de uma história (como por exemplo idioma, resumo, personagens, género literário, etc.).

O objetivo deste projeto consiste na criação de uma plataforma que preencha de forma automática estes formulários com informações relevantes, tais como género literário, resumo, lista de personagens, idioma e classificação de conteúdo impróprio. Para além disto, a plataforma, também deverá recomendar histórias aos utilizadores, utilizando para isso, as informações retiradas das histórias lidas previamente por estes.

1.3 Análise de Valor

O produto desenvolvido nesta dissertação é uma plataforma *online* para escrita e partilha de histórias.

Não é um conceito em si inovador pois já existem várias plataformas similares, no entanto, nenhuma destas fornece aos seus utilizadores opções que tornam a sua passagem pela plataforma mais cómoda e simples, nomeadamente, na procura do que ler ou ajuda na sua escrita.

Assim sendo propomo-nos a criar uma plataforma para escrita, partilha e leitura de histórias dos mais variados géneros literários, cujo objetivo é facilitar a vida dos escritores e leitores, quer ao nível da escrita — com o preenchimento automático de atributos descritores das histórias — quer ao nível da sugestão de conteúdos aos utilizadores — analisando para isso as pesquisas efetuadas.

1.4 Abordagem

Este projeto consiste na criação de uma plataforma e de toda a infraestrutura (base de dados entre outras) para a escrita, leitura e partilha de histórias *online*.

Esta infraestrutura será dividida essencialmente em três partes:

- *Website* (não tratado em detalhe nesta dissertação);
- Caracterização de histórias e recomendação das mesmas;
- Sincronização das bases de dados SQL (*website*) e NoSQL (caracterização e recomendação).

Para a recomendação de histórias, recorremos a métodos de *data mining* para sugestão de histórias aos leitores da plataforma. Já para realizar a caracterização de uma história de forma automática, são utilizadas técnicas de *text mining.* Para guardar os dados resultantes da caracterização e recomendação de histórias é utilizada uma base de dados NoSQL.

O *website* tem a sua própria base de dados (SQL). Para sincronizar ambas as bases de dados é utilizada uma aplicação, que implementa o padrão produtor/consumidor.

1.5 Resultados Atingidos

Através da implementação da abordagem anterior (Secção 1.4), foram obtidos resultados medianamente satisfatórios para a resolução deste desafio.

Apesar destes resultados, o problema não foi totalmente resolvido. Tal como foi referido na Secção 1.1, o projeto foi desenvolvido em parceria, sendo que o outro membro do grupo ficou

responsável pelo desenvolvimento do *website*. No entanto, o *website* não foi completamente terminado.

Em suma, do ponto de vista desta dissertação, todos os objetivos propostos a realizar foram terminados. Já do ponto de vista da plataforma, os objetivos não foram totalmente terminados.

1.6 Estrutura da Dissertação

Esta dissertação encontra-se organizada de acordo com os *outcomes* propostos no início da unidade curricular. Assim sendo, a estrutura será a seguinte:

1. Outcome1: Interpretar o problema
    a. Introdução
2. Outcome2: Contexto e Estado da arte
    a. Contexto
3. Outcome3: Avaliar soluções e abordagens existentes
    a. Abordagens ao Problema
4. • Outcome4: Design da solução
    a. Design da solução
5. Outcome5: Construção da solução
    a. Solução
6. Outcome6: Avaliação da solução
    a. Avaliação dos classificadores e Testes
7. Conclusão

2 Contexto

2.1 Problema

Desde os primórdios da história, que o ser Humano procura uma forma de se expressar. Com a invenção da escrita em 4000 a.C. pelos Sumérios (Luís Reis 2010), o ser humano passou a poder registar os seus pensamentos em algo mais duradouro do que a sua memória.

Com a evolução Humana, também a escrita evolui. Com a entrada das novas tecnologias na vida quotidiana e a constante aproximação das pessoas *online*, também a escrita se revolucionou. A escrita criativa e a publicação de obras, deixou de ser apenas acessível a uma elite de escritores reconhecidos, para ser viral, na medida que, qualquer pessoa pode escrever uma história e rapidamente partilhá-la com milhões de outras pessoas *online*.

Com esta aparição de novos escritores, passou a haver a necessidade de existirem plataformas em que estes escritores amadores pudessem partilhar as suas obras de forma rápida e acessível, para que possíveis interessados a lê-las conseguissem encontrar facilmente.

Atualmente já existem várias plataformas de escrita criativa com sucesso na Internet (Piotr Kowalczyk 2016). No entanto, com os avanços nas técnicas de descoberta de conhecimento, já implementadas por plataformas como *Facebook* e *Youtube*, nenhuma destas plataformas ainda implementou um sistema que possa levar mais facilmente ao leitor o conteúdo que quer ler.

Para além da ausência ou implementação rudimentar de sistemas de recomendação personalizada de histórias, estas plataformas fornecem opções de pesquisa de histórias limitadas, como por exemplo, pesquisa por nome da história ou escritor. Pesquisas por personagem, no caso de histórias de *fanfictions*, ou pesquisa por histórias num determinado idioma, podem ser inseridas se existir algum tipo de processamento e extração destas informações aquando do carregamento da história na plataforma.

Esta dissertação tem como objetivo estudar as plataformas já existentes na área da escrita criativa e propor uma solução que fomente o mercado de escrita e partilha de conteúdo literário gratuito, principalmente utilizado por escritores amadores e os seus leitores, aplicando técnica de *text* e *data mining*.

2.1.1 Conceitos Importantes

Para compreender este problema, é necessário entender alguns conceitos fundamentais, quer na área da escrita criativa, quer na área de *data* e *text mining*.

Fanfiction

Fanfictions ou *fanfics* (Cavalcanti n.d.) são histórias de ficção criadas por fãs de uma determinada série de televisão, livro, *anime*, banda, novela, entre outras, que utiliza uma ideia já estabelecida para criar uma nova história, baseada nessa mesma ideia, alterando certos acontecimentos e desfechos.

Machine Learning

Machine Learning (SAS n.d.) ou aprendizagem máquina é uma subárea da inteligência artificial dedicada ao desenvolvimento de algoritmos, cujo objetivo é aprender um determinado comportamento ou padrão a partir de exemplos, para que consigam prever comportamentos.

Big Data

Por *Big Data* (Wu et al. 2014) entende-se um grande conjunto de dados complexos, estruturados ou não estruturados, que as aplicações de processamento utilizadas normalmente não conseguem tratar. Para o tratamento deste tipo de dados, são utilizados processos de análises preditivas (*data mining*) ou outro tipo de processos mais avançados para extração de valor dos dados.

Data Mining

Data mining (Witten et al. 2011) ou extração de dados é o processo de descoberta de padrões nos dados, utilizando para isto *machine learning*, estatísticas e base de dados.

O processo de *data mining* envolve, para além de tratamento dos dados, gestão de base de dados, pré-processamento, considerações de complexidade, o pós-processamento de estruturas descobertas e visualização e atualização de informação.

Text Mining

Text mining (DELL 2015) ou extração de dados de texto é um processo de descoberta de conhecimento, que consiste na análise de texto e da consequente extração de informação deste. É uma subcategoria de *data mining* uma vez que só se aplica em situações em que os dados de *input* são documentos de texto.

Natural Language Processing

Natural Language Processing (NLP) ou Processamento de Linguagem Natural (Javier Couto 2015) é um campo da inteligência artificial, que se preocupa com a interação dos computadores com as linguagens naturais, isto é a utilização de computadores para extrair e dar significado às linguagens naturais ou humanas.

2.1.2 Requisitos

Analisando o problema enunciado anteriormente, podemos compreender quais os requisitos que esta plataforma de escrita criativa deve ter, de forma a responder às necessidades dos seus utilizadores.

Primeiro que tudo, através da análise do problema, identificamos três atores:

- **Visitante –** leitor não registado na plataforma;
- **Utilizador –** leitor/escritor registado na plataforma;
- **Administrador -** responsável pela plataforma. Para além das permissões de um utilizador normal, acrescenta ainda permissões de administração da plataforma, como por exemplo, opções para banir comentários ou histórias, responder a reclamações/sugestões, entre outros.

Identificados os atores, podemos então tentar encontrar as ações que cada um destes atores vai querer realizar na plataforma.

Desde já, podemos dividir estas ações em dois grupos: os que se relacionam com a conta em si e com as histórias.

Para as ações relacionadas com as contas (Figura 1):

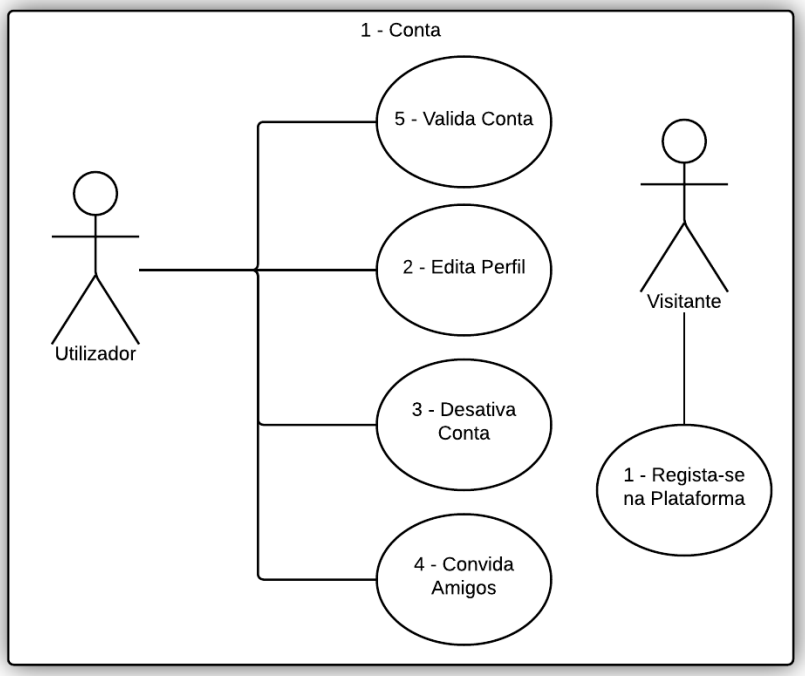

Figura 1 - Use case: domínio da conta

No diagrama anterior podemos ver quais as funcionalidades disponíveis para cada ator (em mais detalhe no Anexo 1):

- **Utilizador**:
    - Podem editar o seu perfil;
    - Podem desativar a sua conta;
    - Podem convidar pessoas.
    - Podem validar a sua tentativa de criar uma conta.
- **Visitante**:
    - Pode tentar criar conta na plataforma.

Do ponto de vista das histórias temos (Figura 2):

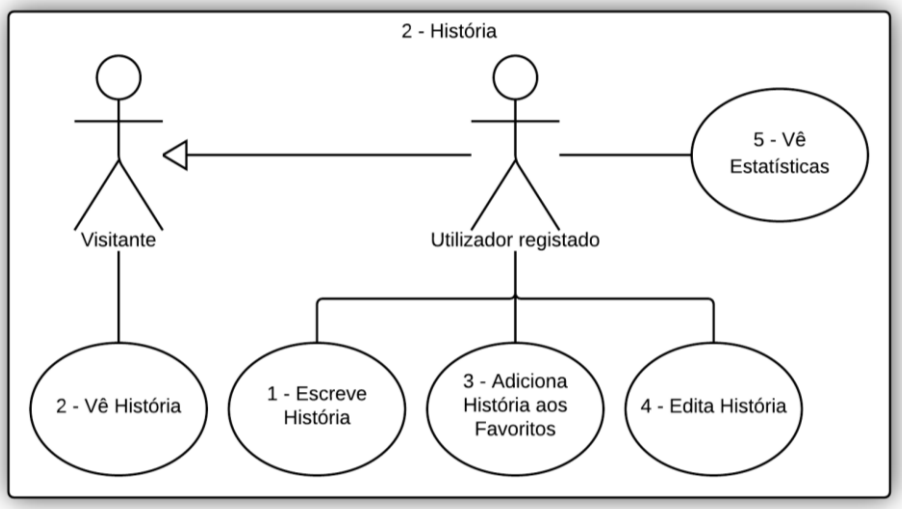

Figura 2 - Use case: histórias

Neste diagrama, podemos ver (em mais detalhe no Anexo 2):

- **Utilizador:**
    - Podem escrever histórias;
    - Podem editar as suas histórias;
    - Podem ver estatísticas relativas às suas histórias, como por exemplo, número de visualizações por mês;
    - Podem guardar as suas histórias preferidas;
    - Podem, tal como os Visitantes, ler histórias.
- **Visitante**:
    - Pode ler histórias.

Para além das ações realizadas diretamente pelos utilizadores na plataforma, temos também ações realizadas automaticamente no seguimento destas mesmas ações (Figura 3):

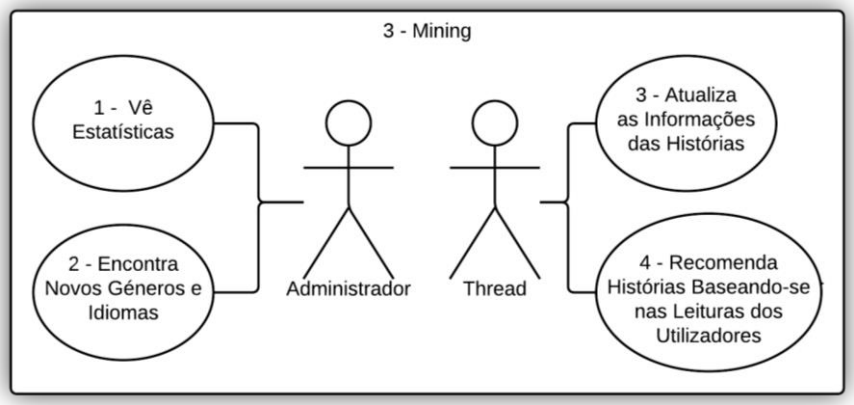

Figura 3- Use case: aplicação de *data* e *text mining*

No diagrama anterior, podemos constatar que, temos apenas dois atores: Thread e Administrador.

Isto deve-se ao facto de esta aplicação ser apenas para o uso do Administrador do *website*. Assim sendo as ações possíveis são (em detalhe no Anexo 3):

- **Administrador:**
  - Pode ver as estatísticas sobre as histórias tratadas e guardadas na base de dados;
  - Pode encontrar novos géneros literários e idiomas presentes na base de dados.
- **Thread**:
  - Analisa as histórias publicadas pelos Utilizadores e preenche os campos em branco, utilizando para isto, técnicas de *text mining*. Neste ponto, o sistema quererá, concretamente analisar as histórias de forma a encontrar:
    - Idioma em que a história está escrita;
    - Lista de personagens da história;
    - Género literário a que a história pertence;
    - Encontrar frases chave na história de forma a conseguir criar um resumo desta;
    - Analisar a história de forma determinar se o seu conteúdo é impróprio para certos leitores.
  - Analisa as histórias lidas pelo Utilizador para tentar prever o que este poderá querer ler, utilizando para isto, técnicas de *data mining*;
  - Estas ações são realizadas periodicamente pelo sistema, são despoletadas por ações realizadas pelos utilizadores registados na plataforma (leituras ou escritas de histórias) e influenciam as ações destes (preenchem informações nas histórias e recomendam-nas).

2.1.3 Restrições existentes

Para a realização deste projeto foram impostas, desde o início, restrições quer a nível técnico, quer a nível monetário:

1. O orçamento para este projeto é de 0€;
2. Todas as tecnologias a utilizar devem ser *opensource*;
3. Todas as bibliotecas devem ser estudadas em conjunto, para avaliar se a sua relação restrições/benefícios são benéficas para o projeto.

2.2 Análise de Valor

2.2.1 Proposta de Valor

Definir a proposta de valor, ou seja, definir o "conjunto de produtos e serviços que criam valor para um segmento específico de clientes" (OSTERWALDER, 2011), é uma componente importante para se ter um negócio de sucesso, uma vez que, através desta análise, se pode demonstrar a clientes e investidores como esta se diferencia da restante concorrência.

Mas para criar uma proposta de valor, é necessário primeiro definir o valor de cada um dos produtos ou serviços que a empresa possui, ou seja, se estes produtos ou serviços vão de encontro às necessidades dos possíveis clientes.

No entanto, um produto pode até ir de encontro às necessidades dos clientes, no ponto de vista da empresa, mas se o seu cliente alvo não compreender que, de facto, o produto resolve os seus problemas. Se o cliente não perceber o valor do produto, não o irá adquirir. Isto leva-nos à definição de valor percebido.

Por valor percebido compreende-se, a perceção por parte do cliente, que o benefício que retira de um determinado produto ou serviço é superior ao seu custo e por isso benéfico para si mesmo.

Podemos então olhar para este valor percebido pelo cliente de um ponto de vista longitudinal, ou seja, compreender o valor do produto tendo em conta uma linha temporal. Neste ponto de vista, podemos dividir a linha temporal do produto em quatro fases diferentes (Nicola 2016):

1. **Pré-compra –** tentar prever como os possíveis clientes percebem o produto/serviço;
2. **Compra -** valor para o cliente percebido no ponto de compra;
3. **Pós- Compra –** obtenção de resultados baseados em experiências dos clientes e fornecedores;
4. **Depois da utilização -** ponto de eliminação / venda.

Pensando então no caso concreto tratado nesta dissertação, o produto desenvolvido é uma plataforma *online* para escrita e partilha de histórias. Não é um conceito em si inovador pois

já existem várias plataformas similares, no entanto, nenhuma destas fornece aos seus utilizadores opções que tornam a sua passagem pela plataforma mais cómoda e simples, nomeadamente, na procura do que ler ou ajuda na sua escrita.

Assim sendo propomo-nos a criar uma plataforma para escrita, partilha e leitura de histórias dos mais variados géneros literários, cujo objetivo é facilitar a vida dos escritores e leitores, quer ao nível da escrita — com o preenchimento automático de atributos descritores das histórias — quer ao nível da sugestão de conteúdos aos utilizadores — analisando para isso as pesquisas efetuadas.

2.2.2 Cenários de Negócio

Quando pensamos num negócio, pensamos numa necessidade constante de resolver os conflitos que vão aparecendo no trajeto da empresa.

Estes conflitos devem ser resolvidos através de negociações. Segundo *Michael Filzmoser* e *Rudolf Vetschera* em "*A classification of bargaining steps and their impact on negotiation outcomes*" (Filzmoser & Vetschera 2008) "as negociações são processos dinâmicos em que as partes envolvidas comunicam para trocar ofertas, fazer concessões, levantar ameaças, ou de outra forma influenciar-se mutuamente, com o objetivo de chegar a um acordo" (tradução nossa) ou seja, a negociação é feita de discussões sucessivas entre os intervenientes, para que no final cheguem a um acordo.

Numa negociação não existe número limite de intervenientes, podendo haver mais que dois indivíduos ou equipas envolvidos, no entanto, na maioria dos casos em que existe conflito, a disputa tem normalmente dois intervenientes.

Negociar algo é importante pois leva a que duas partes, que inicialmente têm visões diferentes para um mesmo problema, se entendam e cheguem a uma solução viável que satisfaça ambas as partes.

Uma negociação é normalmente composta por vários problemas que estão relacionados entre si, podendo ser chamados de "grupo de problemas"(Carnevale & Pruitt 1992).

Num processo de negociação podem existir, tendo em conta as questões discutidas na mesma, quatro "desfechos" ou conclusões possíveis:

- ***Lose-Lose* ou não acontecimento de acordo:** Quando as entidades envolvidas não chegam a nenhum acordo;
- ***Win-Lose***: Quando os cenários discutidos são apenas benéficos para uma das partes;
- ***Simple compromisse* ou compromisso simples:** Quando um compromisso é definido como o meio-termo entre as entidades;
- ***Win-Win* ou negociação integrativa:** Quando as partes envolvidas encontram e aceitam uma solução que contribui de forma benéfica para ambas.

Na Figura 4 podemos ver em detalhe estes tipos de cenário de negociação. Os possíveis resultados de uma negociação podem ser entendidos em termos de espaço de utilidade conjunta (Carnevale & Pruitt 1992). Os pontos neste espaço correspondem às opções existentes para se estabelecer um grupo de problemas, sendo os pontos negros são as opções conhecidas no início da negociação e os brancos as opções que podem ser encontradas com um pouco mais de análise do problema e que serão mutuamente benéficas para ambas as partes. Os eixos dos gráficos mostram a utilidade de cada opção para a resolução do problema. As linhas a tracejado representam as fronteiras que os intervenientes não querem transpor na negociação.

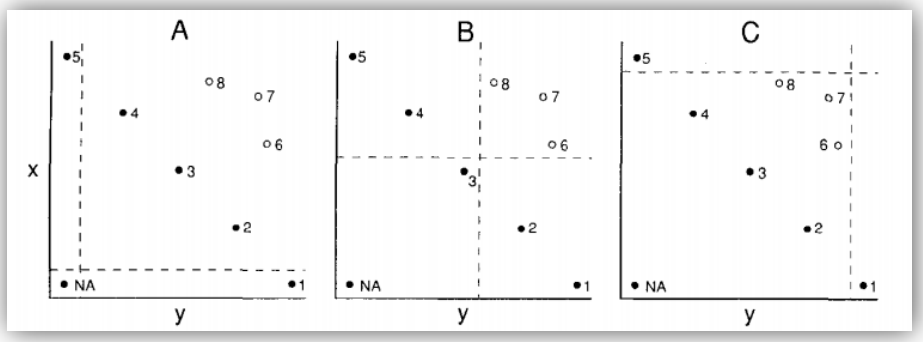

Figura 4 - Cenários de negociação (Carnevale & Pruitt 1992)

Na negociação A temos presentes quatro desfecho possíveis:

1. As partes não concordam com nada – ponto NA;
2. Apenas uma das partes sai vitoriosa (*Win-Lose*) – pontos 1 ou 5;
3. Ambas as partes chegam a um compromisso simples – pontos 2,3 e 4;
4. Situação de *Win-Win* em que ambas as partes obtêm o máximo do que poderiam concordar – pontos 6,7,8.

Já na negociação B, os pontos iniciais estão fora das fronteiras dos intervenientes: nesta situação pode não haver um acordo se as partes não dialogarem para encontrar soluções alternativas, que satisfaçam todos os intervenientes.

Na negociação C, os limites são tão elevados que nenhuma das opções é viável. Nestes casos a única opção para existir acordo é se uma ou ambas as partes diminuírem os seus limites.

Quando encaramos uma negociação e pensámos no seu desfecho, ou seja, se vamos ter uma situação *Win-Win*, *Win-Lose*, *Lose-Lose*, podemos então enquadrar estes cenários em dois tipos distintos de negociação:

- **Negociação distributiva -** associada com o cenário *Win-Lose*, isto é, o ganho de uma dos intervenientes significa a perde do(s) restante(s);
- **Negociação integrativa -** associada ao cenário *Win-Win*, onde os intervenientes, com o acordo estabelecido, adquirem uma solução que beneficia ambos.

Enquadrando os cenários de negócio, enunciadas anteriormente, no desenvolvimento desta dissertação, o cenário ideal para uma negociação seria um acordo *Win-Win* com o público-alvo, ou seja, uma negociação integrativa.

Podemos pensar por exemplo, numa funcionalidade que será implementada na plataforma: deteção de personagens numa história.

Esta funcionalidade poderia não estar implementada na aplicação e os utilizadores não teriam acesso às personagens duma história. No entanto esta funcionalidade poderia ser muito desejada pelos utilizadores.

Sendo a implementação desta funcionalidade o problema, como base da negociação podemos ter as seguintes opções:

- Os utilizadores querem esta funcionalidade e que esta funcione em 20 idiomas diferentes;
- Os programadores apenas querem desenvolver a deteção para histórias em inglês.

Depois de alguma negociação podem chegar a um compromisso simples de implementar a funcionalidade para os idiomas mais usados no *website*.

Já uma situação de *Win-Win* poderia ser a da implementação imediata da deteção para os idiomas principais e a promessa de, num futuro próximo, mais idiomas serem implementados e ainda ser adicionada a opção de pesquisa por personagem.

2.2.3 Modelo CANVAS

Num negócio é importante compreender e visualizar como será o seu modelo de negócio. Uma forma simples e fácil de compreender a influência de certas áreas têm sobre o negócio é construir um modelo CANVAS. Com o modelo CANVAS podemos mapear os principais pontos que constituem um negócio, podendo assim perceber como um negócio pode "combinar os meios para entregar valor às partes interessadas relevantes e capturar valor para a organização" (EINOV 2014).

O modelo CANVAS divide-se, essencialmente em nove partes diferentes (Nicola 2016):

- ***Customer Segments* (Segmentos de Clientes):** grupo ou grupos de pessoas que a empresa pretende servir. Estas pessoas têm necessidades ou comportamentos comuns;
- ***Value Proposition* (Proposta de Valor):** representa produtos ou serviços que irão gerar valores para os segmentos de clientes enunciados no capítulo *Customer Segments*;
- ***Channels* (Canais):** forma como os produtos ou a empresa vão chegar ao cliente;
- ***Customer Relationships* (Relacionamento com o Cliente):** forma como a empresa comunica com cada cliente ou potencial cliente;

- ***Revenue Streams* (Fontes de Receita):** como o produto ou serviço irá gerar dinheiro;
- ***Key Resources* (Recurso-Chave):** ativos necessários para fazer o Modelo de Negócio funcionar;
- ***Key Activities* (Atividades-Chave):** funções que a empresa deve desempenhar para que o Modelo de Negócio funcione;
- ***Key Partners* (Parcerias-Chave):** fornecedores e parceiros essenciais ao funcionamento do Modelo de Negócio;
- ***Cost Structure* (Estrutura de Custos):** principais custos que são necessários ao funcionamento do Modelo de Negócio.

O modelo de negócio no caso concreto deste *website* é bastante simples (Figura 5):

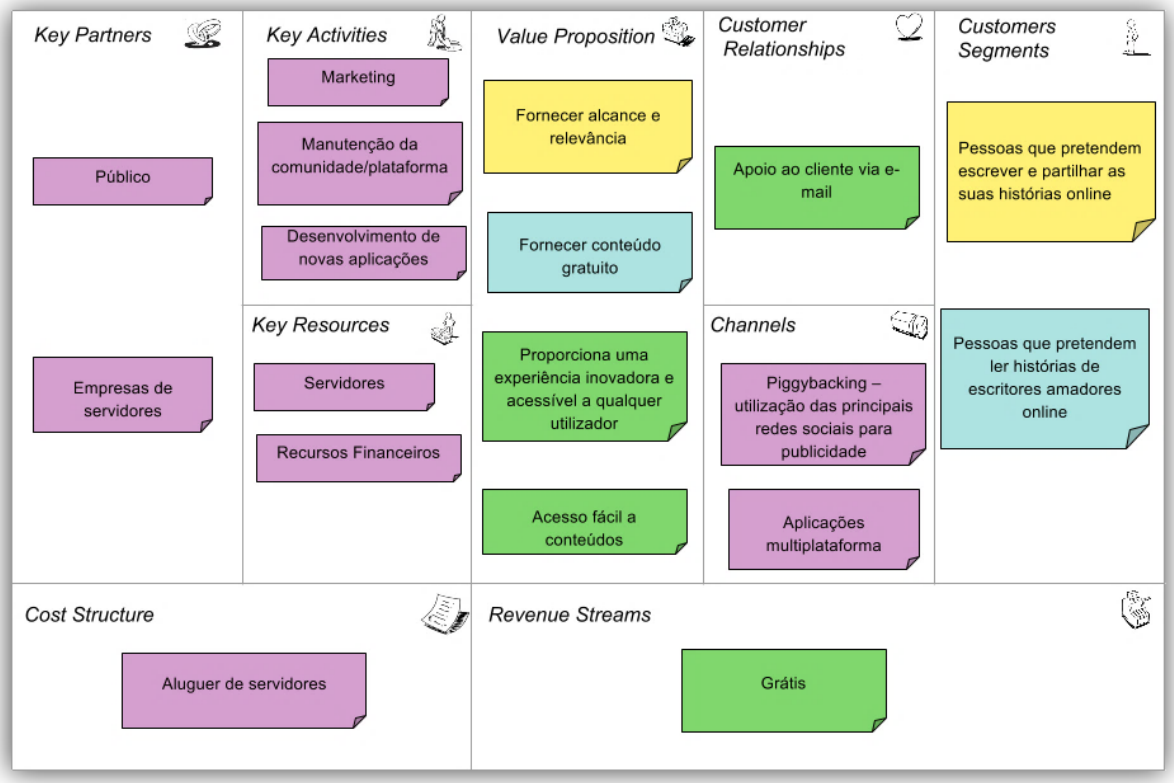

Figura 5 - Modelo CANVAS

Para este negócio temos, então, dois segmentos de clientes distintos: as pessoas que querem escrever e partilhar as suas histórias e as que as desejam ler.

Com este produto propomo-nos a:

- Dar alcance e relevância a autores desconhecidos;
- Fornecer conteúdo literário gratuito *online*;
- Proporcionamos uma experiência acessível e inovadora aos nossos utilizadores;
- O acesso a conteúdos é fácil.

O objetivo deste produto é ser grátis, logo não vai qualquer entrada de dinheiro, no entanto vamos ter custos de aluguer de servidores para alojar a plataforma.

Por ser uma plataforma grátis e não existir capital disponível para publicitar o produto, esta publicidade irá ser realizada através da utilização de plataformas já conhecidas, como o *Facebook*.

2.2.4 Criação de Valor

Como já foi referido anteriormente, ao criar um novo produto ou serviço que responda às necessidades dos clientes, estamos a criar valor.

Mas quanto valor está a ser criado? Atualmente existem vários modelos que conseguem calcular o valor criado pelo produto. Estes são (Nicola 2016):

- Teoria de Jogos;
- AHP (*Analytic hierarchy process*).

2.2.4.1 Teoria de Jogos

A teoria de jogos é uma teoria matemática que dá ênfase ao processo de tomada de decisão dos jogadores que jogam um jogo, isto é, desenvolve critérios racionais para a seleção de uma estratégia.

Um jogo começa com vários jogadores, em que cada um tem um certo número de opções ou estratégias que pode usar.

Os jogadores vão sempre trocando de estratégia, até não conseguirem trocar mais (neste caso encontraram a estratégia melhor). Quando o jogadores não trocam mais de estratégia, foi atingido o equilibro de *Nash*.

2.2.4.2 AHP (Analytic hierarchy process)

AHP ou processo de hierarquia analítica é uma técnica utilizada lidar com decisões complexas em que várias variáveis são consideradas para a seleção de alternativas. Estas variáveis são(Nicola 2016):

- Problema;
- Fatores/Critérios;
- Processo de tomada de decisão;
- Alternativas.

Tendo então definido os pontos que respondem a cada das variáveis enunciadas anteriormente, o processo de análise passa por (Nicola 2016):

1. Definir o problema e estruturar o problema num diagrama hierárquico;
2. Definir e comparar os diferentes critérios;
3. Definir as prioridades das alternativas relativamente a cada um dos critérios encontrados;
4. Atribuir um peso para cada critério, tendo em conta o objetivo;
5. As prioridades locais são multiplicadas pelos pesos dos critérios que correspondem a essa prioridade;
6. Os resultados locais são somados para encontrar a prioridade global de cada alternativa.

2.3 Plataformas Existentes

Como já foi dito anteriormente existem em funcionamento várias plataformas para escrita criativa e que contam com um grande número de utilizadores.

De acordo com o *website Ebook Friendly* (Piotr Kowalczyk 2016), baseando-se para este estudo no número de histórias publicadas, em 2015 os 15 melhores *websites* de escrita criativa, foram:

1) FanFiction
2) Quotev
3) Kindle Worlds
4) Wattpad
5) Archive of Our Own
6) Asianfanfics
7) deviantART
8) FicWad
9) Internet Archive
10) Feedbooks
11) Goodreads
12) Tumblr
13) LiveJournal
14) FictionPad
15) MediaMiner

Seguidamente iremos falar das três plataformas mais usadas na escrita e partilha de histórias e os serviços que as tornam tão populares.

2.3.1 FanFiction

FanFiction[1] (Figura 6) é considerado, de acordo com *Piotr Kowalczyk* (Piotr Kowalczyk 2016), a maior plataforma de *fanfiction* do mundo. Fundado em Outubro de 1998 pelo programador *Xing Li*, em Los Angeles e atualmente conta com cerca de dois milhões de utilizadores.

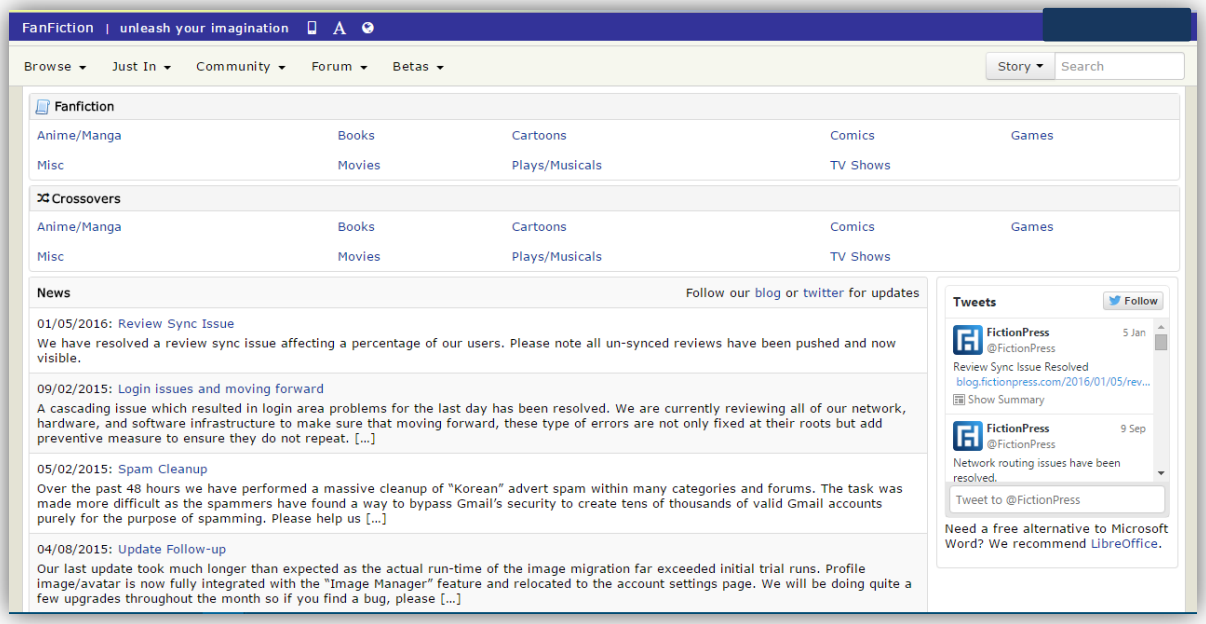

Figura 6 - FanFiction

Como foi mencionado anteriormente, a plataforma *FanFiction* é especializada em *fanfictions* de várias áreas, como por exemplo, livros, *anime*, banda de desenhada, filmes, entre outros.

Esta plataforma não disponibiliza uma aplicação *mobile* aos seus utilizadores, no entanto possui uma versão do seu *website* para dispositivos móveis.

2.3.2 Quotev

Quotev[2] (Figura 7) é outra plataforma bastante utilizada para a escrita e partilha de histórias. Não se conhecem os seus autores ou a data em que foi fundado mas sabe-se que originalmente a plataforma era conhecida por *Quizazz* e só mais tarde mudou o seu nome para *Quotev*.

[1] https://www.fanfiction.net/
[2] http://www.quotev.com/

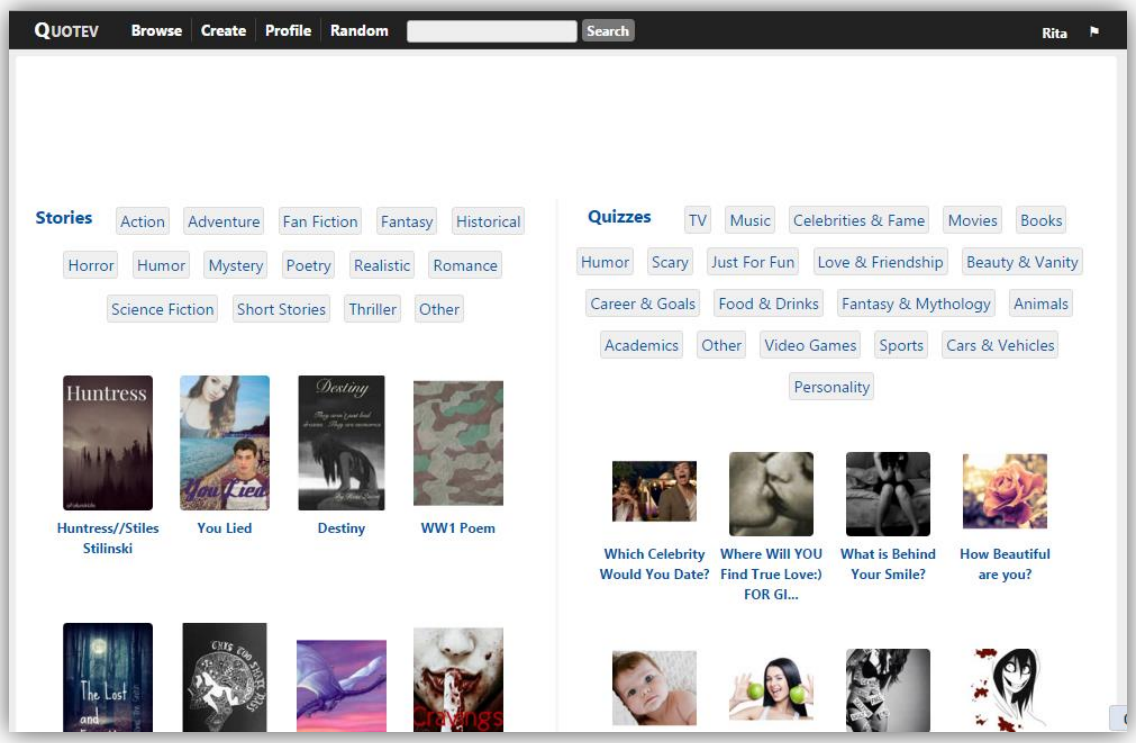

Figura 7 - Quotev

Este *website* tornou-se conhecido não só pelas suas histórias, mas também porque oferece a opção de criar *quizzes*, votações, entre outros.

2.3.3 Kindle Worlds

A plataforma *Kindle Worlds* (Figura 8) foi criada pela empresa *Amazon* em Maio de 2013. Esta plataforma foi criada com um objetivo um pouco diferente das anteriores: vender histórias de *fanfiction* dos mais variados tipos, devidamente licenciadas pela própria *Amazon*.

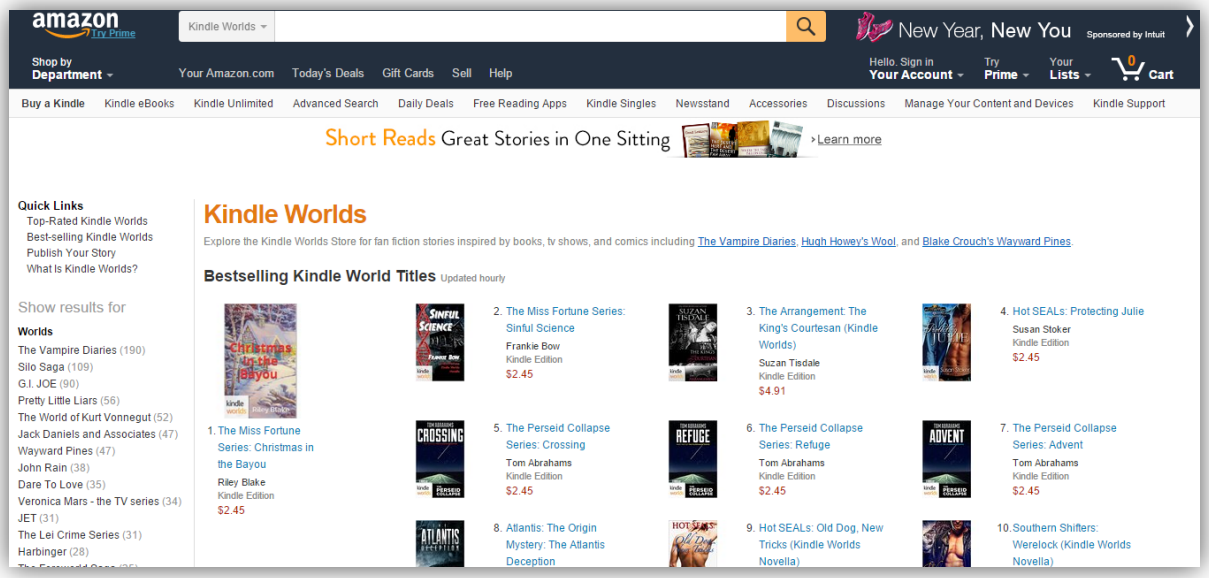

Figura 8 - Kindle Worlds

Nesta plataforma os autores de *fanfiction* e outros tipos de histórias, tem a oportunidade de obter lucro com as suas obras, desde que estas se encaixem nos parâmetros de licenciamento. Estas obras são posteriormente vendidas em formato de *e-book* no *website*.

2.4 Tecnologias Relevantes

O processo de *data* e *text mining* envolve a estruturação dos dados de entrada, normalmente analisando-os, adicionando recursos linguísticos e removendo outros, utilizando para isto padrões, avaliação e interpretação dos dados de saída. Isto pode ser realizado através um conjunto de métodos de aprendizagem, estatísticos e de técnicas de *machine learning.*

Os dados conseguidos através de *text mining* podem ser utilizados para *business intelligence*, análise exploratória de dados, pesquisas, investigação, *marketing*, entre outros.

2.4.1 Algoritmos Utilizados

Para extrair informações de qualquer tipo de dados, já existem várias metodologias no mercado que ajudam a tratar os dados retirados, quer de informações de utilizadores, pesquisas, imagens, entre outras, e processá-las de forma a, com esta informação, criar estatísticas, prever outras informações, como por exemplo, gostos de um utilizador, o que este poderá querer ler/ver, entre outros.

Dos algoritmos que serão apresentados a seguir, o TF-IDF e Word2Vec (Secções 2.4.1.1 e 2.4.1.2 respetivamente) são utilizados para estruturar os dados. Os algoritmos SVM, *Naive Bayes* e *Decision Tree* (Secções 2.4.1.3, 2.4.1.4 e 2.4.1.5) para classificar estes dados.

Já os algoritmos *K-means*, *Gaussian Mixture Model* e *Latent Dirichlet Allocation* (Secções 2.4.1.6, 2.4.1.7 e 2.4.1.8 respetivamente) são utilizados para aglomerar dados.

2.4.1.1 TF-IDF

Term frequency-inverse document frequency (TF-IDF) é um algoritmo de vectorização de *features* ou características, muito usado no tratamento de documentos, uma vez que calcula a importância de um determinado termo no documento.

A importância de um termo é calculada através da proporcionalidade do termo relativamente ao documento, isto é, quanto mais vezes o termo for encontrado num documento, maior será o seu valor para o algoritmo TF-IDF.

Este valor é depois compensado, utilizando o inverso da frequência desse termo no documento, para que palavras de utilização mais comum, como os artigos definidos "o,a,os,as", que aparecem muito frequentemente em todos os documentos do corpus, não sejam identificados como mais relevantes num texto em detrimento a outras palavras que de facto lhe dão sentido e têm potencial discriminante mais acentuado.

Para o cálculo da importância de um termo são utilizadas duas fórmulas diferentes. A primeira calcula a importância de um termo, através do inverso da sua frequência (Spark n.d.)(1):

$$IDF(t, D) = log \frac{|D| + 1}{DF(t, D) + 1} \quad (1)$$

A segunda fórmula calcula a medida TF-IDF do termo (Spark n.d.)(2):

$$TFIDF(t, d, D) = TF(t, d) \,.\, IDF(t, D) \quad (2)$$

2.4.1.2 Word2Vec

Word2Vec, ao contrário do anterior, não tem uma implementação única, uma vez que pode utilizar um de dois modelos possíveis: *Continuous Bag of Words* (CBOW) *ou Skip-gram*.

O modelo *Skip-gram*, é uma generalização do modelo *n-gram*[3], em que os componentes desse *n-gram* (palavras) não têm que estar em sequência no texto. Algumas palavras podem ser avançadas (*skip*).

Assim sendo, podemos definir um *Skip-Gram* como *n-gram*, com um tamanho **N** de palavras por sequência e que pode saltar **K** palavras.

[3] Sequência contígua de N palavras.

Analisando a frase “Em Portugal chove pouco no Alentejo” utilizando o modelo *Skip-gram* obtemos as seguintes sequências: “Em chove”, “Portugal pouco”, “chove no” e “pouco Alentejo”, sendo 2 o número de palavras por sequência e 1 o número de palavras que podem ser ignoradas.

O modelo *Continuous Bag of Words* funciona de forma diferente, dividindo os documentos em n-*grams*, mas desta vez em *n-grams* de uma palavra só.

A principal diferença entre os modelos está na forma como calcula a importância de um termo num documento.

O modelo *Skip-gram* calcula a importâncias dos termos utilizando um termo (palavra) e atribuindo aos termos à sua volta (o seu contexto) diferentes pesos, sendo que quanto mais próximo um termo estiver do termo utilizado para o cálculo, maior será o seu peso.

Já o modelo *Continuous Bag of Words* calcula a importância duma palavra através da sua frequência no contexto.

2.4.1.3 Linear Support Vector Machine (Linear SVM)

O algoritmo *Linear Support Vector Machine* ou Máquina de Vetores de Suporte Linear é um algoritmo de classificação utilizado para realizar previsões, de acordo com os dados que lhe são dados durante o seu treino. Este algoritmo processa-se em duas fases distintas.

A primeira fase é fase de treino: durante este treino o algoritmo aprende várias classes[4] distintas.

Durante a classificação propriamente dita, o algoritmo classifica um novo documento de acordo com o que foi treinado, ou seja, atribui ao novo documento a categoria aprendida mais similar a este.

2.4.1.4 Naive Bayes

O algoritmo *Naive Bayes* é um algoritmo de classificação probabilístico, ou seja, é capaz de prever a probabilidade da distribuição dos dados nas categorias, em vez de apontar qual a categoria em que o dado se encaixa melhor. Esta probabilidade é calculada através da aplicação do teorema de *Bayes* (3)(Lobo 2007) :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (3)$$

Em que:

- P(A) e P(B) são as probabilidades marginais de A e B;
- P(A|B) e P(B|A) são probabilidades condicionais, que é calculada assumindo que B é verdadeiro, no primeiro caso, e que que A é verdadeiro, no segundo.

[4] Elemento do domínio da classificação

Para este algoritmo cada característica é totalmente independente do vetor de características onde esta estava inserida.

Analisando isto duma perspetiva diferente, podemos pensar num documento como um objeto **X**. Podemos caracterizar este objeto com uma cor, um tamanho e uma forma. O algoritmo *Naive Bayes* analisa de forma independente a cor, o tamanho e a forma, sendo que cada uma destas características contribui de forma independente para a probabilidade de ser o objeto **X**.

2.4.1.5 Decision Tree Learning

O algoritmo *Decision Tree Learning* utiliza como modelo árvores de classificação. Nesta árvore as folhas representam as classes aprendidas e os ramos representam as conjunções de características dessa classe.

Este algoritmo é um algoritmo de classificação que escolhe a solução ótima em cada etapa. Isto quer dizer que o algoritmo vai decidindo a cada iteração qual a melhor solução para essa iteração e através dessa solução vai prevendo novos nós para a árvore, até que não consegue prever uma solução melhor do que a anterior.

2.4.1.6 K-means

O algoritmo *K-means* é um dos mais conhecidos algoritmos usados para *clustering*. Este funciona através da fixação de um número de *clusters* e nestes definir os seus centros, um para cada *cluster*. Os centros devem ser definidos de forma a obter os resultados pretendidos, uma vez que cada centro produz um resultado diferente. Portanto, é boa prática posicionar os centros dos *clusters* o mais distante possível entre eles.

Depois de fixados o número de *clusters* e os seus centros, os dados são processados e associados ao centro mais próximo do seu valor.

Seguidamente, os centros são recalculados e os dados são novamente associados aos centros mais próximos do seu valor. Este ciclo continua até que os dados se mantenham associados ao mesmo centro em que se encontravam na volta anterior. Para recalcular o valor dos novos centros é utilizada a seguinte fórmula (4) (Azad Naik n.d.):

$$v_i = \left(\frac{1}{c_i}\right) \sum_{j=1}^{c_i} x_i \qquad (4)$$

Em que c_i representa o centro do *cluster* i.

Em suma, o algoritmo *K-means* tem como objetivo determinar os *clusters* (soma das distâncias de cada ponto no conjunto para o centro) da seguinte equação (5) (Azad Naik n.d.):

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (||x_i - v_j||)^2 \quad (5)$$

Em que:

- $||x_i - v_j||$ - distância Euclidiana entre x e v;
- 'c_i' - número de instâncias no cluster I;
- 'c' – número de centros

2.4.1.7 Modelo de Mistura Gaussiana

Outro algoritmo de *clustering* existente é o modelo de mistura Gaussiana ou *Gaussian Mixture Model*. Este modelo consiste na representação de uma distribuição de pontos, em que estes pontos são calculados a partir das sub-distribuições Gaussianas.

Este algoritmo calcula as probabilidades de ocorrência através da soma dos componentes Gaussianos (6)(Reynolds 2008):

$$p(x|\lambda) = \sum_{i=1}^{K} w_i \, g(x|\mu_i \Sigma_i) \quad (6)$$

Em que:

- x – vetor de dados
- w_i - peso das distribuições normais
- $g(x|\mu_i \Sigma_i)$ – Densidade Gaussiana

O cálculo da densidade Gaussiana é realizado através da fórmula (7):

$$g(x|\mu_i \Sigma_i) = \frac{1}{(2\pi)^{D/2} \, |\Sigma_i|^{1/2}} \exp\{ -\frac{1}{2} \, (x - \mu_i)' \Sigma_i^{-1} (x - \mu_i) \} \quad (7)$$

Em que:

- μ_i – Média do vetor
- Σ_i - Matriz de covariância

*2.4.1.8 L*atent Dirichlet Allocation*(LDA)*

O último algoritmo de *clustering* apresentado é o *Latent Dirichlet Allocation* (LDA) que, ao contrário dos algoritmos apresentados anteriormente, é específico para o tratamento de texto. Este algoritmo representa os documentos como misturas de tópicos, em que cada

tópico tem associado palavras com uma certa probabilidade. O funcionamento deste algoritmo pode ser explicado através da distribuição conjunta da mistura de tópicos **θ**, com um conjunto **D** de temas **z** e um conjunto **N** de palavras **w**, como podemos ver na seguinte equação(8) (Blei et al. 2000):

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta) \quad (8)$$

O algoritmo LDA tenta encontrar, a partir dos documentos, os tópicos com mais probabilidade de ter gerado a coleção.

2.4.2 *Frameworks* de *Text Mining*

Atualmente, a aplicação de metodologias de *text mining* já não passa pela implementação dos algoritmos de raiz: já existem várias *frameworks* que implementam os principais métodos de *clustering*, extração e classificação.

2.4.2.1 Apache Spark

Inicialmente desenvolvido no laboratório *AMPLab* na Universidade da Califórnia, *Berkeley*, e mais tarde doado à *Apache Software Foundation*, *Apache Spark*[5] é uma *framework opensource* utilizada para processar grandes volumes de dados (*Big Data*).

Esta *framework* foi construída para ser rápida, fácil de usar e ser capaz de realizar análises sofisticadas, fornecendo implementações em R, Java, Scala e Python.

Dentro desta *framework*, existe a biblioteca *MLlib*, que tem por objetivo tornar a aprendizagem máquina mais escalável. Esta biblioteca tem várias implementações dos algoritmos mais comuns de classificação, regressão, *clustering*, filtragem, entre outros.

2.4.2.2 Apache Mahout

A *framework Apache Mahout*[6], também pertencente à *Apache Software Foundation*, é outra *framework opensource* existente para aprendizagem máquina e que se foca na filtragem, agrupamento e classificação de dados.

Para além de algoritmos para *text mining*, em Java e Scala, o *Apache Mahout* também fornece bibliotecas Java para operações de álgebra linear e estatística.

[5] https://spark.apache.org/
[6] http://mahout.apache.org/

2.4.3 Outras Frameworks e Bibliotecas

Para além de *frameworks* apresentadas anteriormente, que são utilizadas para a classificação de texto, também existem várias bibliotecas e *frameworks* disponíveis para outros tipos de análise aos textos, como por exemplo idioma ou classificação morfológica de palavras.

2.4.3.1 Stanford CoreNLP

Stanford CoreNLP[7] é uma *framework* integrada, cujo objetivo é tornar a aplicação da análise linguística de textos mais acessível. Esta framework integra muitas das ferramentas de NLP desenvolvidas pelo grupo de NLP de *Stanford*, incluindo o *part-of-speech* (POS) *tagger*, reconhecedor de entidades (NER), análise de sentimentos, entre outras.

As suas análises fornecem os blocos fundamentais para a criação de aplicações de compreensão de texto de mais alto nível.

2.4.3.2 OpenNLP

A biblioteca *OpenNLP*[8], da *Apache*, apresenta um conjunto de ferramentas baseadas em *machine learning* para o processamento de textos escritos em linguagem natural.

Esta biblioteca suporta as tarefas mais comuns de NLP, tais como tokenização, segmentação de frases, *part-of-speech* (POS) *tagger*, entre outras. Estas tarefas são geralmente necessárias para construir serviços mais avançados de processamento de texto.

2.4.3.3 JavaFX

JavaFX [9]é uma biblioteca para a criação de aplicações tanto do tipo *desktop*, como aplicações do tipo *Rich Internet Applications*[10] (RIAs), que podem ser executadas numa grande variedade de dispositivos. Esta biblioteca tem como principal objetivo substituir o *Java Swing* como biblioteca *standard* para a criação de aplicações em *Java*.

2.4.3.4 Cybozu Language Detector

A biblioteca *Cybozu Language Detector*[11] é uma biblioteca *opensource* em Java e cujo principal objetivo é detetar os idiomas em que os textos são escritos.

Esta biblioteca tem 99% de precisão (Shuyo Nakatani n.d.) na deteção de idiomas e fá-lo para mais de quarenta idiomas diferentes. O idioma é detetado utilizando filtros Bayesianos e os perfis de idiomas utilizados são gerados a partir da base de dados da *Wikipedia*.

[7] http://stanfordnlp.github.io/CoreNLP/
[8] https://opennlp.apache.org/
[9] http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm
[10] Aplicações *Web* desenhadas para oferecer as mesmas características e funções normalmente associadas com aplicações de *desktop*.
[11] http://developer.cybozu.co.jp/archives/oss/2010/10/language-detect.html

2.5 Preparação de Experiências

De forma a conseguir compreender se as funcionalidades implementadas estão de acordo com o que é desejado, foram realizados um conjunto de testes, em quatro etapas diferentes:

1. Aquando da implementação dos algoritmos, foram realizados testes a estes de forma a escolher os melhores (Capítulo 6);
2. Numa primeira etapa foram feitos testes unitários a cada uma das funcionalidades implementadas (Secção 5.1.10);
3. Numa segunda fase, foram realizados testes de integração para compreender se a aplicação(ões) estão a guardar e copiar os dados das bases de dados corretamente (Secções 5.1.10 e 5.2.1);
4. Na fase final foram realizados um conjunto de testes de sistema, analisando a aplicação(ões) do ponto de vista do utilizador final, para compreender se esta(as) responde(m) aos casos de uso pensados na 2.1.2(Capítulo 7).

Mais a frente será explicado em detalhe como e com que objetivos serão realizados os testes enunciados.

3 Abordagens ao Problema

Para o desenvolvimento da plataforma de escrita criativa, começámos por analisar as abordagens possíveis, optando por aquela que nos pareceu mais adequada para resolver este problema em concreto.

Para o *website*, o paradigma será o de cliente-servidor, em que teremos vários clientes e um ou mais servidores. Dependendo do número de clientes, pode ser necessário ter um ou mais servidores ativos para responder aos clientes em tempo útil.

Para além do *website*, existem também duas aplicações independentes. Uma destas aplicações (aplicação de análise automática) tem a responsabilidade de processar as histórias para extrair informações e recomendá-las, através de técnicas de *text* e *data mining* respetivamente. A segunda aplicação (aplicação para sincronização) é responsável por sincronizar as bases de dados.

3.1 Aplicação para Análise Automática

3.1.1 Implementação de Raiz vs. Utilização de *Framework*

Como referido na Secção 2.4.2, existem várias *frameworks* na área do *text mining*. Estas *frameworks*, para além de fornecerem implementações de vários algoritmos de *clustering*, extração e classificação, também são construídos de forma a serem escaláveis e a terem o menor impacto possível na performance geral da máquina.

Quando se implementa uma aplicação semelhante a esta, em que é necessário implementar uma série de algoritmos, existe sempre a dúvida entre implementá-los de forma independente, isto é, codifica-los de raiz seguindo instruções de peritos, ou optar por utilizar *frameworks* (no caso de existirem), já conhecidas e com bom *feedback* na comunidade e que talvez possam resolver o problema com uma melhor performance do que uma implementação própria.

Tendo em conta que o tempo necessário para estudar e implementar de raiz vários algoritmos, quer sejam de *clustering*, extração ou classificação de características, só por si, poderá ocupar quase a totalidade do tempo previsto para realizar esta dissertação, e uma vez que para além da implementação destes algoritmos, também existirá uma componente na aplicação de descoberta de conhecimento ao nível das leituras dos utilizadores (recomendações), que não irão necessariamente utilizar estes algoritmos, foi decidido que a melhor abordagem seria utilizar uma framework para a realização dos processos de *text mining*.

3.1.2 Framework para *Text Mining*

Estando estabelecido que para realizar extração de informação será utilizada uma *framework*, o próximo passo será escolher qual a melhor framework mais adequada ao nosso problema.

Na Secção 2.4.2 foram apresentadas duas *frameworks* existentes para realizar *data mining*: *Apache Spark* e *Apache Mahout* (Landset et al. 2015).

Estas *frameworks*, apesar do seu propósito ser o mesmo, têm diferenças ao nível do seu funcionamento, que pode levar, tendo em conta os requisitos do sistema, a optar por uma e não por outra.

Um ponto a ter em atenção na escolha da *framework* é a quantidade e qualidade da documentação existente sobre a *framework*, quer seja documentação oficial fornecida pelos responsáveis desta, quer sejam fóruns de discussão ou trabalhos relacionados. Isto é importante pois pode vir a ser uma grande ajuda na resolução de possíveis problemas que possam aparecer durante a implementação deste módulo.

Analisando nesta perspetiva, as duas *frameworks* têm alguma documentação disponível, no entanto o *Apache Spark* fornece, no seu *website,* um conjunto de explicações de como implementar de forma simples, os seus algoritmos, assim como outro tipo de explicações de como utilizar outros módulos da *framework*. O *Apache Mahout* não possui explicações tão acessíveis quanto o *Apache Spark*.

Outro ponto importante é a rapidez de execução das ações da *framework*, isto é, como o processo de *clustering*, extração e classificação são processos com muitas iterações e por isso demorados, é necessário que o sistema seja o mais rápido possível, para dar resposta em tempo útil.

A *framework Apache Mahout* escreve para o disco em cada uma das iterações, o que pode tornar a *framework* mais lenta, em comparação ao *Apache Spark*, que evita escrever em disco.

Tendo em conta os pontos referidos anteriormente, a *framework Apache Spark* parece ser a melhor *framework*, de entre as duas, para dar respostas às necessidades do sistema.

3.1.3 Linguagem de Programação

Tendo definido a *framework* a usar, o próximo passo é definir qual a linguagem de programação a utilizar.

Neste caso específico, temos a limitação da aplicação para análise automática só poder ser codificada numa das linguagens de programação em que a *framework* está definida. Como já foi dito na Secção 2.4.2.1, a framework *Apache Spark* tem implementações para *Java*, *Scala*, *Python* e R.

Destas quatro linguagens de programação, para três delas (*Scala, Python* e *R*) seria necessário despender tempo a estudar e aprender a linguagem.

Por uma questão de gestão de tempo, foi decidido utilizar *Java* como linguagem de programação a utilizar, uma vez que já existe uma experiência anterior com a linguagem.

3.1.4 Algoritmos a Implementar

A seleção de algoritmos, devido à escolha da utilização de uma *framework* para a implementação da descoberta de conhecimento, está restringida aos algoritmos que esta implementa.

Na Secção 2.4.1 foram enunciados vários algoritmos de *clustering*, extração e classificação. Estes algoritmos são alguns exemplos da lista de algoritmos implementados pelo *Apache Spark*.

Estes algoritmos foram escolhidos, não só pelas suas propriedades (propriedades estas enunciadas no *website* do *Apache Spark*), mas também pela reputação que estes têm enquanto algoritmos.

As razões que levaram as escolher estes algoritmos são:

- ***K-means*** – na literatura é um algoritmo bastante falado e utilizado para agrupamento de dados;
- **Modelo de Mistura Gaussiana** – maior flexibilidade e precisão nas estatísticas subjacentes aos dados (Yogesh Raja n.d.);
- **LDA** – por ser um algoritmo exclusivo para tratar texto, é um algoritmo a implementar;
- **TF-IDF** – este algoritmo classifica as palavras encontradas num texto de acordo com o número de vezes que aparece nesse texto;
- ***Word2Vec*** – este algoritmo agrupa palavras semelhantes, o que ajuda na criação de novos padrões;
- **SVM** – este algoritmo já foi utilizado durante a unidade curricular de *Projeto/Estágio (PESTI)* para deteção e classificação de imagens de automóveis. Pelos bons resultados obtidos nesta análise, este era um algoritmo quase obrigatório a implementar;
- ***Naive Bayes*** – algoritmo dito como rápido a classificar (Zhang 2004);
- ***Decision Tree Learning*** - este algoritmo é bastante usado e fácil de compreender.

Cada um destes algoritmos tem propriedades únicas, que podem ser uma mais-valia para a classificação das histórias em relação ao seu conteúdo (se é impróprio para certos leitores) e ao seu género literário.

3.1.5 Outras *Frameworks* e Bibliotecas a Utilizar

Para além da *framework* utilizada para *text mining*, existe uma grande panóplia de outras *frameworks* e bibliotecas que podem dar resposta aos outros desafios desta aplicação, como por exemplo, a deteção e recomendação de idioma (análise das histórias lidas por um utilizador, mais concretamente o idioma predominante nas suas leituras e a partir dessa informação recomendar ao utilizador histórias escritas no mesmo idioma), deteção de resumo e personagens, recomendação por semelhança de títulos.

No caso da deteção do idioma, esta poderia ser realizada através da utilização dos métodos enunciados na Secção 3.1.4. No entanto existe já uma biblioteca *Java* disponível e que tem uma precisão muito maior àquela que poderíamos conseguir utilizando os algoritmos anteriores: *Cybozu Language Detector*.

Essa biblioteca é descrita no seu *website*, como tendo uma precisão de 99% para mais de 40 idiomas (Secção 2.4.3.4) e por este facto é uma biblioteca a utilizar.

Para a deteção do resumo (encontrar os nomes e verbos mais frequentes numa história, para com as frases a que estas palavras pertencem, criar um resumo) e para a deteção de personagens duma história (procurar os nomes próprios presentes no texto que não são países, cidades, organizações, ente outras) existem várias bibliotecas que processam linguagem natural e que classificam morfologicamente as palavras (*part-of-speech tagger*). Esta característica é precisamente o que é necessário para encontrar as palavras-chave do texto para criar o resumo e a lista de personagens.

Para além desta classificação de palavras, a biblioteca *Stanford CoreNLP* também fornece um reconhecedor de entidades (NER – *Named Entity Recognition*), que classifica os nomes encontrados no texto como pessoas, cidades, organizações, entre outras. O único problema deste reconhecedor é que só reconhece entidades em textos em inglês. Por isso para os nomes em português será utilizado o *part-of-speech tagger* para reconhecer nomes.

Para a recomendação de títulos, existem duas soluções possíveis: utilizar os algoritmos de *text mining* para prever as próximas possíveis leituras dos utilizadores, a partir do título, ou utilizar um algoritmo que determine a semelhança de títulos.

A primeira opção não é válida pois, para classificar textos, é necessário ter pelo menos dois tipos de dados (por exemplo verdadeiros e falsos), para treinar os algoritmos. Com as leituras do utilizador só temos um tipo de dados (verdadeiros).

Por causa deste facto será usado um algoritmo para determinar semelhança de títulos, mais concretamente a distância de *Levenshtein* (Michael Gilleland n.d.).

Este algoritmo calcula a distância de uma *String* a outra através das transformações que são necessárias fazer a uma para obter outra. Quanto mais semelhantes as *Strings*, menor a distância entre elas (em detalhe na Secção 5.1.5).

A recomendação por personagens e por idioma são realizados através da análise dos dados obtidos na deteção (lista de personagens e o idioma) e da sua conjunção com o número de leituras dos textos recomendados. Os textos recomendados por idioma e personagens são os mais lidos de entre os textos com uma personagem em concreto ou com um determinado idioma.

3.1.6 Servidor vs. Aplicação *Standalone*

Decididos todos os pontos anteriores, coloca-se uma questão: criar uma aplicação independente, exclusiva para realizar descoberta de conhecimento sobre os dados, ou implementar a descoberta de conhecimento na aplicação servidor.

A implementação dos algoritmos de *data* e *text mining* diretamente na aplicação servidor pode tornar a manutenção mais simples, uma vez que concentra numa aplicação toda a lógica subjacente à plataforma. Infelizmente, como já foi dito anteriormente, a realização de processos de descoberta de conhecimento é pesada, o que pode tornar a resposta do servidor a pedidos que nada têm a ver com a descoberta de conhecimento, mais lentos. Para além disto e apesar de tornar a manutenção mais simples, qualquer manutenção da aplicação de descoberta de conhecimento que requeira interromper o seu funcionamento, vai também interromper a aplicação principal.

Já a implementação dos algoritmos de *data* e *text mining* numa aplicação separada, cuja única função será proceder a análise e tratamento de informação, tem a desvantagem de ter que a sua base de dados separada da base de dados principal. No entanto tem a grande vantagem de poder ser completamente separada da aplicação principal e por isso não interfere em nada com esta. Para além disso, a sua manutenção em nada prejudicará o servidor e a respostas aos clientes, isto é, se existir o caso em que a aplicação tenha que ser momentaneamente interrompida para manutenção, o servidor continuará a funcionar normalmente, apenas deixará de existir o preenchimento automático de campos das histórias e recomendação destas.

Tendo em conta o que foi referido, a melhor escolha será a de implementar uma aplicação independente.

3.2 Aplicação para Sincronização das Base de Dados

Tendo em conta que existem duas bases de dados distintas (uma NoSQL para as pesquisas e tratamento de textos e uma SQL para guardar todos os dados referentes aos dados do *website*), existe uma necessidade de as sincronizar.

Para esta sincronização existem três soluções possíveis:

- Sincronização inserida no servidor;
- Sincronização inserida na aplicação para análise automática;
- Criar uma aplicação para realizar o sincronismo.

Numa fase inicial podemos excluir a primeira opção pois o servidor não deve ter conhecimento da base de dados NoSQL, uma vez que não a irá utilizar.

A aplicação para análise automática, aplicando as funcionalidades propostas, será uma aplicação pesada só por si (tendo em conta que só a utilização do *Apache Spark* consome bastantes recursos da máquina). Portanto, adicionar-lhe mais uma funcionalidade que terá que ser executada periodicamente e que irá utilizar recursos que podem ser preciosos para outras funcionalidades mais críticas, poderá afetar a sua performance.

Por estes motivos, a melhor solução será a de criar uma aplicação separada, que irá exclusivamente tratar da sincronização da base de dados.

3.3 Aplicação Servidor e Cliente

Tendo sido decido na Secção 3.1.3 que a linguagem de programação da aplicação para análise automática será em *Java*, por uma questão de homogeneidade a aplicação servidor será também em Java.

Já para a implementação da aplicação cliente foi decido utilizar *javascript* puro, sem utilização de qualquer *framework*. Esta escolha deveu-se ao facto das *frameworks*, no caso do *javascript*, tornarem o *website* mais lento (Sam 2015).

Esta escolha deve-se ao facto de se pretender implementar uma arquitetura da aplicação cliente simples, tendo só a função de fazer pedidos, recebê-los e mostrá-los, deixando toda a lógica para o servidor. Assim, não será totalmente necessário utilizar uma *framework* (Capítulo 4).

3.4 Abordagem Geral

Depois de analisadas as abordagens possíveis para os problemas e escolhidos os mais adequados, a plataforma terá a seguinte configuração (Figura 9):

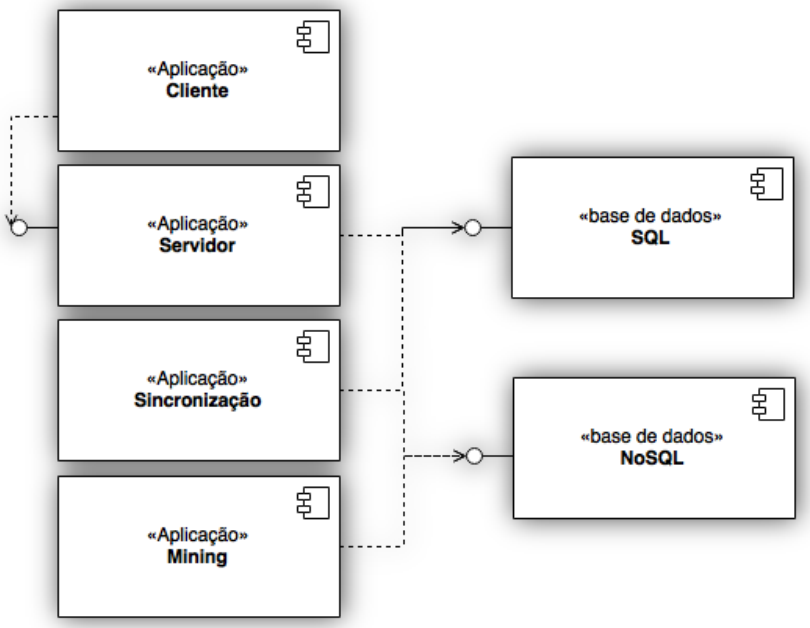

Figura 9 - Diagrama de componentes

4 Design da Solução

4.1 Servidor

Para a criação de um *website*, é necessário existir um servidor onde albergá-lo.

No caso concreto da plataforma de escrita, em que, para além da resposta a pedidos das aplicações clientes e acessos a bases de dados, existem também aplicações de suporte ao servidor (aplicação para análise automática e a aplicação de sincronização das base de dados), uma abordagem possível será a utilização de máquinas virtuais. Assim todas as aplicações podem estar a funcionar em paralelo e, caso exista necessidade, poderão ser adicionados novos servidores, sem que seja necessário mais *hardware*.

Tendo em conta isto, a arquitetura proposta é a seguinte (Figura 10):

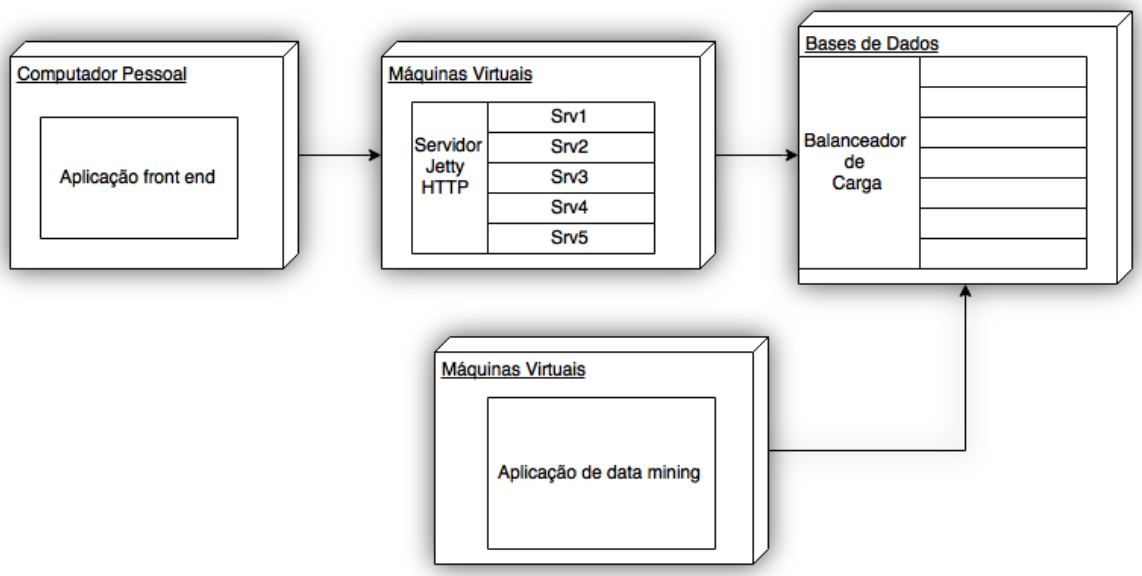

Figura 10 – Diagrama de Implementação (*Deployment View*)

4.1.1 Base de Dados

Uma vez determinado como vai ser a arquitetura do servidor, o próximo passo é definir que tipo de base de dados a usar.

Para a plataforma estão planeadas duas bases de dados distintas: uma relacional (MySQL), para guardar dados como contas, histórias, visualizações, entre outros, e uma outra não relacional (NoSQL) em que serão guardadas, entre outras informações, cópias das histórias para realizar *data* e *text mining*.

Para a base de dados SQL, a estrutura proposta (Figura 11) pode ser dividida em três partes:

- Tabelas referentes às histórias: *genre, genre_aliases, story_genres, stories,chapter, stories_language, language, characters* e *story_colab*;
- Tabelas referentes ao utilizador: *users, sessions* e *user_profile*;
- Tabelas referentes às atividades do utilizador em relação às histórias (ler, escrever e histórias recomendadas): *story_views* e *story_recommendation*.

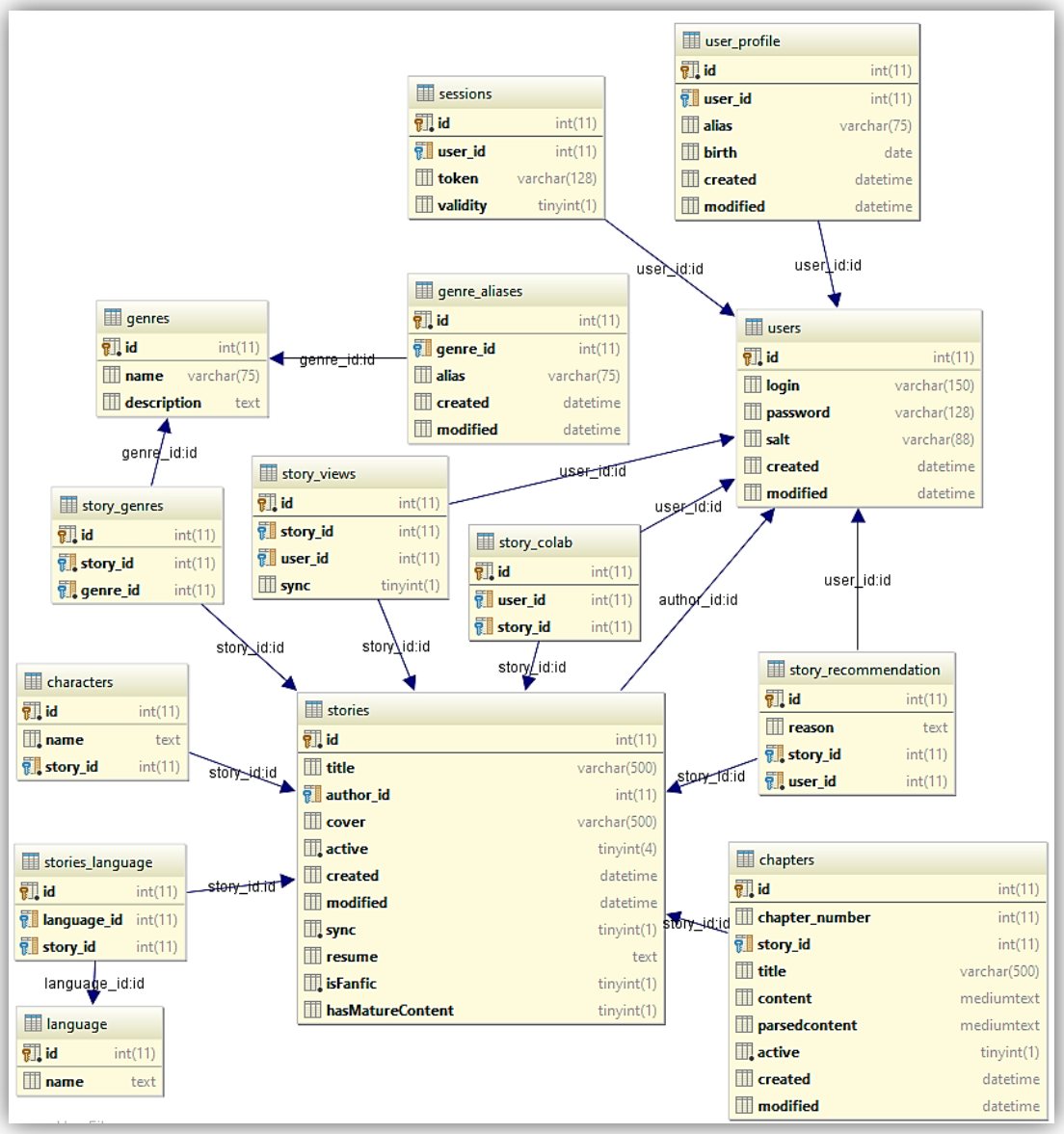

Figura 11 - Modelo de base de dados SQL

Esta base de dados será maioritariamente utilizada para armazenamento dos dados dos utilizadores e para pesquisas.

Como foi dito anteriormente, irá existir uma segunda base dados (NoSQL) cuja única função é guardar uma cópia de tabelas chave para a análise das histórias.

Como Richard K. Lomotey e Ralph Deters referem em *"Data Mining from Document-Append NoSQL"*(Richard K. Lomotey and Ralph Deters 2012), estas bases de dados foram criadas para dar resposta ao problema que é acomodar uma grande quantidade de dados não estruturados. Estas bases de dados são as recomendadas para quem trabalha com descoberta de conhecimento pois "são distribuídas, não relacionais, projetadas para armazenamento de dados em larga escala e para o processamento paralelo de dados massivos num grande número de servidores" (tradução nossa) (Moniruzzaman & Hossain 2013).

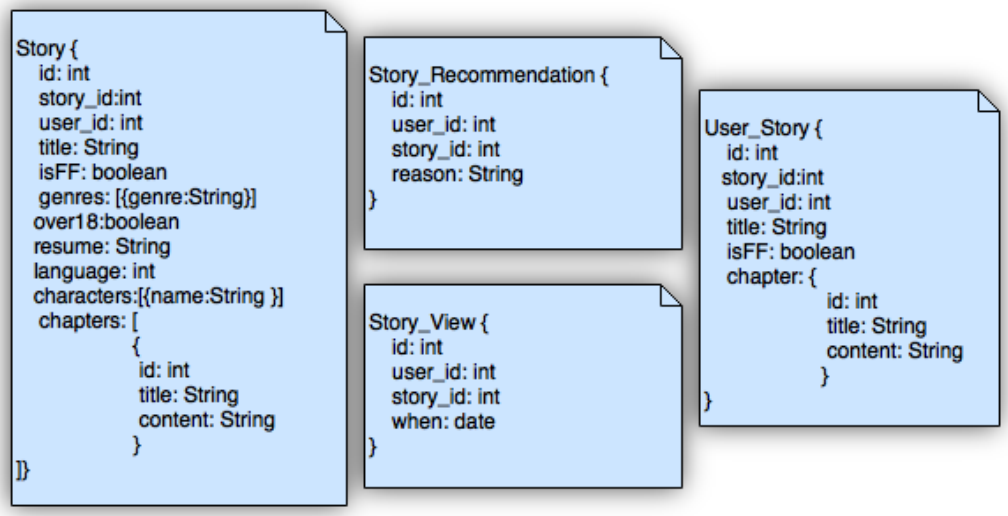

Figura 12 – Modelo de base de dados NoSQL

Por esta razão, a estrutura proposta para a base de dados NoSQL (Figura 12), passa pela criação de quatro tipos de documentos, que irão albergar os dados necessários ao processamento da descoberta de conhecimento ao nível das histórias:

- **User_Story**: guarda a informação das histórias clonadas da base dados SQL, antes de serem tratadas;
- **Story**: guarda as histórias depois de terem passado por um processamento de *text mining*, para preencher os campos em falta e é sincronizada com a tabela equivalente na base de dados MySQL;
- **Story_View**: guarda a informação clonada das histórias visualizadas por utilizadores. O id do utilizador aponta para uma entrada na base de dados MySQL e o id da história aponta para uma história que se encontra em ambas as bases de dados. Com este id, pode-se facilmente encontrar a história em causa e proceder à análise de, por exemplo, o título, o idioma, as personagens, para poder preencher o documento seguinte;

- **Story_Recommendation**: ao contrário dos outros documentos, este é gerado primeiro na base de dados NoSQL, utilizando as tabelas *Story_View* e *Story* para inferir quais histórias pode o utilizador querer ver e é depois sincronizado com a sua tabela homónima, na base de dados SQL.

4.2 Aplicação para Análise Automática

4.2.1 Dataset

Uma componente muito importante quando se realiza *data* e *text mining* sobre qualquer tipo de dados, é ter um *dataset* com dados controlados, isto é, um conjunto de instâncias em que são conhecidos os seus atributos *a priori*, para que seja possível treinar o classificador.

Para construir um *dataset*, é necessário recolher um grande volume de instâncias similares às que se prevê encontrar na plataforma.

No caso da aplicação para *data* e *text mining* ou análise automática, o conjunto de instâncias é chamado de *corpus* (conjunto de documentos de texto).

O *dataset* será organizado por idiomas; para cada idioma existirá uma coletânea de géneros literários, de acordo com a estrutura da Figura 13.

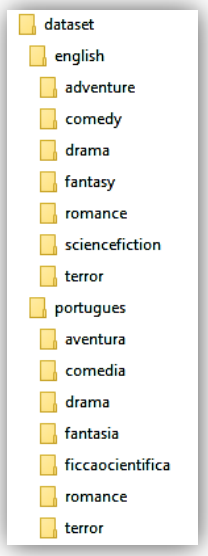

Figura 13 – Estrutura do dataset

Em cada um dos géneros literários mostrados na imagem anterior, deverão existir entre 50 a 100 histórias, sendo que este número será apenas um ponto de partida para o *dataset*. Quando maior o número de histórias presentes no *dataset*, melhor serão os resultados obtidos com os algoritmos.

4.2.2 Arquitetura

A aplicação para análise automática terá, para o caso do *text mining* duas fases distintas (Figura 14): a fase de treino e a fase de classificação.

A primeira fase em que será utilizado o *dataset* para treinar os classificadores (a verde na Figura 14). Esta fase de treino é executada uma única vez.

A segunda fase (a azul na Figura 14) é a fase responsável classificar as histórias criadas pelos utilizadores, utilizando para isso os classificadores treinados anteriormente. Em cada iteração também são aplicados outro tipo de deteções utilizando para isso técnicas de NLP (*Natural Language Processing*). Esta fase é repetida ao longo do tempo, estando constantemente à procura de novas histórias para tratar.

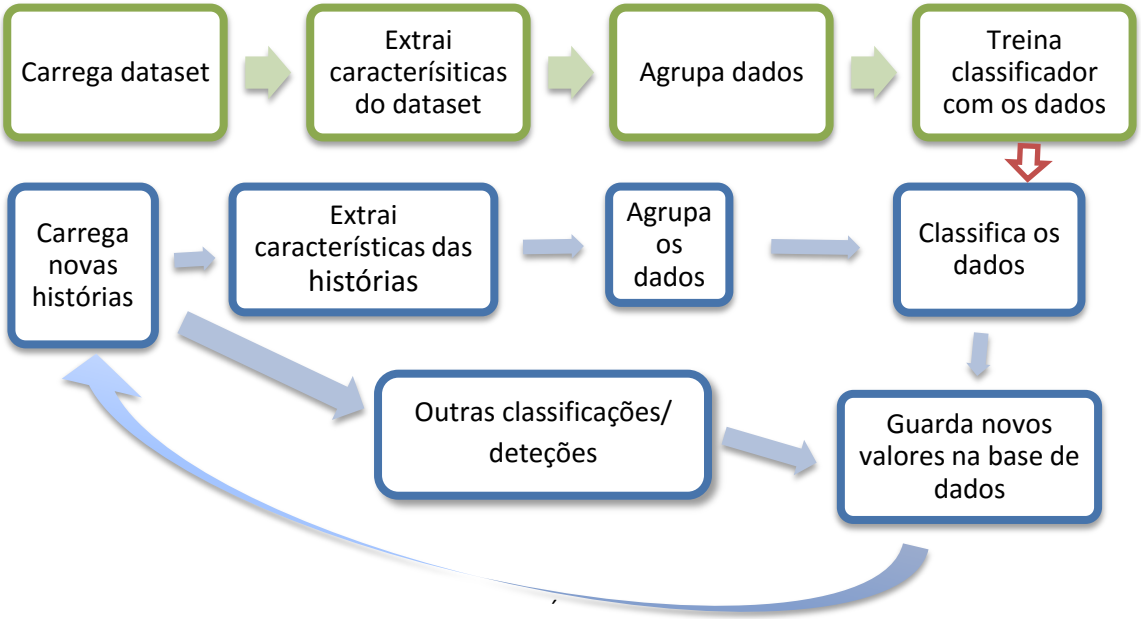

Figura 14 – Funcionamento da aplicação (*text mining*)

Já para o caso das recomendações aos utilizadores, existirá apenas uma fase. Nesta fase, a aplicação irá carregar e analisar as leituras do utilizador, tendo em conta as três formas de recomendação existentes: recomendação por personagens, idioma e título.

Para cada uma destas recomendações, a aplicação irá procurar na base de dados as histórias com características semelhantes às lidas anteriormente e recomendar as que determinar serem as mais semelhantes.

Assim sendo e, como podemos ver nas próximas figuras, a aplicação será composta essencialmente por cinco *packages* que iram dar resposta a estas fases: *package DataMining* (Figura 15), *TextMining* (Figura 16), *TextProcessing* (Figura 17), *Clustering* (Figura 18) e base de dados (Figura 19).

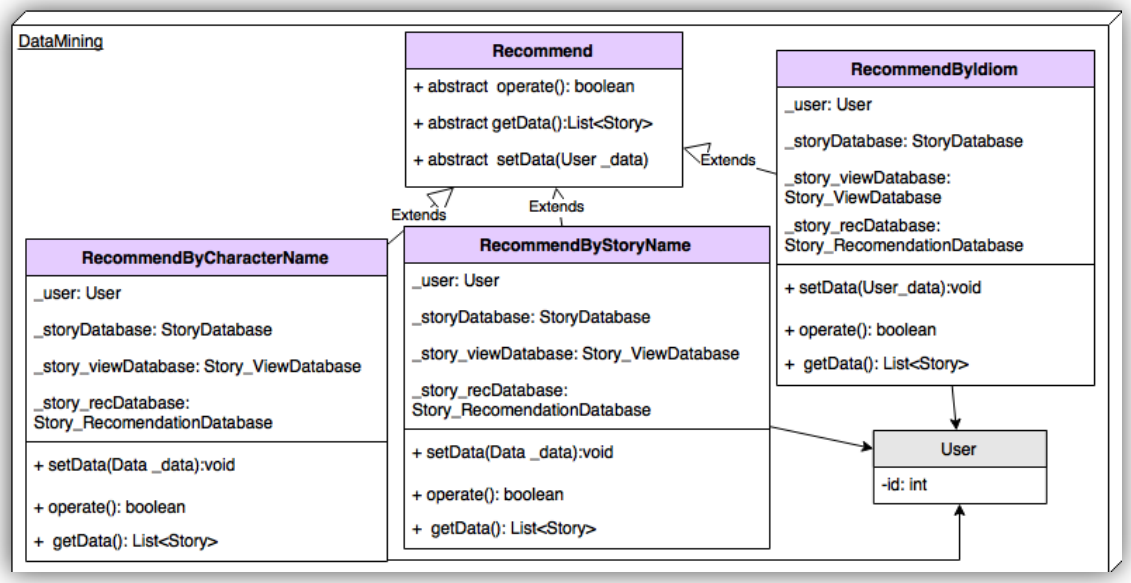

Figura 15 - Arquitetura da aplicação (*Data Mining*)

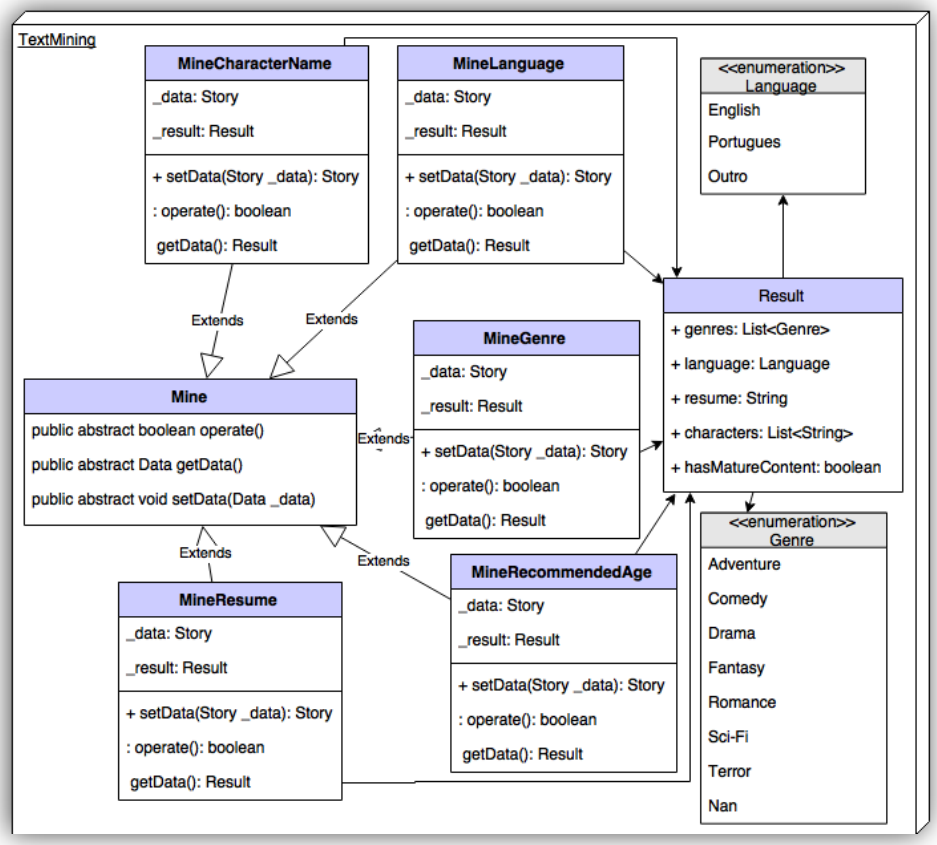

Figura 16 - Arquitetura da aplicação (*Text Mining*)

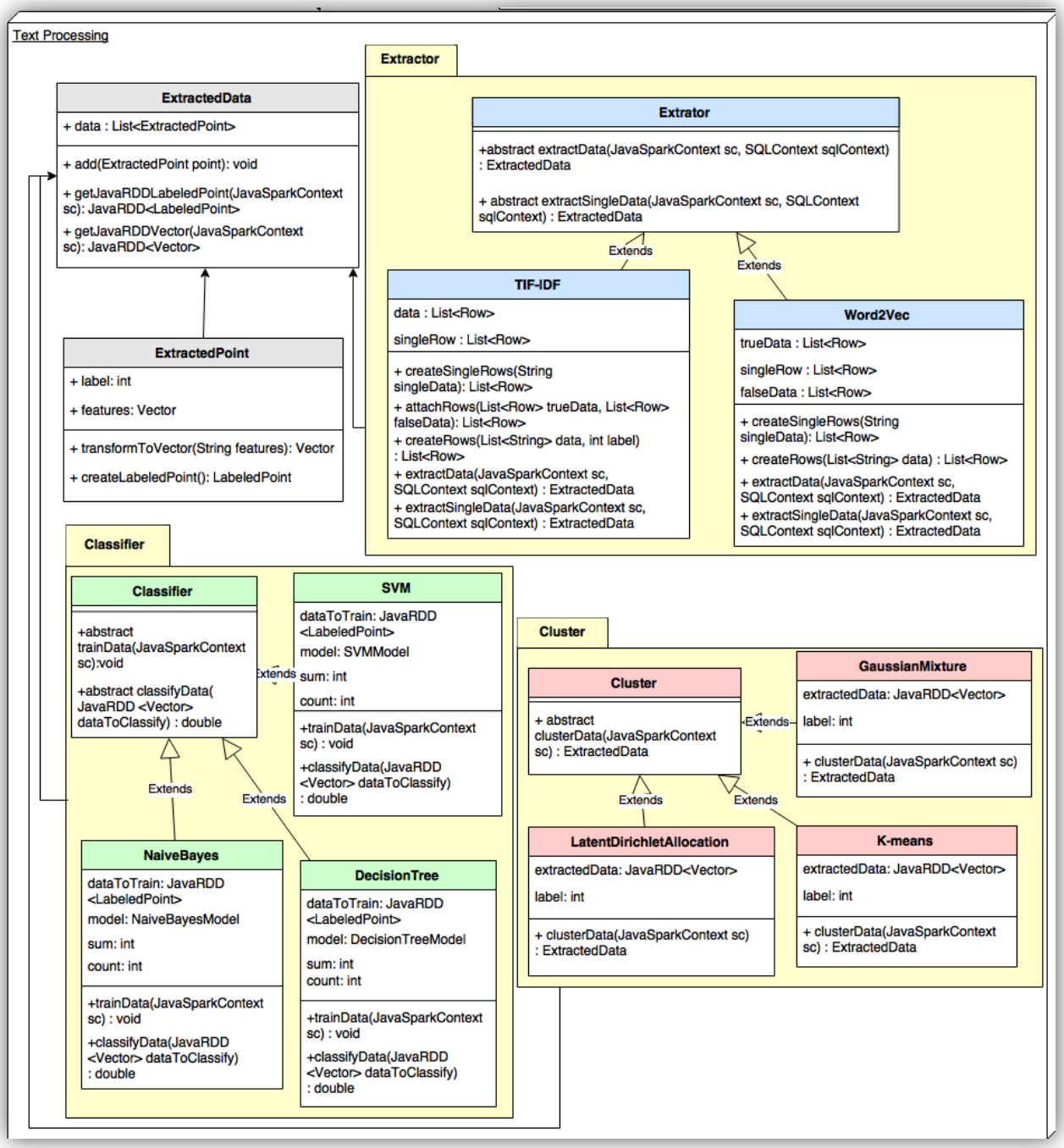

Figura 17 - Arquitetura da aplicação (Algoritmos para *Text Mining*)

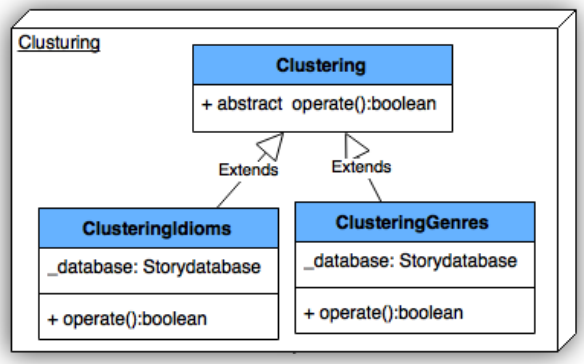

Figura 18 - Arquitetura da aplicação (*Clustering*)

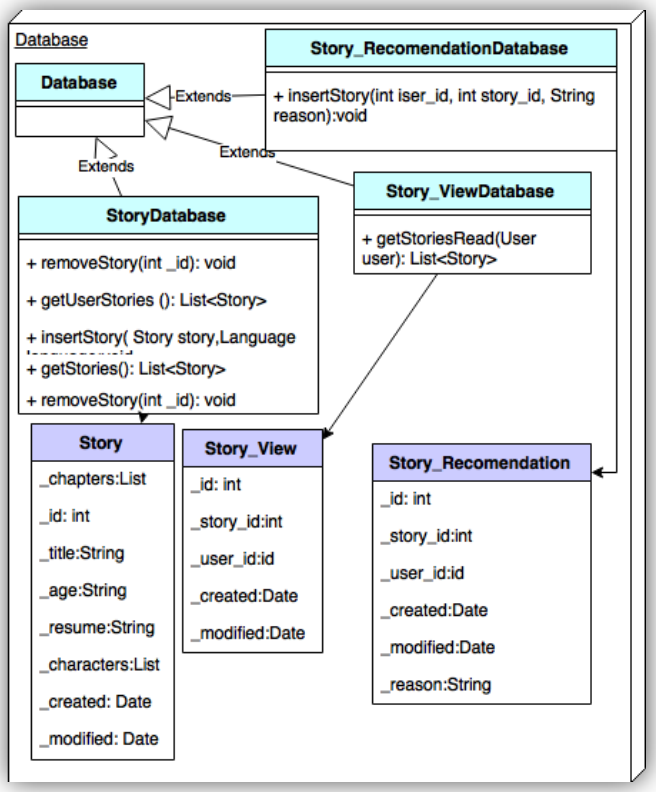

Figura 19 - Arquitetura da aplicação (Base de Dados)

O *package TextMining* é composto por uma superclasse (*Mine*) e pelas suas subclasses (*MineCharacterName, MineLanguage, MineGenre, MineResume* e *MineRecommendedAge*) e é o responsável pelo processamento dos dados das histórias recém-criadas ou editadas. Algumas destas classes utilizam classes do *package TextProcessing* (por exemplo a classe *MineGenre*), enquanto outras implementam técnicas de NLP para obter informações dos textos (por exemplo a classe *MineCharacters*).

O *package DataMining* funciona de forma semelhante: é também composta por uma superclasse (*Recommend*) e por várias subclasses (*RecommendByStoryName, RecommendByIdiom* e *RecommendByCharaterName*). É o *package* responsável pelas recomendações de histórias aos utilizadores.

Já o *package Clustering* que é o responsável pela descoberta de novos géneros literários e idiomas e utiliza o *package TextProcessing*, mais concretamente os algoritmos de *clustering* para, através dos textos guardados na base de dados, inferir novos dados.

Todos estes *packages* (exceto o *TextProcessing*) utilizam o *package Database,* quer para leitura de dados (*Text Mining* e *Clustering*) quer para atualização/inserção de dados (*Text Mining* e *Data Mining*).

Este tipo de arquitetura é benéfico para a implementação desta aplicação, uma vez que muito facilmente se pode inserir novos algoritmos ou tentar descobrir outras informações nos textos, basta estender, no primeiro caso uma das superclasses do *package TextProcessing* e no segundo ou a superclasse *Recommend*, *Mine* ou *Clustering*.

4.3 Aplicação para Sincronização das Base de Dados

Tal como já foi dito anteriormente, foi criada uma aplicação à parte para sincronizar as bases de dados MySQL e NoSQL.

Esta aplicação sincroniza em dois sentidos diferentes, isto é, certas tabelas serão sincronizadas no sentido a azul ou direita - esquerda (da base de dados NoSQL para a MySQL) na Figura 20 e outras no sentido a vermelho ou esquerda - direita (da NoSQL para a MySQL).

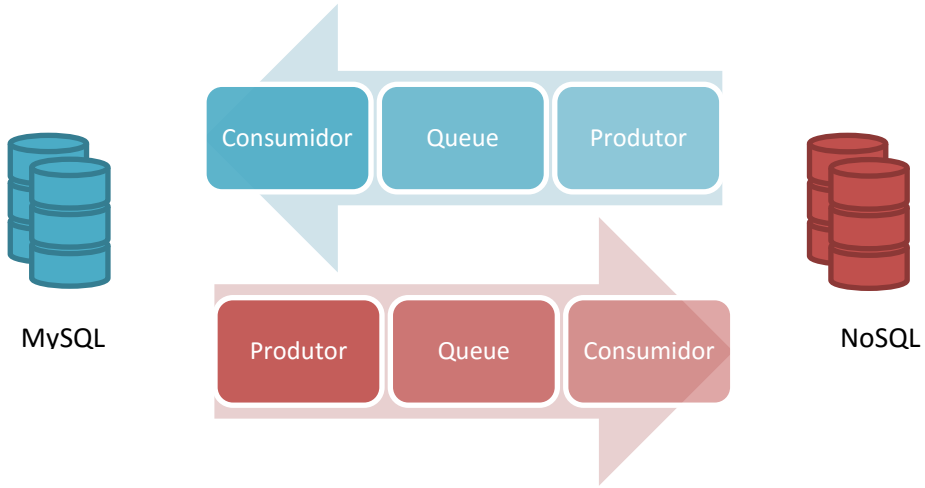

Figura 20 - Funcionamento da aplicação de sincronização

Tal como podemos ver na Figura 20, para a sincronização das bases de dados será aplicado um padrão de produtor/consumidor, com duas *queues* associadas. A utilização deste padrão, em detrimento, da utilização de apenas uma *Thread* para copiar diretamente os dados de uma

base de dados para outra, melhora a performance da aplicação. As pesquisas nestas tendem a demorar mais tempo que as inserções e edições, e por isso, se existir necessidade, podem ser adicionados mais produtores e manter o número de consumidores.

Assim sendo, a arquitetura da aplicação de sincronização terá o seguinte aspeto (Figura 21):

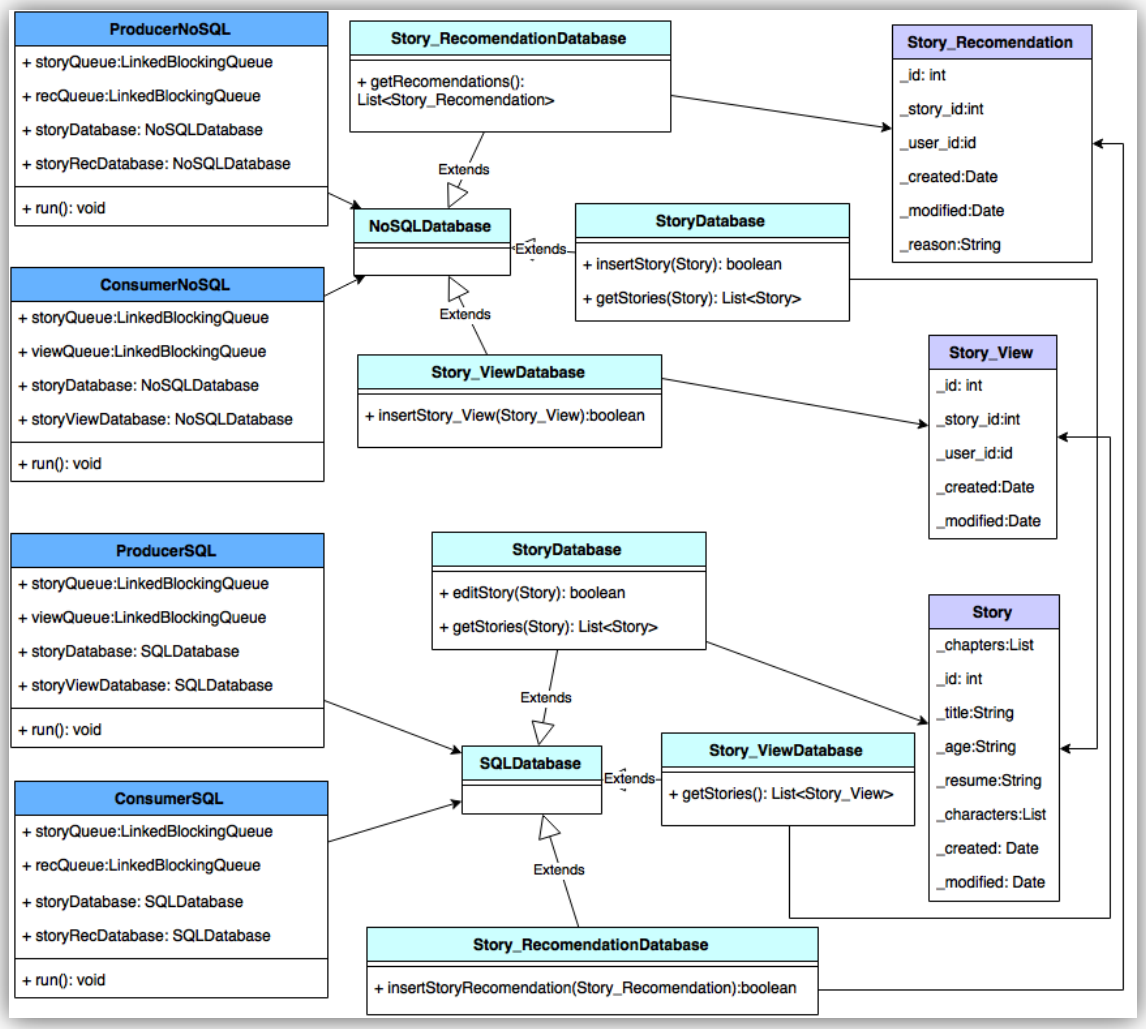

Figura 21 - Arquitetura da aplicação de sincronização

4.4 Aplicação Servidor e Cliente

Para o caso da plataforma de escrita criativa, o servidor foi desenhado de forma a conter toda a lógica em si mesmo (Figura 22 e Figura 23), deixando à aplicação cliente a única responsabilidade de fazer pedidos e mostrá-los. Este tipo de arquitetura é benéfico para a adição futura de outro tipo de clientes, como por exemplo Android, IOS e Windows (em suma aplicações móveis). Isto fará com que a criação de um novo tipo de cliente torna-se mais fácil e sem necessidade de modificar o servidor para receber este novo tipo de cliente.

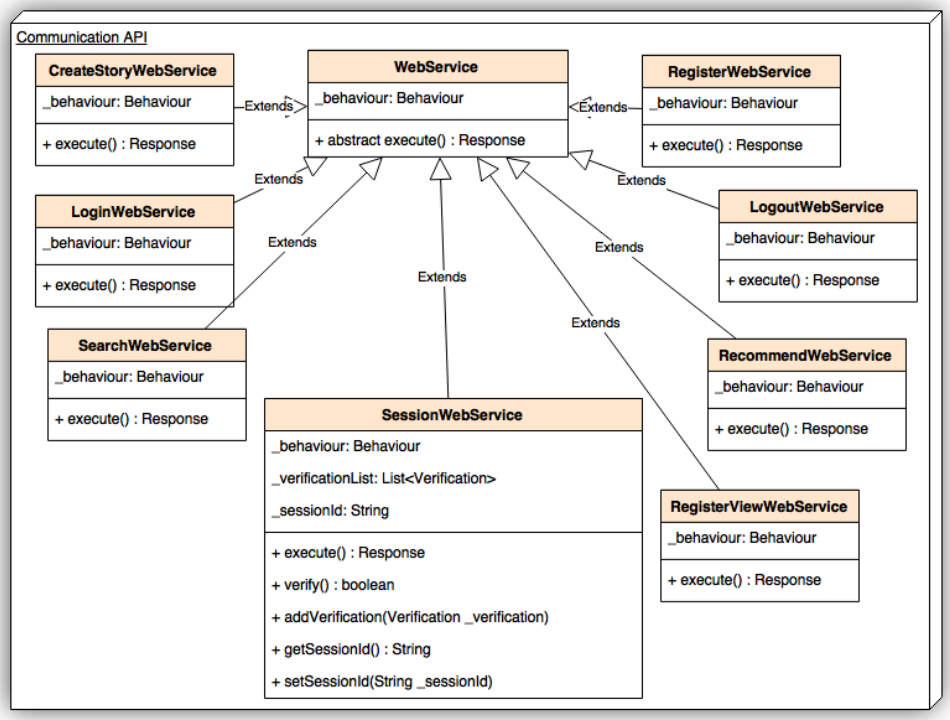

Figura 22 - Diagrama de classes da aplicação servidor (Comunicação)

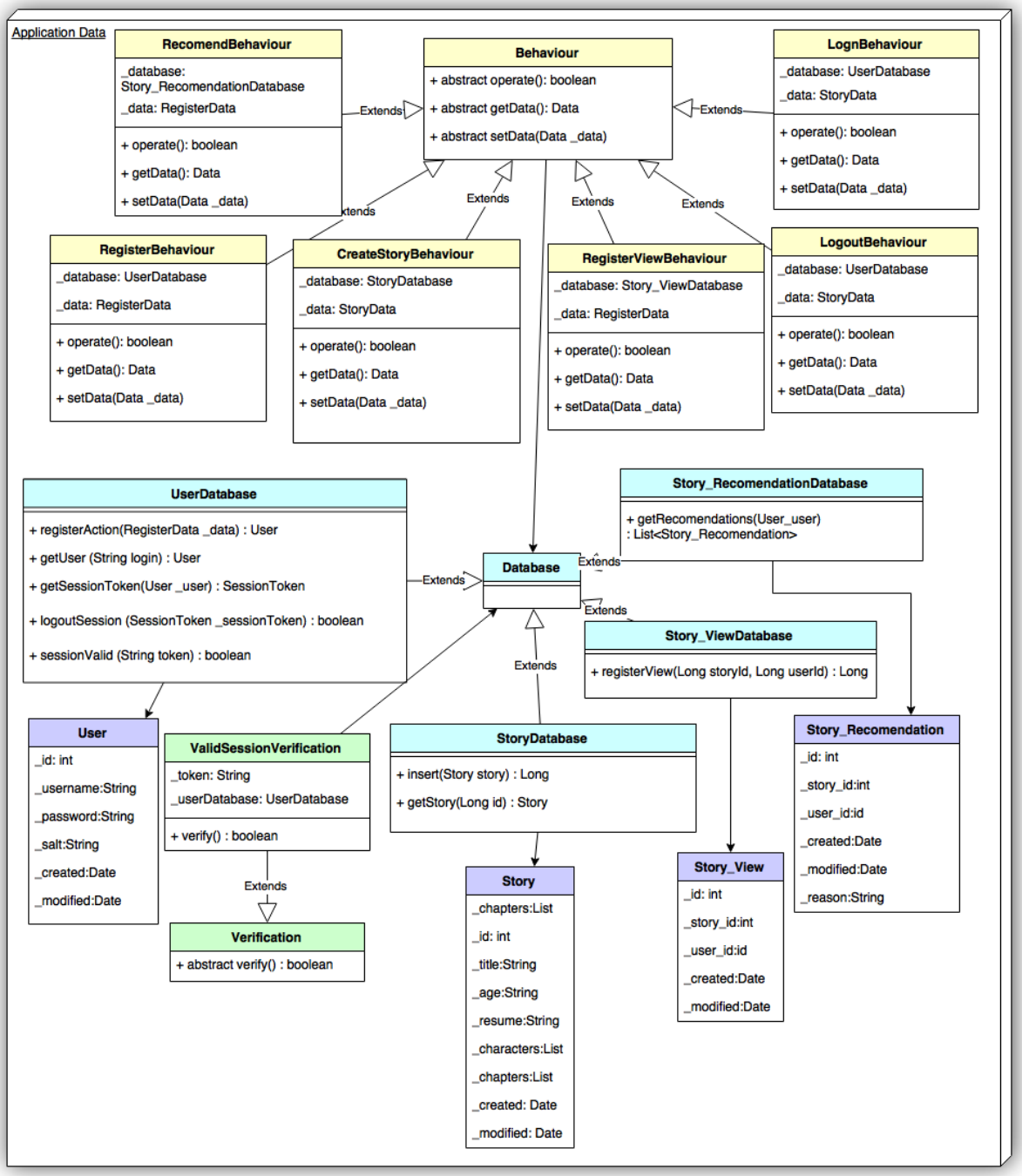

Figura 23 - Diagrama de classes da aplicação servidor (Dados)

Analisando o diagrama anterior, podemos ver que existem três superclasses e várias subclasses. Este facto deve-se a que existem três comportamentos chave distintos:

- Comunicação entre a aplicação cliente e a aplicação servidor;
- Comportamento a executar;
- Base de dados.

Para além destas superclasses, existem ainda classes de dados, como *User*, *Story, Story_View* e *Story_Recomendation*.

A aplicação deste padrão de herança é benéfico, por ser possível, tal como podemos ver no diagrama, agrupar comportamentos semelhantes e adicionar novos sempre que necessário, sem que a estrutura da aplicação tenha que ser alterada a fundo.

A arquitetura da aplicação cliente é simples, comparada com a arquitetura do servidor. A sua responsabilidade é fazer pedidos e mostrar as respostas. Assim sendo, a arquitetura proposta (Figura 24) passa por ter dois tipos diferentes de classes: as classes responsáveis pelo processamento dos dados (*Controllers*) e as classes responsáveis pelos pedidos ao servidor (*Core*).

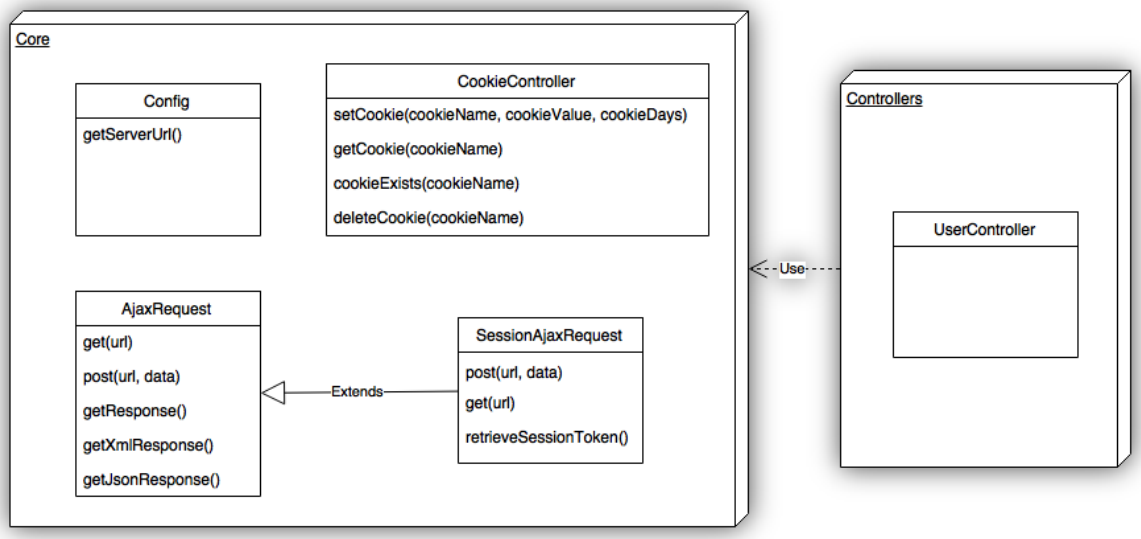

Figura 24- Aplicação cliente

5 Solução

Depois de desenhada a solução, passamos à fase de implementação desta. Dos componentes apresentados no Capítulo 4 serão explicados em detalhe:

- Aplicação para análise automática e todos os componentes relacionados com esta (base de dados);
- Aplicação para sincronização de base de dados e os componentes relacionados (base de dados).

Os restantes componentes desta plataforma estão a ser desenvolvidos pelo meu colega Filipe Couto, uma vez que esta plataforma foi criada e desenhada em parceria com ele.

5.1 Aplicação para Análise Automática

Como já foi vindo a ser referido ao longo desta dissertação, uma componente importante da plataforma para escrita criativa é a aplicação responsável pelo tratamento das histórias, através de técnicas de *text mining*, e da sugestão de conteúdos aos utilizadores, através de técnicas de *data mining*.

Para facilitar a utilização da *framework*, foi então criada uma interface (Figura 25), em *JavaFX* (2.4.3.3).

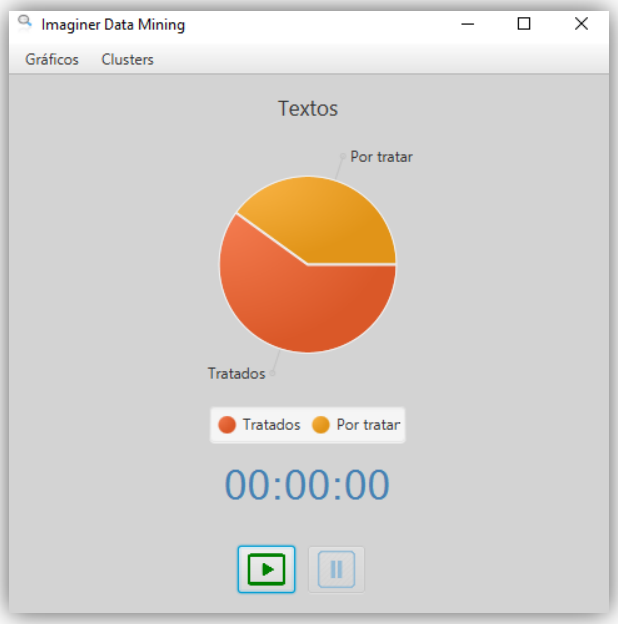

Figura 25 – Janela inicial

Quando a aplicação é iniciada são criadas duas *Threads* (Figura 26): uma para tratar da deteção do idioma, género literário, personagens, conteúdo impróprio e resumo (Anexo 4) e outra para recomendar histórias consoante o título, idioma e personagens (Anexo 5).

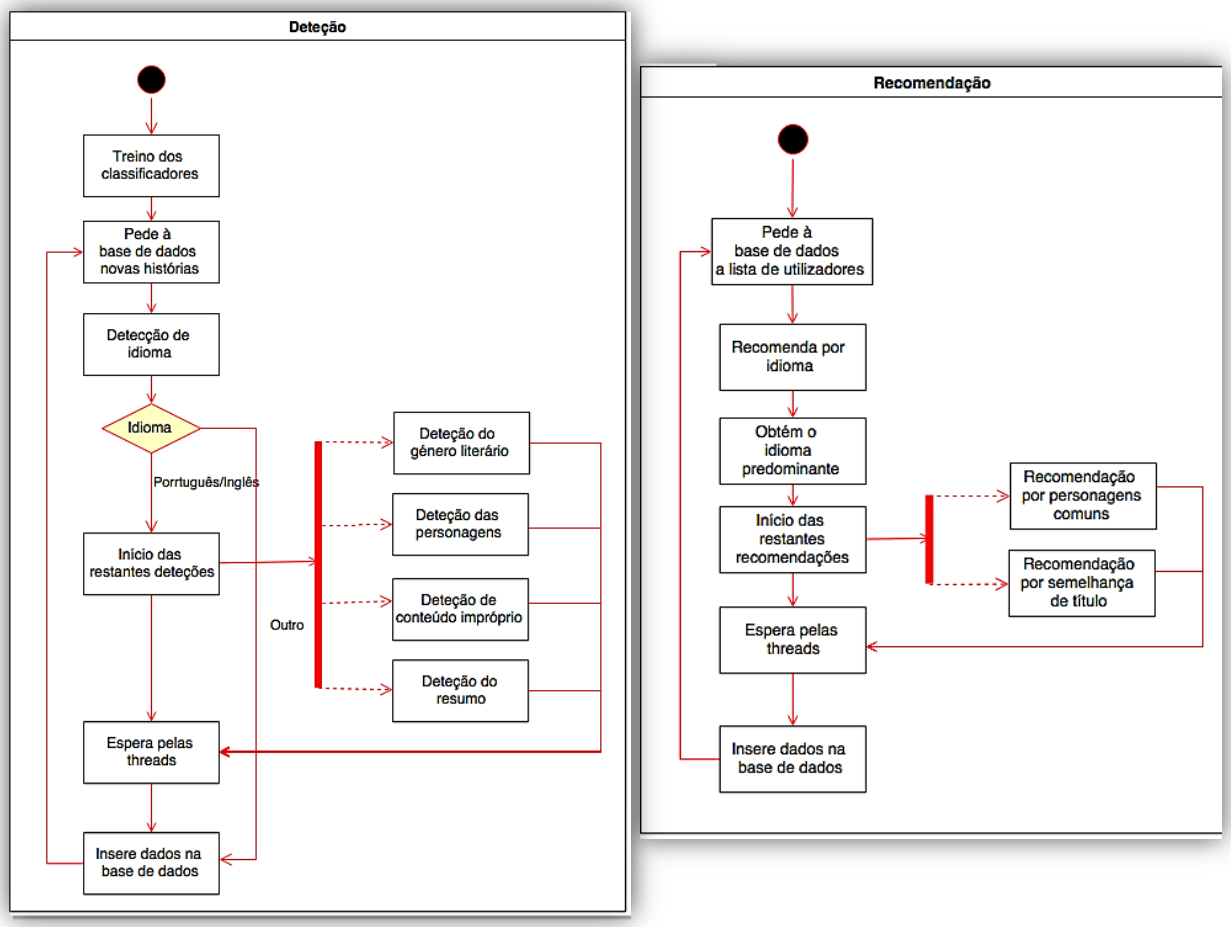

Figura 26 – Diagrama de atividades para a deteção e recomendação

Estas *Threads* por sua vez instanciam novas *Threads* que vão tratar de cada uma das deteções e recomendações (em detalhe mais à frente nas Secções 5.1.1 a 5.1.7).

Para além das funcionalidades mencionadas anteriormente, para facilitar a perceção dos dados guardados na base de dados são disponibilizadas estatísticas como a distribuição de géneros literários, conteúdo impróprio e idioma, no universo das histórias criadas pelos utilizadores (Figura 27).

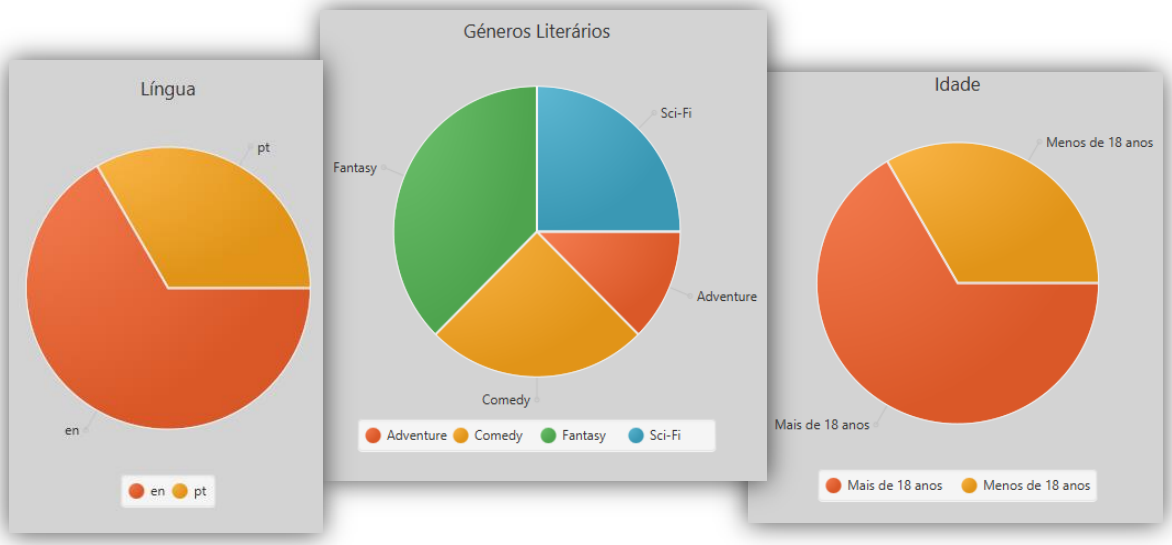

Figura 27 - Estatísticas disponíveis

5.1.1 Deteção de Géneros Literários e Conteúdo Impróprio

Para a implementação da deteção de géneros literários e conteúdo impróprio foi decidido, na Secção 3.1.2, que iria ser utilizada a *framework Apache Spark* (Código 1).

```
JavaSparkContext sc = new
        JavaSparkContext(newSparkConf().setAppName("Mining App")
        .setMaster("local[*]"));
```

Código 1 – Instanciação da framework *Apache Spark*

5.1.1.1 Algoritmos Implementados

Como já foi referido na Secção 3.1.4 foram escolhidos vários algoritmos a implementar, todos pertencentes à *framework*.

TF-IDF

O TF-IDF foi o primeiro algoritmo a ser implementado. Tal como foi explicado na Secção 2.4.1.1 é um algoritmo de extração de características que calcula a importância de um determinado termo no documento, sendo composto por dois algoritmos: TF (*HashingTF* no *Apache Spark*) que calcula a importância de cada palavra num documento e IDF que calcula o seu potencial discriminatório no universo de todos os documentos.

Este algoritmo (TF-IDF) recebe inicialmente dois *ArrayList* do tipo *string* com os valores verdadeiros e falsos, para o posterior treino de um classificador, ou apenas uma *string* no caso de a extração de características para classificação. Seguidamente é feito um tratamento dos dados, para transformar estes num *ArrayList* do tipo *Row* que, no caso do TF-IDF, contém a

label (se é um valor verdadeiro ou falso) e uma *string*. Como o TF-IDF aceita *labels*, os *arrays* resultantes deste tratamento são concatenados num *array* só para facilitar a extração.

Depois deste tratamento, o *ArrayList* é convertido em *JavaRDD* (lista própria da *framework*). Este *array JavaRDD* do tipo *Row* é então utilizado para criar duas *DataFrames*: uma juntamente com uma estrutura (*StructType*) que indica à *DataFrame* como deve organizar os dados (a estrutura *label* representa a *label* do *Row* e a *sentence* o texto) e uma segunda, em que é utilizado um *Tokenizer* para separar o texto da primeira *DataFrame* em palavras.

É esta segunda *DataFrame* que o algoritmo *HashingTF* utiliza para transformar o texto em dados. A *DataFrame* resultante do *HashingTF* é então utilizada para o algoritmo IDF. No final os dados são armazenados numa variável (*ExtractedData*) criada especialmente para o efeito.

Assim sendo, a implementação deste algoritmo funcionará da seguinte forma (Figura 28):

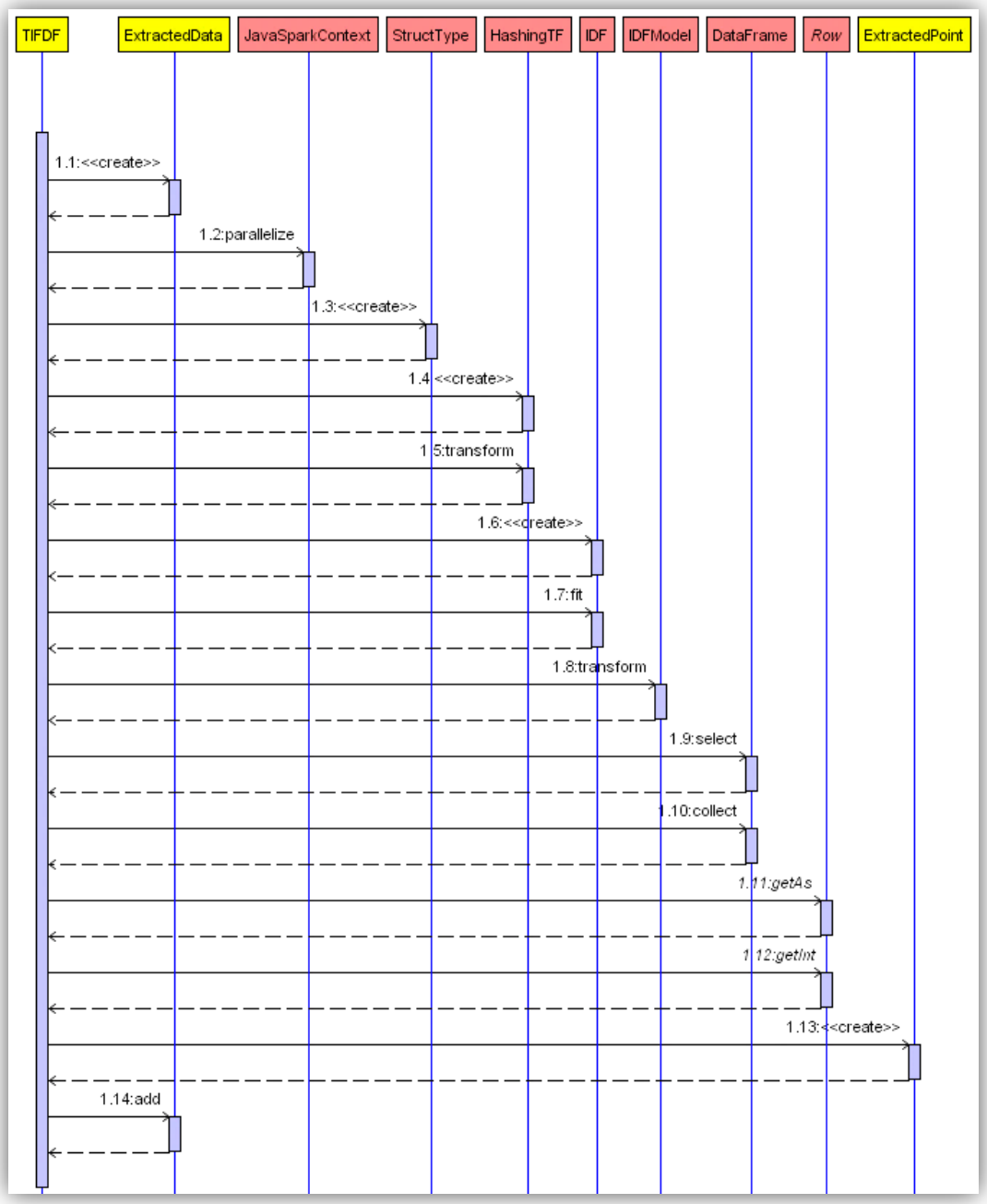

Figura 28 – Diagrama de sequência representativo do funcionamento do TF-IDF

Word2Vec

O segundo algoritmo de extração implementado foi o *Word2Vec*. Tal como foi explicado na Secção 2.4.1.2, este algoritmo pode ser implementado utilizando um de dois algoritmos: *Skip-gram* ou *Continuous Bag of Words* (CBOW). Na *framework* utilizada o algoritmo é implementado através do algoritmo *Skip-gram*.

Todo o pré-processamento dos dados é idêntico ao TF-IDF (conversão do *ArrayList* de texto ou *string* num *ArrayList* do tipo *Row*), sendo a única diferença que o *Word2Vec* não aceita as *labels* no *ArrayList* do tipo *Row*, como no caso do TF-IDF, e por isso a extração de características tem que ser em duas fases distintas (no que toca a valores para aprendizagem) (Figura 29).

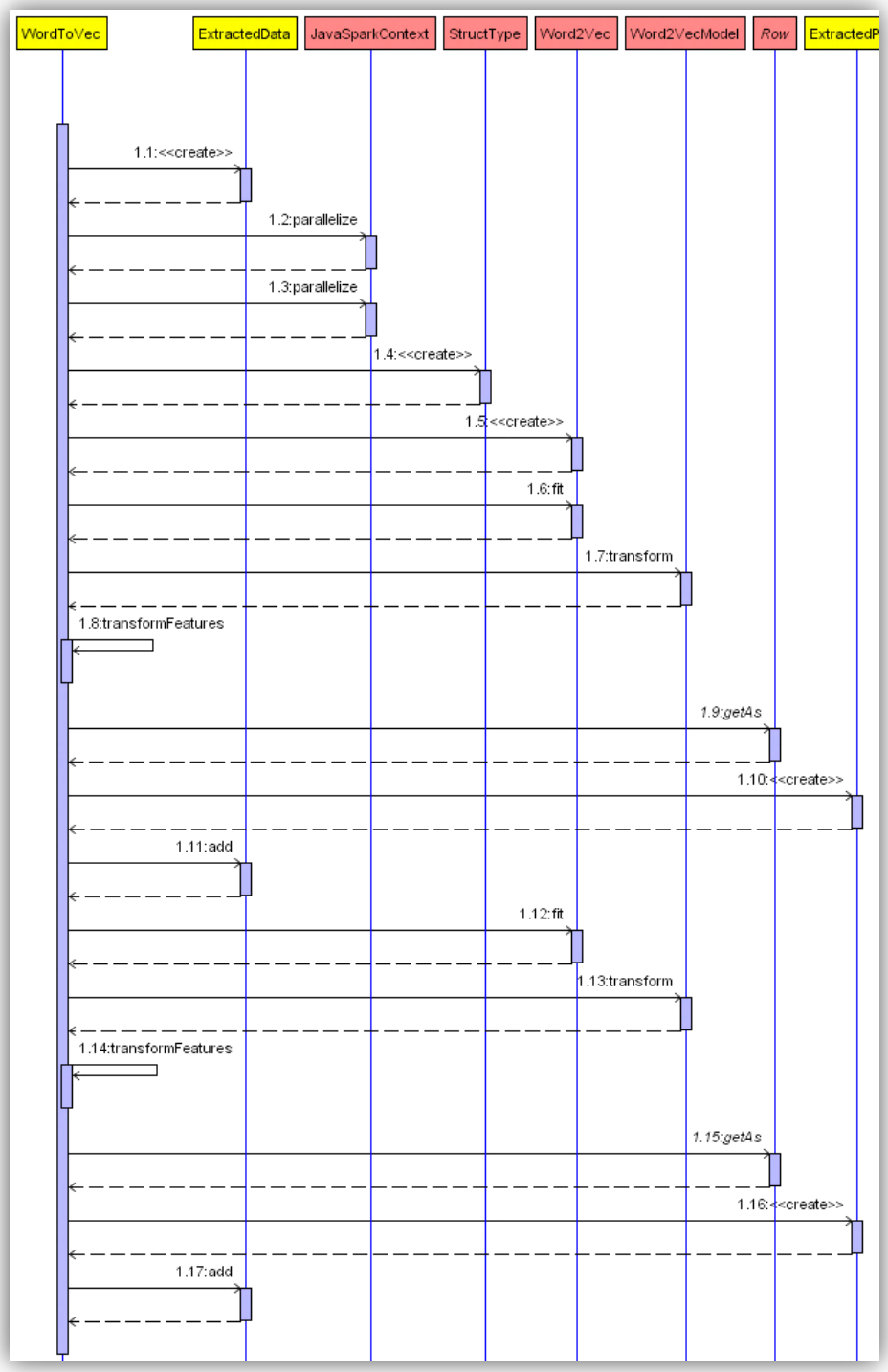

Figura 29 - Diagrama de sequência representativo do funcionamento do Word2Vec

K-Means

Para além dos algoritmos de extração, foram também implementados algoritmos de *clustering* para agrupar os dados obtidos da extração de forma a tentar aumentar a *performance* dos classificadores, através da utilização dos *clusters* gerados a partir do agrupamento das características para treinar os classificadores e classificar.

Um dos algoritmos de *clustering* implementado foi o *K-means*, que tal como explicado na Secção 2.4.1.6, funciona através da fixação de um número de *clusters* e nestes definir os seus centros, um para cada *cluster*.

Utilizando a *framework* para implementar este algoritmo é necessário determinar o número de *clusters* a utilizar e o número de iterações a fazer sobre os dados.

Depois de treinar o *K-means*, os valores agrupados podem ser acedidos através do método *clusterCenters()* que retorna todos os centros obtidos(Figura 30).

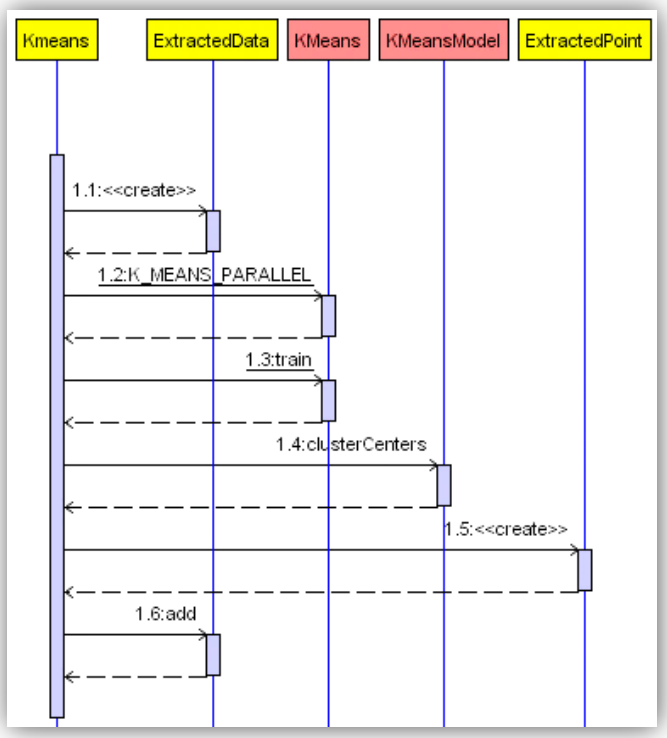

Figura 30 - Diagrama de sequência representativo do funcionamento do K-means

Gaussian Mixture Model

À semelhança *do K-means*, o algoritmo *Gaussian Mixture* também é um algoritmo de *clustering*, com algumas diferenças, como podemos ver na Secção 2.4.1.7. Este algoritmo agrupa os dados utilizando sub-distribuições Gaussianas.

Tal como é feito no *K-means*, aqui também é determinado o número de *clusters* a utilizar. Este algoritmo calcula sub-distribuições Gaussianas, contendo cada uma um peso, uma matriz *sigma* (matriz de variância de cada um dos componentes) e um vetor *mu* (vetor das médias dimensionais), onde se encontram os dados agrupados (Figura 31).

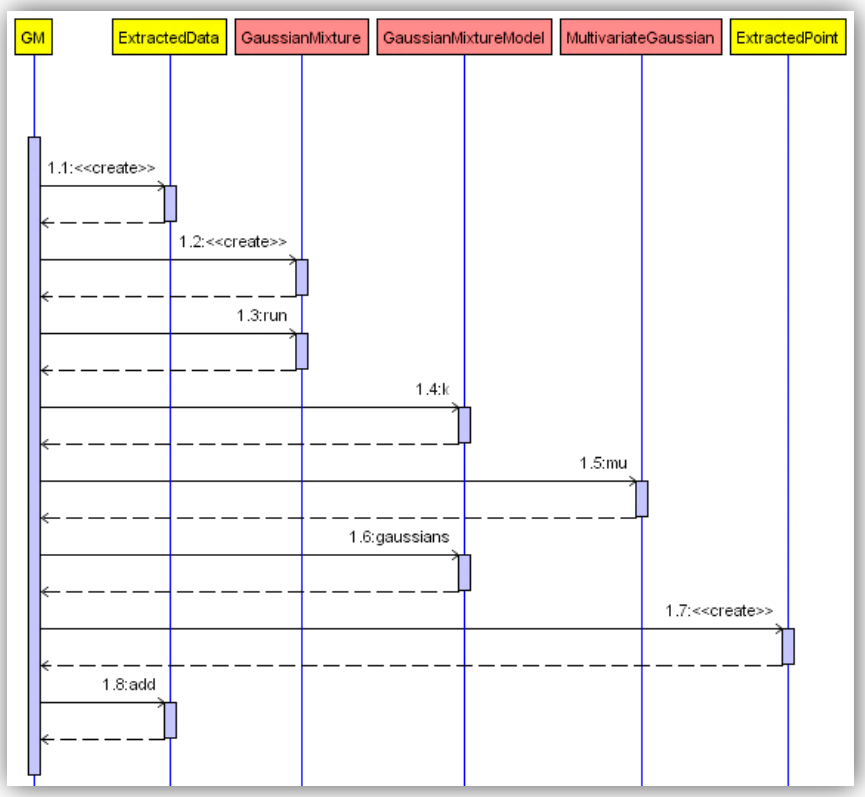

Figura 31 - Diagrama de sequência representativo do funcionamento do *Gaussian Mixture Model*

Latent Dirichlet Allocation

O algoritmo *Latent Dirichlet Allocation* (LDA) é também um algoritmo de *clustering* mas, ao contrário dos anteriores, este algoritmo representa os documentos como misturas de tópicos e é específico para tratar documentos de texto (Figura 32).

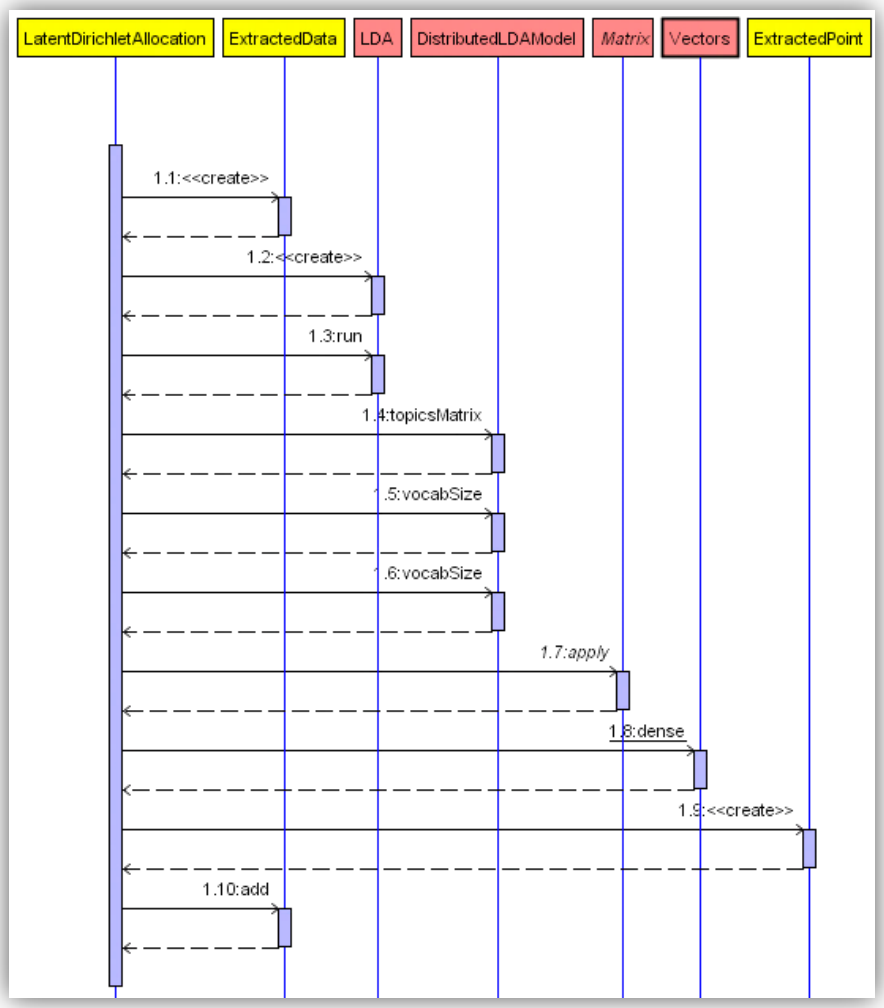

Figura 32 - Diagrama de sequência representativo do funcionamento do *Latent Dirichlet Allocation*

Decision Tree

O algoritmo Decision Tree, à semelhança de qualquer outro classificador, tem duas fases distintas: treino e classificação.

No treino (Código 2) é utilizado um *Array* do tipo *LabeledPoint* (classe do *Apache Spark*) e que é constituído por uma *label* e por um vetor de características. É com a *label* que o classificador compreende a que classe é que os valores que está a aprender pertence, sendo a classe a classificação do dados que o classificador aprende.

Na classificação (Código 3) é dado ao algoritmo um *array* do tipo *Vector* com as características obtidas da extração num texto do utilizador e é obtida a classificação do vetor, através dum valor *Double* e que corresponde a uma das *labels* aprendidas.

```
Integer numClasses = 2;
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<Integer,
      Integer>();
String impurity = "gini";
Integer maxDepth = 5;
Integer maxBins = 32;
model = DecisionTree.trainClassifier(dataToTrain, numClasses,
      categoricalFeaturesInfo, impurity, maxDepth, maxBins);
```

Código 2 – Implementação da aprendizagem do algoritmo *Decision Tree*

```
Vector v = PreProcessor.SVD(dataToClassify.getArrayVector(),sc);
double x = model.predict(v);
```

Código 3 - Implementação da classificação do algoritmo Decision Tree

Naive Bayes

Tal como o *Decision Tree*, o algoritmo *Naive Bayes* é também um algoritmo de classificação. Ao contrário do anterior, o *Naive Bayes* prevê a probabilidade da distribuição dos dados nas categorias, em vez de apontar qual a categoria em que o dado se encaixa. A implementação deste é similar ao *Decision Tree* (Código 4 *e* Código 5).

```
model = NaiveBayes.train(dataToTrain.rdd(), 1.0);
```

Código 4 - Implementação da aprendizagem do Naive Bayes

```
Vector v = PreProcessor.SVD(dataToClassify.getArrayVector(),sc);
double d = model.predict(v);
```

Código 5 - Implementação da classificação do algoritmo Naive Bayes

SVM

O último algoritmo de classificação a ser implementado foi o SVM ou *Support Vector Machine.*

Este algoritmo tem uma implementação muito similar aos dois algoritmos anteriores: no treino recebe um *array* do tipo *LabeledPoint* e na classificação um *array* do tipo *Vector*. No final da classificação retorna um *Double* com a classificação do texto (Código 6 e Código 7).

```
int numIterations = 100;
model = SVMWithSGD.train(dataToTrain.rdd(), numIterations);
model.clearThreshold();
```

Código 6 - Implementação da aprendizagem do SVM

```
Vector v = PreProcessor.SVD(dataToClassify.getArrayVector(),sc);
double d = model.predict(v);
```

Código 7 - Implementação da classificação do algoritmo SVM

5.1.1.2 Construção do Dataset

Como foi referido na Secção 4.2.1, é necessário encontrar um elevado número de histórias (na escala das centenas), devidamente classificadas consoante o seu género literário e conteúdo impróprio, para construir o dataset que irá ser usado para classificar histórias.

Numa fase inicial, foi criado um dataset com cerca de 50 histórias, em inglês. Estas histórias são retiradas do website *Shortbread Stories*[12](*website* de partilha de *short stories*).

No entanto estas histórias, por serem criadas e classificadas por escritores amadores, não davam grande segurança quanto à veracidade das classificações.

Numa segunda fase de construção do *dataset*, contactamos várias instituições (editoras, associações e outras instituições) para propor algum tipo de parceria, no sentido de conseguir textos para o dataset. Foram contactadas as seguintes instituições: Bertrand, Civilização, Presença, Leya, Porto Editora, Apel (Associação Portuguesa de Editores e Livreiros), Booktailors, CLUL (Centro de Linguística da Universidade de Lisboa) e CLUP (Centro de Linguística da Universidade do Porto).

De todas estas instituições apenas a CLUP, CLUL, Porto Editora, Booktailors e Presença responderam, sendo que destes apenas a Porto Editora se mostrou interessada em marcar uma reunião exploratória para discutir a temática.

Desta reunião e depois de explicados os parâmetros da temática e do iria ser necessário, não mostraram interesse em ajudar. Foi então abandonada a ideia de pedir ajuda a instituições.

Na fase final da construção do dataset, foi então decidido procurar *online* por excertos de livros (que facilmente se consegue obter uma classificação mais correta).

Através do *website Goodreads*[13], que tem uma classificação dos livros, foi possível encontrar 20 livros, para cada género literário, cujos excertos se encontravam disponíveis *online*, e adicioná-los ao dataset de textos em inglês.

Para o dataset em português, foi utilizado o *website LeLivros*[14], sendo escolhidos deste *website* 20 livros, para cada género literário, de onde foram retirados excertos.

Devido à complexidade de tratamento necessário a dar aos textos para que estes estivessem de acordo com as especificidades necessárias a ser viáveis de utilizar (livros em formato pdf, mal formatados, etc.) na última versão do *dataset*, foi apenas possível criar um dataset com um total de 455 textos em inglês e 120 textos em português.

[12] http://www.shortbreadstories.co.uk/
[13] https://www.goodreads.com/
[14] http://lelivros.online/

5.1.1.3 Abordagem Final

Tendo sido implementados todos os algoritmos anteriores na aplicação, apenas dois foram escolhidos para serem na aplicação final: *TF-IDF* e *Naive Bayes* (Secção 6.2.1).

Esta abordagem resolve o desafio que é detetar o género literário, mas não dá uma resposta viável no que toca a determinar se uma história tem conteúdo impróprio ou não, como irá ser explicado em detalhe na Secção 6.2.2.

Para aumentar a *performance* destes algoritmos, no que toca a encontrar o(s) género(s) literário(s) correto(s), foram aplicadas técnicas de pré-processamento como a transformação dos textos em *Ngrams*[15], remoção de palavras comuns ("*alone*", "*along*", "*already*", entre outras) e palavras temporais ("*Monday*", "*Tuesday*", "*January*", "*Tomorrow*", entre outras) que não trazem novo conhecimento aos algoritmos. Também foi utilizado o *Stemmer* de *Porter* para reverter palavras à sua raiz semântica (transforma palavras como "*revival*" em "*reviv*") (Ibrahim Naji 2013).

Estas técnicas são aplicadas utilizando o código original de *Martin Porter* (Martin Porter 2006) para implementar o *stemming* e a *framework* da *Apache*, *Apache Lucene*, para transformar histórias inteiras em *bigrams* (*ngrams* de duas palavras). Para a remoção de palavras comuns e com significado temporal foram criados métodos próprios, em português e inglês. As palavras utilizadas nestes métodos encontram-se armazenadas em ficheiros de texto e que o programa carrega sempre que necessário (Figura 33).

[15] Sequência contígua de n palavras de um determinado texto.

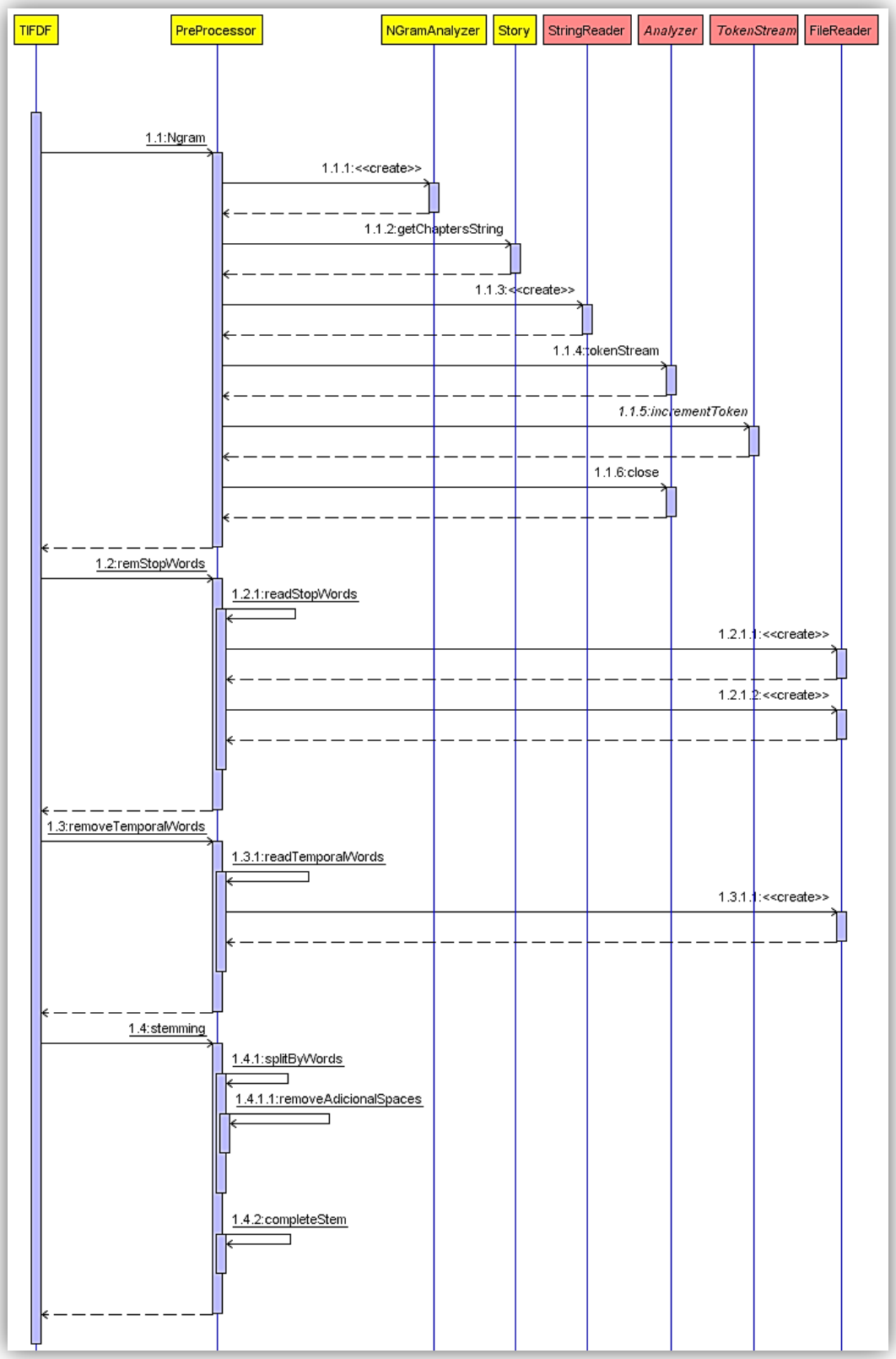

Figura 33 - Diagrama de sequência representativo do funcionamento do pré-processamento

Para além das técnicas de pré-processamento, também foram aplicadas outras, como a remoção de nomes de pessoas, cidades, países, entre outras e remoção de adjetivos (Ibrahim Naji 2013). No entanto e por não aumentarem a performance dos algoritmos, não foram utilizadas na versão final da aplicação.

Como já foi referido na Secção 2.4.1, os algoritmos de classificação funcionam em duas fases: uma primeira fase em que os algoritmos aprendem características do *dataset* e uma segunda fase em que usam o que aprenderam para prever se um texto é de um determinado género literário ou não.

Para ambas as fases são utilizados sete classificadores diferentes para cada idioma, ou seja no total catorze classificadores, para que, cada um destes possa determinar se a história tem esse género presente.

Tal como será explicado em detalhe na Secção 6.2.1, isto ajuda a mitigar o problema que é a generalidade de alguns géneros literários, que se encontram muitas vezes combinados com outros géneros literários (por exemplo comédia romântica).

Esta abordagem vai também um pouco ao encontro com o que alguns *websites* (como por exemplo o *website Shortbread Stories*, que foi utilizado para obter parte do *dataset*) fazem quando classificam histórias: atribuem-lhe vários géneros em vez de um só.

A fase de treino só tem que ser realizada uma vez, mas a fase de teste terá que ser repetida todas as vezes que aparecer uma nova história na base de dados. Por isso estas fases foram separadas: a fase de treino é executada apenas uma vez, quando a aplicação é iniciada (Código 8) e são treinados sete classificadores (um por cada género literário) que são guardados num *array* de classificadores.

```
ENclassifiers = new ArrayList<>();
trainClassifierEN(trueENAdventure, falseENAdventure);
trainClassifierEN(trueENComedy, falseENComedy);
trainClassifierEN(trueENDrama, falseENDrama);
trainClassifierEN(trueENFantasy, falseENFantasy);
trainClassifierEN(trueENRomance, falseENRomance);
trainClassifierEN(trueENSciFi, falseENSciFi);
trainClassifierEN(trueENTerror, falseENTerror);
…
Extractor e = new TIFDF(True, False);
ExtractedData extracted = e.extractData(sc, sqlContext);
JavaRDD<LabeledPoint> dataToTrain = extracted.getJavaRDDLabeledPoint(sc);
Classifier c = new Naivebayes(dataToTrain);
c.trainData(sc);
ENclassifiers.add(c);
```

Código 8 - Treino dos classificadores para textos em inglês

Só depois de todos os classificadores treinados, é que as histórias começam a ser tratadas (Código 9), sendo que para facilitar a classificação a cada género é atribuído um número, que depois é utilizado para identificar o seu resultado.

```
classifieStory(extracted, 0, ENclassifiers);
classifieStory(extracted, 1, ENclassifiers);
classifieStory(extracted, 2, ENclassifiers);
classifieStory(extracted, 3, ENclassifiers);
classifieStory(extracted, 4, ENclassifiers);
classifieStory(extracted, 5, ENclassifiers);
classifieStory(extracted, 6, ENclassifiers);
…
results.add(classifiers.get(i).classifyData(extracted, sc));
```

Código 9 - Classificação das histórias em inglês

Como já foi dito anteriormente, esta abordagem só dá uma resposta razoável (mais de 50% de precisão) à deteção dos géneros literários de um texto, mas não à determinação de se um texto tem conteúdo impróprio ou não.

Por esta razão, foi necessário encontrar uma nova forma de detetar se uma história tem ou não conteúdo impróprio para alguns leitores, como iremos ver no próximo capítulo.

5.1.2 Deteção de Conteúdo Impróprio

Depois de terem sido feitos testes (Secção 6.2) e de se concluir que a utilização de algoritmos de *text mining* é inviável neste caso, foi necessário encontrar outra abordagem.

Uma abordagem possível e que é utilizada pela *Google* (AXON 2010), é a utilização de uma lista de palavras, ditas como ofensivas (palavras com conteúdo sexual, etc.), para comparar com as palavras dos textos, atribuindo um rácio de quantas palavras por texto (0.1 %) pode um texto ter até ser considerado impróprio (Código 10).

```
int countOffensive = 0;
int countWords =
        PreProcessor.removeStopWords(countWords(), language).size();
ArrayList<String> offensiveWords = offensiveWords(language);
ArrayList<String> ngrams = Ngram(story);
for (String string : ngrams) {
        if (offensiveWords.contains(string.toUpperCase())) {
              countOffensive++;
        }
}
double perc = ((double)countOffensive/(double)countWords)*100;
if (perc>=0.1) {
        result.setOver18(true);
} else {
        result.setOver18(false);
}
```

Código 10 - Palavras ofensivas

Este valor (0,1%) foi encontrado através do teste de várias possibilidades (Secção 6.2.3), para encontrar aquela que respondesse corretamente ao maior número de histórias do dataset criado para a tentativa de solução anterior.

5.1.3 Deteção das Personagens e Construção do Resumo

Para a deteção das personagens de uma história e obtenção do seu resumo, optou-se pela utilização de técnicas de NLP (Secção 2.1.1), mais concretamente as *frameworks OpenNLP* (Secção 2.4.3.2), para textos em português e também para os resumos em inglês, e *Stanford CoreNLP* (Secção 2.4.3.1), para as personagens em inglês.

Existe uma distinção para a forma de encontrar as personagens no texto, uma vez que a *framework Stanford CoreNLP* fornece um reconhecedor de nomes (NER ou *Named Entity Recognizer*), que analisa o texto para encontrar nomes de pessoas, cidades, organizações, entre outras (Código 11).

```
ArrayList<String> maybeNames =
        getDiferentChars(PreProcessor
        .NERNameFinder(story.getChaptersString()));
```

Código 11 - Obtenção dos nomes de personagens para textos em inglês

Este reconhecedor carrega o ficheiro "english.all.3class.distsim.crf.ser.gz" (ficheiro disponibilizado pelo departamento de NLP de *Stanford*) e a partir deste, cria um classificador de palavras, que classifica as palavras como "PERSON", "ORGANIZATION", "LOCATION" e "O", sendo que "O" são todas as palavras que não se encaixam nas outras categorias (Código 12).

```
ArrayList<String> entities = new ArrayList<>();
String serializedClassifier =
        "classifiers/english.all.3class.distsim.crf.ser.gz";
AbstractSequenceClassifier classifier =
        CRFClassifier.getClassifierNoExceptions(
        serializedClassifier);
String fileContents = story.replaceAll("[^a-zA-Z ]", "");
String classifyContent = classifier.classifyToString(fileContents);
String[] tmp = classifyContent.split(" ");
for (int i = 0; i < tmp.length; i++) {
        String[] tmp1 = tmp[i].split("/");
        if (!tmp1[0].isEmpty()) {
               if (tmp1[1].equals("PERSON")) {
                    entities.add(tmp1[0]);
                }
        }
}
```

Código 12 - Utilização do NER

Depois de obtidos todos os nomes, é necessário retirar os repetidos (Código 13).

```
ArrayList<String> uniqueChars = new ArrayList<>();
for (String name : chars) {
         if (!uniqueChars.contains(name)) {
                    uniqueChars.add(name);
         }
}
```

Código 13 - Escolha dos diferentes personagens

O NER reconhece entidades em vários idiomas, mas não em português. Por isso esta abordagem não é aplicável aos textos em português.

Para encontrar os nomes em português, foi então usado a *framework OpenNLP*, que classifica as palavras morfologicamente (*Part-of-Speech Tagger*).

Este classificador também carrega um ficheiro ("pt-pos-maxent.bin"), para puder classificar cada palavra. Neste caso, só serão aproveitadas as palavras classificadas como "NNP" (nome próprio singular), uma vez que os nomes de personagens aparecem sempre no singular (Código 14). Depois de obtidos os nomes, estes são comparados a uma lista de nomes criada para o efeito, uma vez que entre estes nomes encontrados, podem vir nomes de cidades, organizações, entre outras.

```
ArrayList<String> reallyNames = PreProcessor.loadNameList(language);
InputStream modelIn = new FileInputStream("postagger/pt-pos-maxent.bin");
POSModel model = new POSModel(modelIn);
POSTaggerME tagger = new POSTaggerME(model);
String[] wordsArray =
       removeAdicionalSpaces(story.getChaptersString().split(" "));
String tags[] = tagger.tag(wordsArray);
for (int i = 0; i < tags.length; i++) {
       if (tags[i].equals("NNP")){
              String word = wordsArray[i].toUpperCase()
                     .replaceAll("[^a-zA-Z ]", "");
              if (!names.contains(word) && isAName(reallyNames, word)) {
                     names.add(word);
              }
       }
}
```

Código 14- Obtenção dos nomes de personagens para textos em português

A obtenção do resumo é bastante semelhante à obtenção dos nomes em português (Código 15). Desta vez são obtidos os nomes singulares e plurais ("NN" e "NNS"), assim como os verbos ("VB") (Código 16).

```
words = PreProcessor.remStopWords(PreProcessor
       .findNounsAndVerbs(data, language), language);
resume = resumeSetences(words);
```

Código 15 - Obtenção do resumo

```
POSModel model = new POSModel(modelIn);
POSTaggerME tagger = new POSTaggerME(model);
String[] wordsArray = removeAdicionalSpaces
       (story.getChaptersString().split(" "));
String tags[] = tagger.tag(wordsArray);
for (int i = 0; i < tags.length; i++) {
       if (tags[i].equals("NN") ||tags[i].equals("NNS") ||
            tags[i].equals("VB")) {
            if (!words.contains(wordsArray[i])) {
                  words.add(wordsArray[i]);
            }
      }
}
```

Código 16 - Obtenção de nomes e verbos

Depois de encontradas, as palavras são ordenadas de acordo com a frequência de ocorrências no texto. As dez palavras mais comuns são utilizadas para criar o resumo, ou seja todas as frases com estas palavras são utilizadas para o resumo (Código 17).

```
String[] wordsArray = PreProcessor
        .removeAdicionalSpaces(data.getChaptersString().split(" "));

String[] ordenateWords = ordenateWords(words, wordsArray);
String[] sentences = data.getChaptersString().split("\\.");
ArrayList<String> chosenSentences= new ArrayList<>();
String [] chosen = new String[sentences.length];
for (int i = 0; i < 10; i++) {
        for (int j = 0; j < sentences.length; j++) {
              if (sentences[j].toUpperCase()
                          .contains(ordenateWords[i].toUpperCase())) {
                    if(!chosenSentences.contains(sentences[j])) {
                          chosenSentences.add(sentences[j] + ".");
                          chosen[j] = sentences[j] + ".";
                          break;
                    }
              }
        }
}
String resume = "";
for (int i =0; i<chosen.length;i++){
        if(chosen[i]!=null){
              resume+= chosen[i]+" ";
        }
}
```

Código 17 - Escolha das frases para o resumo

5.1.4 Deteção do Idioma

Para a deteção do idioma (Código 18) das histórias foi utilizada a biblioteca *Language Detection* da *Cybozu Labs*[16]. Esta biblioteca tem uma precisão de 99% e deteta 47 linguagens diferentes, entre elas o português e inglês (Shuyo Nakatani n.d.).

Para utilizar esta biblioteca basta instanciar os perfis dos idiomas e utilizar o detetor para encontrar o idioma do texto (Código 19).

```
language = Language.transformStringIntoLanguage(getLanguage(data));
```

Código 18 - Obtenção do idioma

```
DetectorFactory.loadProfile("Language profiles");
Detector detector = DetectorFactory.create();
String text = story.getChaptersString();
detector.append(text);
String lang = detector.detect();
```

Código 19 - Utilização do detetor de idiomas

Este novo idioma é então convertido para um idioma do sistema (Código 20), sendo que idiomas que não sejam português ou inglês são classificados como "Outro".

```
switch (language) {
	case "en":
		return ENGLISH;
	case "pt":
		return PORTUGUES;
	default:
		return OUTRO;
}
```

Código 20 - Transformação do idioma

5.1.5 Recomendação por Título

Para a recomendação de títulos foi utilizada a distância de *Levenshtein*, para encontrar as histórias com os títulos mais semelhantes aos que o leitor leu previamente (Figura 34).

[16] http://labs.cybozu.co.jp/en/

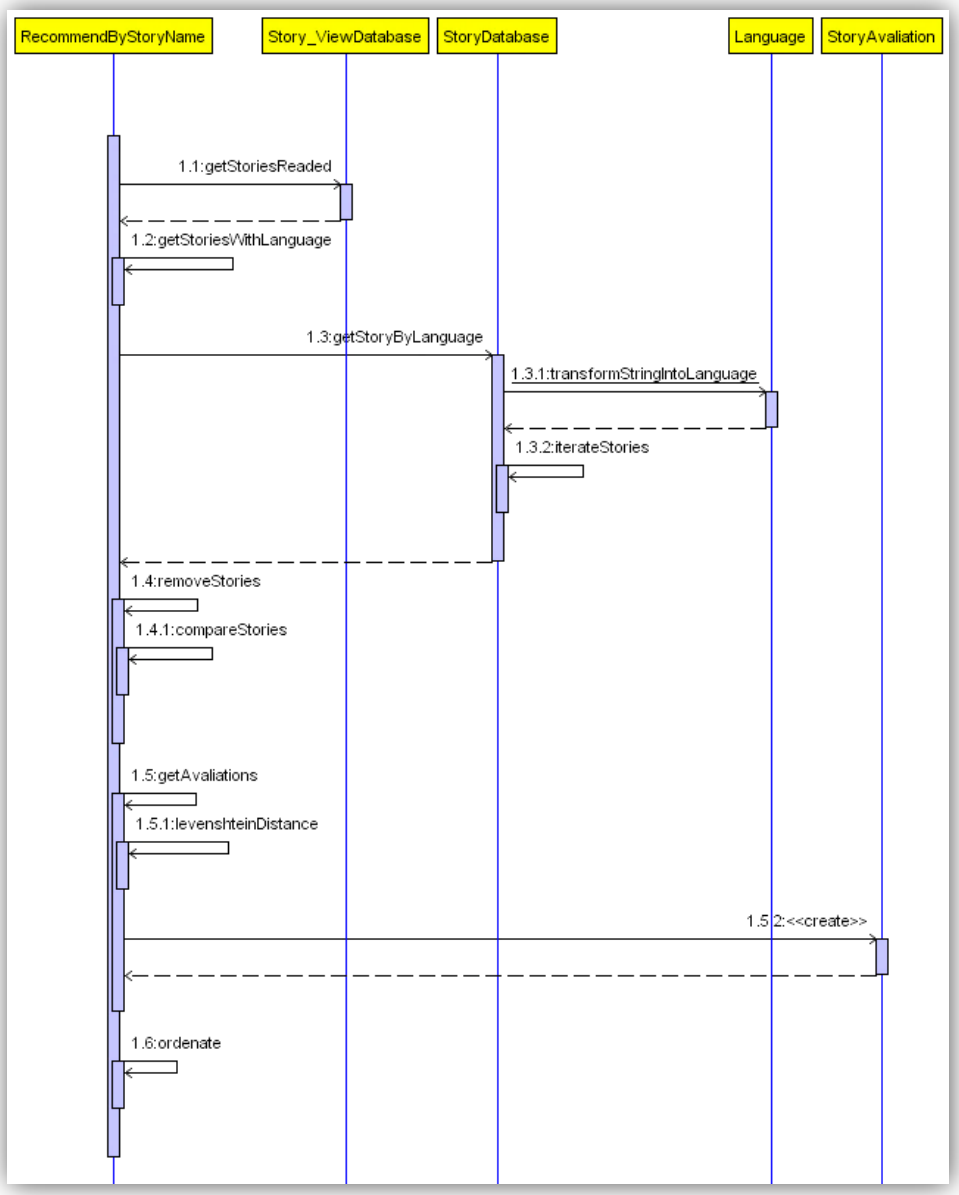

Figura 34 - Diagrama de sequência representativo do funcionamento da recomendação por título

Para calcular a distância de *Levenshtein* (RosettaCode 2016) (Código 21) de uma história a todas as histórias que o leitor leu, no seu idioma predominante, é calculada a média das distâncias da história a todas as histórias lidas (Código 22).

```
a = a.toLowerCase();
b = b.toLowerCase();
int [] costs = new int [b.length() + 1];
for (int j = 0; j < costs.length; j++)
      costs[j] = j;

for (int i = 1; i <= a.length(); i++) {
      costs[0] = i;
      int nw = i - 1;

      for (int j = 1; j <= b.length(); j++) {
            int cj = Math.min(1 + Math.min(costs[j], costs[j - 1]),
              a.charAt(i - 1) == b.charAt(j - 1) ? nw : nw + 1);
            nw = costs[j];
            costs[j] = cj;
 }
}
```

Código 21 - Cálculo da distância de *Levenshtein*

```
int sum=0,count=0;
for (Story read: storiesToTest){
        sum+=levenshteinDistance(read.getTitle(),story.getTitle());
        count++;
}
return new StoryAvaliation((double)(sum/count), story);
```

Código 22 - Cálculo da média das distâncias de *Levenshtein*

Finalmente, as histórias são ordenadas de acordo com a sua distância (as histórias com menor distância são as mais semelhantes) (Código 23).

```
ArrayList<Story> stories = new ArrayList<>();
for (int i = 0; i < storiesRecommended.size() - 1; i++) {
        for (int j = 0; j < storiesRecommended.size() - i - 1; j++) {
              if (storiesRecommended.get(j).getSimilarity() >
                          storiesRecommended.get(j + 1).getSimilarity())
                          {
                    StoryAvaliation temp =
                          storiesRecommended.get(j + 1);
                    storiesRecommended.set(j + 1,
                          storiesRecommended.get(j));
                    storiesRecommended.set(j, temp);
              }
        }
}
for (int k = 0; k < 10; k++) {
        stories.add(storiesRecommended.get(k).getStory());
}
```

Código 23 - Ordenação das histórias consoante a sua distância

5.1.6 Recomendação por Personagens

Na recomendação por personagens não foi utilizada nenhuma *framework* ou biblioteca. Esta recomendação (Código 24) é realizada através da procura de todas as personagens das histórias lidas (Código 25).

```
ArrayList<Story> storiesRead = SVDatabase.getStoriesReaded(user);
if (storiesRead.isEmpty()) return false;
ArrayList<String> charactersFromStories =
       getDiferentChars(storiesRead);
ArrayList<Story_Viewed> storiesRecommended =
SDatabase.getStoriesWithCharacters(charactersFromStories);
ArrayList<Story> mostViewed = ordenate(storiesRecommended);
```

Código 24 - Recomendação por personagens

```
ArrayList<String> uniqueChars = new ArrayList<>();
for (Story story : storiesRead) {
        ArrayList<String> chars = story.getChars();
        if (story.isFanFiction()) {
               for (String name : chars) {
                      if (!uniqueChars.contains(name)) {
                             uniqueChars.add(name);
                      }
               }
     }
}
```

Código 25 - Obtenção das diferentes personagens

Depois de encontradas as diferentes personagens de todas as histórias lidas pelo leitor, é então feito um pedido à base de dados as histórias que tenham como personagens alguma das personagens obtidas anteriormente, e é realizada uma contagem das visualizações de cada uma destas histórias (Código 26).

```
MongoDatabase db = this._database.retrieveConnection();
FindIterable<Document> iterable = db.getCollection("Story").find(
        new Document("$or", findChars(charactersFromStories)));
return iterateStories_Viewed(iterable, db);
            …
ArrayList<Story_Viewed> stories = new ArrayList<>();
iterable.forEach(new Block<Document>() {
    @Override
    public void apply(final Document document) {
        Object tmp[] = document.values().toArray();
        ArrayList<Genre> genres = getGenres(tmp[4].toString());
        ArrayList<String> chars = getCharacters(tmp[8].toString());
        ArrayList<Chapter> chapters =
                   getChapters(tmp[9].toString());
        Story story = new Story(tmp[0].toString(),
                   Integer.parseInt(tmp[1].toString()),
                   Integer.parseInt(tmp[2].toString()),
                   tmp[3].toString(), chapters);
        story.setGenres(genres);
        story.setOver18(Boolean.valueOf(tmp[5].toString()));
        story.setFanFiction(Boolean.valueOf(tmp[7].toString()));
        story.setResume(tmp[6].toString());
```

```
            story.setLanguage(Language.transformStringIntoLanguage(
                        tmp[10].toString()));

        story.setChars(chars);
        AggregateIterable<Document> iterable =
                  db.getCollection("Story_View").aggregate(asList(
                      new Document("$group", new Document("_id",
                     "$story_id").append("count",
                      new Document("$sum", 1)))));
                  iterable.forEach(new Block<Document>() {
                  @Override
                  public void apply(final Document document) {
                      System.out.println(document.toJson());
                      Object tmp[] = document.values().toArray();
                      int count = Integer.parseInt(tmp[1].toString());
                      stories.add(new Story_Viewed(story, count));
                  }
              });
        }
});
```

Código 26 - Pedido à base de dados por histórias com as personagens

Depois de encontradas, as histórias são ordenadas por ordem decrescente do número de visualizações e as dez primeiras são recomendadas (Código 27).

```
ArrayList<Story> stories = new ArrayList<>();
for (int i = 0; i < storiesRecommended.size() - 1; i++) {
    for (int j = 0; j < storiesRecommended.size() - i - 1; j++) {
        if (storiesRecommended.get(j).getCount() <
                    storiesRecommended.get(j + 1).getCount()) {
            Story_Viewed temp = storiesRecommended.get(j + 1);
            storiesRecommended.set(j + 1,
                    storiesRecommended.get(j));
            storiesRecommended.set(j, temp);
        }
    }
}
int size = 10;
if (storiesRecommended.size() < 10) {
    size = storiesRecommended.size();
}
for (int k = 0; k < size; k++) {
    stories.add(storiesRecommended.get(k).getStory());
}
```

Código 27 - Ordenação das histórias por visualizações

5.1.7 Recomendação por Idioma

A recomendação por idioma (Código 28) é bastante similar à anterior:

1. É encontrada a linguagem predominante nas histórias lidas pelo leitor (Código 29);
2. É pedido à base de dados todas as histórias nesse idioma juntamente com a contagem de visualizações;

3. Essas histórias são ordenadas por ordem decrescente do número de visualizações (Código 30);
4. As dez mais lidas são recomendadas (Código 27).

```
ArrayList<Story> storiesRead = SVDatabase.getStoriesReaded(user);
setLanguage(getMostReadLanguage(storiesRead));
ArrayList<Story_Viewed> storiesRecommended =
SDatabase.getStory_ViewedByLanguage(getLanguage());
ArrayList<Story> mostViewed = ordenate(storiesRecommended);
```

Código 28 - Recomendação por Idioma

```
int pt = 0, en = 0, other = 0;
for (Story story : storiesRead) {
        Language tmp = story.getLanguage();
        switch (tmp) {
               case ENGLISH:
                      en++;
                      break;
               case PORTUGUES:
                      pt++;
                      break;
               default:
                      other++;
                      break;
        }
}
if (en > pt && en > other) return Language.ENGLISH;
if (pt > en && pt > other) return Language.PORTUGUES;
return Language.OUTRO;
```

Código 29 - Obtenção do idioma mais lido

```
MongoDatabase db = this._database.retrieveConnection();
FindIterable<Document> iterable =
        db.getCollection("Story").find(new Document("language",
        Language.transformStringIntoLanguage(language)));
return iterateStories_Viewed(iterable, db);
```

Código 30 - Pedido à base de dados por histórias com o idioma

5.1.8 Deteção de Novos Géneros Literários e Idiomas

Como foi dito anteriormente, os algoritmos de *clustering* foram implementados e utilizados para tentar filtrar as características obtidas durante a extração (sendo utilizados neste caso, duma forma não convencional). Estes algoritmos acabaram por não ser utilizados para a classificação, porque, apesar de terem uma performance razoável, não foram os melhores.

O algoritmo K-Means, por ser o mais rápido (Secção 6.2), foi reaproveitado para ser usado na deteção de novos géneros literários (Código 31).

```
ArrayList<Story> stories = SDatabase.getStoryByGenre(Genre.NAN);
if(!stories.isEmpty()) {
     setStoriesProcessed(true);
     Extractor extrator = new TIFDF(stories);
     ExtractedData data = extrator.extractData(sc, sqlContext);
     Cluster cluster = new Kmeans(data.getJavaRDDVector(sc));
     ArrayList<ClusterData> centers = cluster.clusterDataToFile(sc, data);
     ArrayList<String> folderNames = countDifferent(centers);
     ArrayList<ClassifiedStory>  classifiedStories =
               getClassifiedStories(stories,centers);
     createFolders(folderNames);
     addFilesToFolders(classifiedStories);
```

Código 31 - Obtenção de novos géneros literários

A descoberta dos novos géneros é realizada nos textos que são classificados como "Nan", isto é, quando a deteção é feita (Secção 5.1.1), os textos são testados contra sete sub-datasets diferentes ("Aventura", "Comédia", "Drama", "Fantasia", "Romance", "Ficção Cientifica" e "Terror"). Se um texto não for identificado como sendo de nenhum destes géneros é classificado como "Nan".

Assim sendo, a implementação anterior (Secção 5.1.1.1) foi ligeiramente modificada (Código 32):

```
int numClusters = 10;
int numIterations = 20;
ArrayList<ClusterData> correspondentCenter = new ArrayList<>();
KMeansModel model = KMeans.train(extractedData.rdd(), numClusters,
        numIterations, 4, KMeans.K_MEANS_PARALLEL());
ArrayList<ExtractedPoint> points = data.getData();
for (ExtractedPoint point : points) {
        correspondentCenter.add(new
        ClusterData(point.getLabel(),model.predict(
        point.getFeatures())));
}
return correspondentCenter;
```

Código 32 - Nova implementação do K-Means

Nesta nova implementação, podemos ver que todo o *clustering* é realizado, isto é, são encontrados os *clusters* (dez neste caso) dos textos e seguidamente, estes mesmos textos são comparados esses mesmos *clusters*, para tentar perceber qual é o centro mais perto desse texto.

Depois de encontrados os centros e os textos correspondentes a estes, são criadas pastas locais para guardar os textos (Código 33).

```
for (String folder : folderNames) {
        File theDir = new File("Dados para processar/generos/" + folder);
        if (!theDir.exists()) {
try {
        theDir.mkdir();
} catch (SecurityException se) {
                    …
}
}
 …
ArrayList<Integer> different = new ArrayList<>();
for (int i = 0; i < centers.size(); i++) {
        if (!different.contains(centers.get(i))) {
               different.add(centers.get(i));
        }
}
ArrayList<String> folderNames = new ArrayList<>();
for (int i = 0; i < different.size(); i++) {
        folderNames.add(different.get(i) + "");
}
return folderNames;
```

Código 33 - Criação das pastas

Depois de criadas as pastas, os textos são guardados nestas de acordo a que *cluster* pertencem .

```
(for (int i = 0; i < stories.size(); i++) {
       File file = new File("Dados para processar/generos/" +
               stories.get(i).calculateCenter() + "/" +
               stories.get(i).getStory().getTitle() + ".txt");
        if (!file.exists()) {
               file.createNewFile();
        }

       FileWriter fw = new FileWriter(file.getAbsoluteFile());
       BufferedWriter bw = new BufferedWriter(fw);
       bw.write(stories.get(i).getChaptersString());
       bw.close();
}
```

Código 34).

```
for (int i = 0; i < stories.size(); i++) {
       File file = new File("Dados para processar/generos/" +
               stories.get(i).calculateCenter() + "/" +
               stories.get(i).getStory().getTitle() + ".txt");
        if (!file.exists()) {
               file.createNewFile();
        }

       FileWriter fw = new FileWriter(file.getAbsoluteFile());
       BufferedWriter bw = new BufferedWriter(fw);
       bw.write(stories.get(i).getChaptersString());
       bw.close();
}
```

Código 34 – Arquivamento dos textos nas pastas

Os textos são guardados em pastas locais com o objetivo de serem analisados por alguém competente e que possa perceber que géneros literários não classificados existem na base de dados.

Como os idiomas também podem ser classificados como "Outro", uma vez que só são suportados dois idiomas (português e inglês), também foi implementada a deteção de novos idiomas (Código 35).

```
ArrayList<Story> stories = SDatabase.getStoryByLanguage(Language.OUTRO);
ArrayList<String> languages = new ArrayList<>();
for (Story story: stories){
     languages.add(getLanguage(story));
}
ArrayList<String> folderNames = countDiffent(languages);
createFolders(folderNames);
addFilesToFolders(stories, languages);
```

Código 35 -Obtenção de novos idiomas

À semelhança da deteção de novos géneros literários e como foi dito anteriormente, são escolhidos os textos classificados como "Outro" e é utilizando o mesmo método usado para detetar os idiomas da primeira vez (Secção 5.1.4) (Código 36).

```
String text = story.getChaptersString();
DetectorFactory.loadProfile("Language profiles");
Detector detector = DetectorFactory.create();
detector.append(text);
String lang = detector.detect();
return lang;
```

Código 36 - Deteção do idioma

Depois de detetado o idioma e, à semelhança dos géneros literários, são criadas pastas e os textos inseridos nestas de acordo com o seu idioma.

5.1.9 Base de Dados NoSQL

Como já foi referido na Secção 4.1.1, esta aplicação utiliza uma base de dados NoSQL para guardar os dados tratados pela aplicação.

Para aceder a esta base de dados, foram criadas três classes: duas que lidam com os documentos do tipo *User_Story* (histórias por tratar) e *Story* (histórias já tratadas), uma outra para tratar os documentos *Story_View* (histórias lidas) e a última para *Story_Recommendations* (histórias recomendadas).

Para os documentos do tipo *User_Story* não existe necessidade de um método para os inserir na base de dados (uma vez que a responsabilidade de inserir novas histórias nesta base de dados é da aplicação de sincronização), apenas de as carregar para a aplicação e remover depois de tratadas (Código 37).

```
public void removeNewStory(int _id) {
        MongoDatabase db = this._database.retrieveConnection();
        db.getCollection("User_Story").deleteMany(
               new Document("story_id", _id));
}

public ArrayList<Story> getNewStories() {
        ArrayList<Story> stories = new ArrayList<>();
        MongoDatabase _connection = this._database.retrieveConnection();
        FindIterable<Document> iterable =
_connection.getCollection("User_Story").find();
        iterable.forEach(new Block<Document>() {
            @Override
            public void apply(final Document document) {
                Object tmp[] = document.values().toArray();
                ArrayList<Chapter> chapters =
                      getChapters(tmp[5].toString());
                Story story = new Story(tmp[0].toString(),
                      Integer.parseInt(tmp[1].toString()),
                      Integer.parseInt(tmp[2].toString()),
                      tmp[3].toString(), chapters);
                story.setFanFiction(Boolean.valueOf(tmp[4].toString()));
                stories.add(story);
            }
        });
        return stories;
    }
```

Código 37 - Obtenção e remoção de novas histórias

Para os documentos do tipo *Story*, ou seja, as histórias já analisadas, têm três ações possíveis associadas: a sua inserção (Código 38), a obtenção da lista de histórias existentes (Código 39) e a remoção de histórias (Código 40).

```
public void insertStory(Story story, Language language) {
        if (Language.OUTRO != language)
            createFullStory(story);
        else createAlfStory(story);
    }

private void createAlfStory(Story story) {
        MongoDatabase db = this._database.retrieveConnection();
        ArrayList<Document> genres = createGenreDoc(story.getGenres());
        ArrayList<Document> chapters = createChapDoc(story.getChapters());
        ArrayList<Document> chars = createCharsDoc(story.getChars());
        db.getCollection("Story").insertOne(new Document()
                .append("user_id", story.getAuthorId())
                .append("story_id", story.get_story_id())
                .append("title", story.getTitle())
                .append("isFF", story.isFanFiction())
                .append("chapters", chapters)
                .append("language",
        Language.transformStringIntoLanguage(story.getLanguage())));
}

private void createFullStory(Story story) {
         MongoDatabase db = this._database.retrieveConnection();
         ArrayList<Document> genres = createGenreDoc(story.getGenres());
         ArrayList<Document> chapters = createChapDoc(story.getChapters());
```

```
         ArrayList<Document> chars = createCharsDoc(story.getChars());
         db.getCollection("Story").insertOne(new Document()
                        .append("user_id", story.getAuthorId())
                        .append("story_id", story.get_story_id())
                        .append("title", story.getTitle())
                        .append("genres", genres)
                        .append("over18", story.isOver18())
                        .append("resume", story.getResume())
                        .append("isFF", story.isFanFiction())
                        .append("characters", chars)
                        .append("chapters", chapters)
                        .append("language",
        Language.transformStringIntoLanguage(story.getLanguage())));
}
```

Código 38 - Criação de histórias analisadas

```
public ArrayList<Story> getStories() {
        MongoDatabase _connection = this._database.retrieveConnection();
        FindIterable<Document> iterable =
        _connection.getCollection("Story").find();
        return iterateStories(iterable);
}

private ArrayList<Story> iterateStories(FindIterable<Document> iterable) {
        ArrayList<Story> stories = new ArrayList<>();
        iterable.forEach(new Block<Document>() {
            @Override
            public void apply(final Document document) {
                Object tmp[] = document.values().toArray();
                ArrayList<Genre> genres = getGenres(tmp[4].toString());
                ArrayList<String> chars = getCharacters(tmp[8].toString());
                ArrayList<Chapter> chapters =
                      getChapters(tmp[9].toString());
                Story story = new Story(tmp[0].toString(),
                      Integer.parseInt(tmp[1].toString()),
                      Integer.parseInt(tmp[2].toString()),
                      tmp[3].toString(), chapters);
                story.setGenres(genres);
                story.setOver18(Boolean.valueOf(tmp[5].toString()));
                story.setFanFiction(Boolean.valueOf(tmp[7].toString()));
                story.setResume(tmp[6].toString());
                story.setLanguage(Language
                      .transformStringIntoLanguage(tmp[10].toString()));
                story.setChars(chars);
                stories.add(story);
            }
        });
     return stories;
}
```

Código 39 – Obtenção da lista de histórias analisadas

```
public void removeStory(int _id) {
        MongoDatabase db = this._database.retrieveConnection();
        db.getCollection("Story").deleteMany(new Document("story_id", _id));
}
```

Código 40 - Remoção de histórias

No caso das *Story_View*, isto é, o registo das leituras do utilizador, a consulta é a única ação disponível, uma vez que este tipo de dados é criado no servidor, aquando de uma leitura e são copiados para a base de dados NoSQL (Código 41).

```
public ArrayList<Story> getStoriesRead(User user) {
        ArrayList<Story> stories = new ArrayList<>();
        MongoDatabase _connection = this._database.retrieveConnection();
        FindIterable<Document> iterable =
               _connection.getCollection("Story_View").find(new
               Document("user_id", user.getId()));
        iterable.forEach(new Block<Document>() {
            @Override
            public void apply(final Document document) {
                System.out.println(document.values());
                Object tmp[] = document.values().toArray();
                FindIterable<Document> storiesIterable =
                       _connection.getCollection("Story")
                       .find(new Document("story_id",
                         Integer.parseInt(tmp[2].toString())));
                storiesIterable.forEach(new Block<Document>() {
                    @Override
                    public void apply(final Document document) {
                        Object tmp[] = document.values().toArray();
                        ArrayList<Genre> genres =
                             getGenres(tmp[4].toString());
                        ArrayList<String> chars =
                             getCharacters(tmp[8].toString());
                        ArrayList<Chapter> chapters =
                             getChapters(tmp[9].toString());
                        Story story = new Story(tmp[0].toString(),
                             Integer.parseInt(tmp[1].toString()),
                             Integer.parseInt(tmp[2].toString()),
                             tmp[3].toString(), chapters);
                        story.setGenres(genres);
                        story.setOver18(Boolean.valueOf(tmp[5].toString()));
                        story.setResume(tmp[6].toString());
                        story.setChars(chars);
                        story.setLanguage(Language
                             .transformStringIntoLanguage(tmp[10]
                             .toString()));
                       story.setFanFiction(Boolean
                                   .valueOf(tmp[7].toString()));
                        stories.add(story););
                    }});
            }});
        return stories;
    }
```

Código 41 - Obtenção das histórias lidas por utilizador

Já no caso das *Story_Recommendations* temos o contrário: como são geradas nesta aplicação para serem usadas no servidor, por isso inseri-las na base de dados é a única ação que faz sentido (Código 42).

```
public void insertStory(int user_id, int story_id, String reason) {
        MongoDatabase db = this._database.retrieveConnection();
        db.getCollection("Story_Recommendation").insertOne(new Document()
                .append("user_id", user_id)
                .append("story_id", story_id)
                .append("reason", reason));
    }
```

Código 42 - Inserção de uma recomendação

5.1.10 Testes às Funcionalidades

Para compreender se as funcionalidades implementadas funcionam de acordo com o previsto, foram realizados testes unitários e testes de integração.

Os testes unitários foram implementados para compreender se cada uma das funcionalidades tem o comportamento esperado.

Os testes de integração foram implementados para perceber se os dados obtidos nas funcionalidades estão a ser corretamente transmitidos às outras componentes do sistema (base de dados).

Testes Unitários

Foram realizados 5 testes unitários (Tabela 1), um para cada uma das deteções implementadas, utilizando para isso a biblioteca *JUnit*.

Para a realização do teste à deteção idioma e resumo foi utilizado uma parte do dataset, mais concretamente os textos classificados como “Aventura”.

Para os testes à deteção do género literário, personagens e conteúdo impróprio foi utilizada a história disponível no Anexo 6, história esta que é conhecido o seu género, personagens e se tem conteúdo impróprio ou não.

Tabela 1 - Testes unitários: aplicação para análise automática

Teste	Testado	Objetivo	Dados de *input*	Resultado Esperado	Resultado obtido	Tempo decorrido
1	Género Literário	Testar se é encontrado pelo menos um género.	História Anexo 6	Detetar pelo menos "Aventura"	Aventura, Comédia, Fantasia, Sci-fi	2,06 minutos
2	Idioma	Testar se é encontrado o idioma correto	65 testes	Classificar todas como "Inglês"	Todas classificadas como "Inglês"	10,28 segundos
3	Resumo	Testar se é gerado um resumo	65 testes	Gerar um resumo não nulo (com texto)	É gerado um resumo para todas as 69 histórias	1,33 minutos
4	Personagens	Testar se são encontradas as personagens duma história	História Anexo 6	Detetar pelo menos os seguintes nomes: McGraw, Mustang Sally, Minnie, Benny, John, Stormcrow, Billy	Adams, TwoGuns, McGraw, Mustang Sally, Minnie, Benny, John, Billy, Stormcrow, Hobbitssshh	47,37 segundos
5	Conteúdo Impróprio	Testar se a história é classificada como tendo conteúdo próprio/ Impróprio	História Anexo 6	Detetar a história como conteúdo impróprio	Conteúdo impróprio	277 milissegundos

Dos testes apresentados na Tabela 1 podemos retirar várias conclusões: tal como era expetável, o teste unitário à deteção do género literário é bastante demorado, uma vez que este tem que treinar sete classificadores e só depois classificar.

Também podemos ver que a uma história são detetados vários géneros literários. Esta característica da aplicação ajuda a mitigar o problema da deteção dos géneros mais gerais.

Da análise dos mesmos testes, podemos também verificar que a biblioteca NER de *Stanford*, para além das personagens identificadas manualmente na história, também identifica outras.

Testes de Integração

Como esta aplicação comunica com componentes externos a esta (mais concretamente a base de dados NoSQL), foi necessário implementar testes de integração, para compreender se as informações recolhidas das histórias estão a ser guardadas corretamente. Assim como nos testes anteriores, nos testes de integração também foi utilizado o *JUnit*.

Foram então idealizados dois testes: um para verificar se as histórias estão a ser corretamente guardadas na base de dados, e um segundo teste que verifica se as recomendações são corretamente criadas.

Para a realização destes testes será utilizada novamente a história presente no Anexo 6 para o teste à criação da história.

Já para o teste às recomendações, foram utilizadas duas visualizações e uma terceira história, em que as histórias têm idiomas diferentes, mas duas delas partilham várias personagens.

Tabela 2 - Testes de integração: aplicação para análise automática

Teste	Testado	Objetivo	Dados de *input*	Resultado Esperado	Resultado obtido	Tempo decorrido
1	Adição de uma história à base de dados	Testar se a aplicação guarda corretamente os dados	História 6 e valores detetados	Todos os valores estão corretos	Os valores foram guardados corretamente	2,33 minutos
2	Adição de uma recomendação à base de dados		{user_id : 2, story_id : 3} {user_id: 2, story_id : 1}	Deverá recomendar de acordo com o género e gravar na base de dados	{story_id: 2, user_id : 2, reason : "Characters"}	18,14 segundos

Analisando a Tabela 2, podemos ver que os testes correram de acordo com o expectável e os dados foram guardados corretamente na base de dados.

5.2 Aplicação para Sincronização das Base de Dados

Tal como foi explicado na Secção 4.3, a aplicação de sincronização entre base de dados é uma aplicação independente e implementa o padrão de produtor/consumidor para tentar que partes do processamento mais demoradas possam estar separadas e serem adicionados recursos a estas, quando necessário.

Nesta aplicação existem dois tipos de produtores (Código 43) e consumidores (Código 44): produtor e consumidor para a base de dados MySQL e produtor e consumidor para a base de dados NoSQL, sendo que o consumidor MySQL consome os dados das *queues* do produtor NoSQL e vice-versa.

```
 while (isRunning()) {
     ArrayList<Story> stories = storyDatabase.getStories();
     ArrayList<Story_Recommendation> storyRecommendations =
            storyRecommendationDatabase.getStoriesRecommendations();
     if (!stories.isEmpty()) {
           for (Story story : stories) {
               storyQueue.put(story);
               storyDatabase.setSync(story.get_story_id());
           }
      }
      if (!storyRecommendations.isEmpty()) {
            for (Story_Recommendation story : storyRecommendations) {
                 recQueue.put(story)
                 storyRecommendationDatabase.setSync(story.getId());
            }
      }
 }
```

Código 43 – Exemplo de produtor

```
while (producer.isRunning()) {
        for (int i=0;i<100;i++){
             if(recQueue.isEmpty()) break;
             else {
                  Story_Recommendation storyRecommendation
                          =(Story_Recommendation)recQueue.take();
                  recommendationDatabase.insertStoryRecommendation
                          (storyRecommendation);
             }
        }
        for (int i=0;i<100;i++){
             if(storyQueue.isEmpty()) break;
             else {
                   Story storyR =(Story)storyQueue.take();
                   storyDatabase.editStory(storyR);
             }
        }
 }
```

Código 44 – Exemplo de consumidor

A implementação destes consumidores é bastante idêntica: apenas muda as *queues* em que escreve/lê e a base de dados a que se ligam.

A cada um destes pares estão associadas duas *queues*: ambos os pares têm uma *queue* para histórias, o par MySQL-NoSQL tem uma *queue* para as visualizações e a NoSQL-MySQL para as recomendações.

Para proteger os dados partilhados, ou seja, para que os produtores não escrevam todos ao mesmo tempo na *queue* ou que os consumidores retirem histórias da *queue* ao mesmo tempo, foi utilizada uma *queue* (lista ligada) com bloqueio (Código 45).

Esta *queue* apenas bloqueia parcialmente os dados: apenas um produtor pode escrever e um consumidor ler, mas um produtor pode estar a escrever e um consumidor a ler exatamente ao

mesmo tempo. Isto é possível pois a *queue* ordena os elementos para que o primeiro a ser inserido é o primeiro a sair (FIFO), logo o consumidor retira do início e o produtor insere no fim.

```
LinkedBlockingQueue mySQLToNoSQLStoriesQueue = new
        LinkedBlockingQueue(1000);
LinkedBlockingQueue mySQLToNoSQLViewsQueue = new LinkedBlockingQueue(1000);
```

Código 45- Exemplo de queues bloqueantes

5.2.1 Testes de Integração

Para o caso da aplicação de sincronização, cuja única responsabilidade é sincronizar os dados entre as bases de dados, é importante realizar testes de integração para verificar se de facto esses dados estão a ser transmitidos corretamente. Tal como os testes anteriores, estes testes também foram criados recorrendo ao JUnit.

Assim sendo e como temos dois fluxos de dados (MySQL – NoSQL e NoSQL – MySQL) e três tabelas diferentes envolvidas (*Story*, *Story_View* e *Story_Recommendation*), pelo que teremos quatro testes.

Para este teste são utilizadas duas histórias: a história presente no Anexo 6, que será guardada na base de dados SQL e copiada para a NoSQL, e a história presente no Anexo 7, que fará o percurso inverso.

Para além das histórias será utilizada uma visualização ({user_id : 2, story_id : 3}) e uma recomendação ({story_id: 2, user_id : 2, reason : "Characters"}).

Tabela 3 - Testes de integração: aplicação para sincronização

Teste	Testado	Objetivo	Dados de *input*	Resultado Esperado	Resulta do obtido	Tempo decorrido
1	SQL – NoSQL: História	Testar se a aplicação copia os dados de uma base de dados para a outra	História Anexo 6	Deverá aparecer uma cópia da história na base de dados	Os valores foram copiados	5,23 segundos
2	SQL – NoSQL: Visualização		({user_id : 2, story_id : 3}	Deverá aparecer uma cópia da visualização na base de dados	Os valores foram copiados	2,90 segundos
3	NoSQL – SQL: História		História Anexo 7	Deverá aparecer uma cópia da história na base de dados	Os valores foram copiados	4,25 segundos
4	NoSQL – SQL: Recomendação		{story_id: 2, user_id : 2, reason : "Characters "}	Deverá aparecer uma cópia da recomendação na base de dados	Os valores foram copiados	1,79 segundos

Dos testes anteriores (Tabela 3), podemos concluir que as histórias são sincronizadas corretamente entre as bases de dados.

6 Avaliação dos Classificadores

Do ponto de vista dos testes realizados à plataforma para escrita criativa, mais concretamente à aplicação para análise automática, foram realizados um conjunto de testes aos algoritmos apresentados na Secção 2.4.1. Este testes foram realizados com o objetivo de encontrar o conjunto de algoritmos que melhor resolva cada um dos problemas da deteção do género literário e conteúdo impróprio.

Também foram realizados testes à alternativa à deteção de conteúdo impróprio.

6.1 Planeamento

6.1.1 Extração de Dados de Teste

Para a extração de informação relevante para identificar qual o conjunto de algoritmos que pode resolver com mais precisão cada um dos problemas enunciados anteriormente, foi implementada a técnica de *K-fold cross validation*.

Esta técnica de validação de algoritmos divide o *dataset* em várias partes (k partes) para que, em cada iteração, uma parte será usada para testar o classificador e as restantes utilizadas para o treinar. No final das k iterações, todas as k partes terão sido utilizadas para teste e treino.

Destas iterações serão encontrados resultados que poderão ser classificados como positivos e negativo verdadeiros ou falsos, como podemos ver na Tabela 4(Gomes 2016).

Tabela 4 – Matriz de Confusão

	Classificação Prevista	
Verdadeira Classificação	Positivo	Negativo
Positivo	Verdadeiro Positivo	Verdadeiro Negativo
Negativo	Falso Positivo	Falso Negativo
	Positivo= VP+FN	Negativo =FP+VN

Com os dados recolhidos destes testes podemos então tentar perceber o erro existente na execução dos algoritmos através das seguintes fórmulas(Gomes 2016):

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{VP} + \boldsymbol{VN}}{\boldsymbol{P} + \boldsymbol{N}} \quad (9)$$

$$\boldsymbol{Precision} = \frac{\boldsymbol{VP}}{\boldsymbol{VP} + \boldsymbol{FP}} \quad (10)$$

$$\boldsymbol{Recall} = \frac{\boldsymbol{VP}}{\boldsymbol{P}} \quad (11)$$

$$\boldsymbol{F - Measure} = 2 * \frac{\boldsymbol{Precision} * \boldsymbol{Recall}}{\boldsymbol{Precision} + \boldsymbol{Recall}} \quad (12)$$

Para além destes cálculos também será medido o tempo demorado a executar o treino e a classificação do conjunto de algoritmos.

Sabendo o que queremos medir nos testes a ser realizados, o próximo passo é definir o que iremos testar concretamente. Na figura seguinte (Figura 35) podemos ver os testes realizados (na Secção 6.2 será explicado porque não foram testados todos os algoritmos implementados).

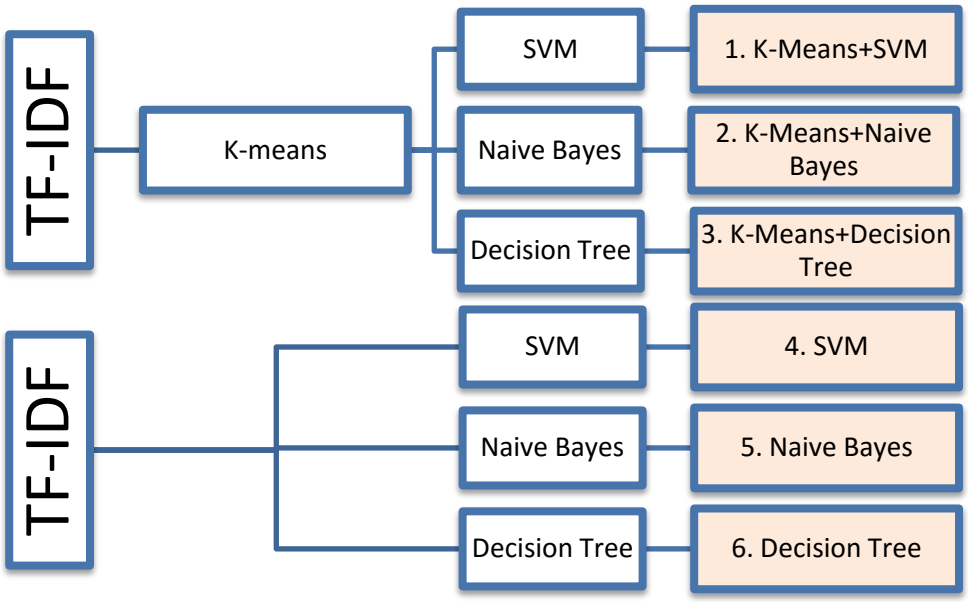

Figura 35 - Combinações dos algoritmos de *clustering*, extratores e classificadores

Como podemos ver na figura anterior, temos 6 combinações possíveis a testar com o dataset de géneros literários e de conteúdo impróprio, ou seja, serão feitos 12 testes diferentes.

Para cada um destes testes, serão retiradas 25 amostragens para os testes aos géneros literários e 20 para o conteúdo impróprio.

Para determinar se, existe alguma diferença entre os conjuntos de algoritmos testados, que leve a escolher uns e não outros, vamos utilizar os testes não paramétricos de *Friedman* e *Nemenyi,* uma vez que a distribuição dos dados não é normal.

6.1.2 Análise dos Dados

Antes de se iniciar a recolha de dados dos testes é importante perceber qual é o peso de cada um dos indicadores de *performance*, no total destes, isto é, cada um destes indicadores recolhidos (*recall*, *precision*, entre outros), deve ter à partida um peso na decisão de qual o conjunto de algoritmos a escolher.

Neste caso concreto, o tempo de execução e a precisão (*precision*) devem ter um maior peso no caso dos géneros literários, enquanto na determinação de conteúdo impróprio as medidas mais importantes serão a precisão e o *recall* (como irá ser explicado mais à frente). No entanto deve existir consistência entre os todos os valores obtidos, para que não seja escolhido um conjunto de algoritmos em que, por exemplo a sua precisão é de 60% e um *recall* de 20%.

Sendo assim, os conjuntos de algoritmos devem respeitar estas três regras:

1. Ter o menor tempo de treino e teste possível;
2. Ter a precisão mais elevada;
3. Deve existir homogeneidade (não deve haver grande discrepância entre os valores) nos valores de *precision*, *accuracy*, *recall* e *f-measure*.

6.1.3 Cross Validation

Para realizar os testes enunciados anteriormente foram criadas classes na aplicação para este efeito (Código 46). Para além de prepararem o *corpus* para os testes (10% para teste e 90% para treino) (Código 47), também calculam o tempo decorrido no treino do classificador e tempo decorrido na classificação dos textos (Código 48).

```
ArrayList<Story> trueAdventure = d.getEngAdventure();
        ArrayList<Story> falseAdventure =
d.generateFalseData(Genre.ADVENTURE, Language.ENGLISH, trueAdventure);
        CrossValidation cvAdventure = new CrossValidation(trueAdventure,
falseAdventure, 4);
cvAdventure.validate(sc, sqlContext, extractor, classifier, "Adventure",
"Genres");
```

Código 46- Exemplo da realização dos testes

```
private void createPartitions() {
        partitions = new ArrayList<>();
        int True = trueData.size() / iterations;
        int False = falseData.size() / iterations;
        int trueCount = 0, falseCount = 0;
        for (int i = 0; i < iterations; i++) {
            ArrayList<Story> tData = getDataPartition(trueCount, trueCount
                      + True, trueData);
            ArrayList<Story> fData = getDataPartition(falseCount,
                      falseCount + False, falseData);
            partitions.add(new Partition(tData, fData));
            falseCount += False;
            trueCount += True;
        }
    }
…
for (int i = 0; i < iterations; i++) {
            ArrayList<Partition> testing = getPartitions(i, 1);
            ArrayList<Partition> training = getPartitions(i + 1, iterations
               - 1);        …
}
```

Código 47 – Criação e utilização das partições (K-fold cross validation)

```
Stopwatch sTrain = new Stopwatch();
…
Double timeTrain = sTrain.elapsedTime();
```

Código 48 – Cronómetro

Para além disso, estas classes também calculam a *accuracy*, *precision*, *recall* e *f-measure*. Estes indicadores de *performance* são então guardados em ficheiros Excel para facilitar a leitura e tratamento dos resultados (Código 49).

```
public void write() throws IOException, WriteException {
        File file = new File(inputFile);
        if(!file.exists()) {
            file.createNewFile();
        }
        WorkbookSettings wbSettings = new WorkbookSettings();
        wbSettings.setLocale(new Locale("en", "EN"));
        WritableWorkbook workbook = Workbook.createWorkbook(file,
               wbSettings);
        workbook.createSheet("Report", 0);
        WritableSheet excelSheet = workbook.getSheet(0);
        createLabel(excelSheet);
        createContent(excelSheet);
        workbook.write();
        workbook.close();
```

Código 49 - Criação dos ficheiros *Excel* com os resultados

6.1.4 Dataset

Para testar os algoritmos foi utilizado o *corpus* criado para a aplicação. Este *corpus* é constituído por 455 histórias, sendo que estas histórias têm uma constituição significativamente diferente umas das outras.

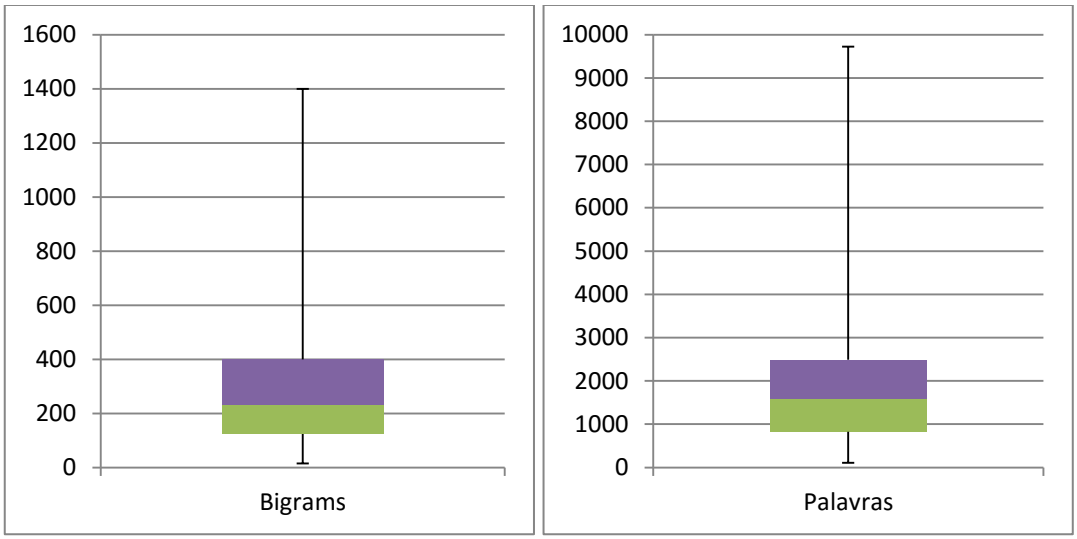

Figura 36 - Distribuição das palavras e bigrams no dataset

Tabela 5 - Estatísticas descritivas dos dados

	Bigrams	***Palavras***
Mean	297,0661	1936,022
Standard Error	11,24454	72,58569
Median	230	1591
Sample Variance	57403,64	2391982
Maximum	1400	9721
Minimum	15	105

Assim sendo e como podemos ver na Figura 36 e na Tabela 5, a história mais pequena contem apenas 105 palavras, enquanto a maior contém 9721, sendo que o número médio de palavras por história é de 1936,022.

Como a criação de bigrams fez parte do pré-processamento dos textos, também foi analisado o número de bigrams criados por história. A história com menos bigrams continha 15 bigrams e a com mais 1400 bigrams, sendo a média de bigrams por história é de 297,0661.

Da análise da Figura 36 podemos observar alguns *outliers*, ou seja, documentos com um número de termos (palavras, *bigrams*) bastante diferente — no caso muito maior — que os restantes. Estes *outliers* correspondem a textos próprios do problema em estudo e, portanto, mantiveram-se no corpus em análise. Os *outliers* correspondem a capítulos integrais ou

mesmo a uma agregação de vários capítulos de livros enquanto os restantes são *short stories*, de uma forma geral mais curtos. Como o corpus resulta da junção de textos de várias fontes diferentes, esta heterogeneidade surge de forma natural.

6.2 Resultados

Dos algoritmos implementados na Secção 5.1.1.1, apenas o TF-IDF, *K-means*, *Decision Tree*, *Naive Bayes* e SVM foram realmente testados. Os restantes (*Word2Vec*, LDA e *Gaussian Mixture*) foram removidos na fase de implementação, pois revelaram-se significativamente lentos (por exemplo o *Word2Vec*, juntamente com o LDA e *Decision Tree* demoraram cerca de 425,579 segundos a realizar um teste).

Os testes foram divididos em três partes distintas: testes aos géneros literários, testes ao conteúdo impróprio utilizando os algoritmos de classificação e testes ao conteúdo impróprio utilizando a lista de palavras banidas do motor de busca da *Google*.

6.2.1 Géneros Literários

Para os géneros literários foram realizados 25 testes. Para este caso concreto foram realizados dois tipos de testes diferentes: um primeiro em que os algoritmos aprendiam todos os géneros e depois identificavam concretamente qual o género em questão, e um segundo, em que apenas eram identificados os dados como verdadeiros e falsos (isto é, textos de Aventura e não Aventura) e em que o algoritmo identificava se o texto era ou não daquele género.

Tabela 6 - Resultados dos testes: géneros literários com vários géneros

	Accuracy (%)	Precision (%)	Recall (%)	F-Measure (%)
Decision Tree	2,48538	2,485380117	16,66667	4,3257
	±49,52334	±49,5233427	±332,0977	±86,1933
Naive Bayes	2,873754	3,232391	15,35948	4,461749
	±57,14922	±64,26298	±305,7853	±88,81541

Os testes realizados com o algoritmo ser treinado com todos os géneros foram bastante insatisfatórios (classificação multi-classe). A probabilidade do algoritmo acertar no género é de $\frac{1}{n}$ (Blackstock n.d.) no caso da utilização de um algoritmo simples com dados uniformes. Sendo que neste caso n=7, uma vez que o n é o número de classes diferentes que o algoritmo aprende, a probabilidade de acertar deveria ser $\frac{1}{7}$.

No entanto utilizando um método e como podemos ver na Tabela 6, a precisão do TF-IDF com o *Naive Bayes*, por exemplo é de 3,23%, que é inferior a $\frac{1}{7}$, e por isso este tipo de abordagem foi descartado.

No entanto, a partir da análise dos resultados dos algoritmos de classificação multi-classe, podemos concluir que grandes partes das histórias eram classificadas como comédia.

Depois de uma análise aos textos do *dataset* de comédia concluiu-se que grande parte destes são comédia, mas tem presente um subgénero, isto é, são por exemplo comédias com a presença de aventura ou fantasia, ou comédias românticas.

Tabela 7 - Resultados dos testes: géneros literários com géneros verdadeiros e falsos

	Tempo de Treino	Tempo de Teste	Accuracy (%)	Precision (%)	Recall (%)	F-Measure (%)
Decision Tree	16,23	4,83	51,19	17,09	28	20,94
	±8,21	±1,32	±11,12	±29,27	±45,83	±34,72
Naive Bayes	6,28	4,28	54,72	55	74,82	60,11
	±1,31	±0,96	±11,52	±16,57	±31,38	±20,55
SVM	65,85	3,79	53,61	39,57	72	51,07
	±11,14	±0,59	±3,35	±25,2	±45,83	±32,51
K-Means + DT	15,2	38,77	52,49	40,96	72	51,86
	±8,03	±7,87	±15,13	±27,69	±45,83	±33,59
K-Means + NB	5,05	36,64	52,39	58,15	76,9	61,25
	±0,94	±8,73	±19,93	±23,48	±34,87	±23,1
K-Means + SVM	67,81	47,09	51,36	46,98	83,76	59,84
	±16,8	±12,8	±18,13	±22,96	±37,33	±27,39

Já para o caso em que o algoritmo aprende o que é um género e ou que não é (classificação binária), a probabilidade deverá ser de 50% ($\frac{1}{2}$). Analisando a Tabela 7, a precisão do TF-IDF com o *Naive Bayes*, por exemplo é de 55%, o que é ligeiramente superior.

Como foi dito anteriormente, para tentar compreender se existe alguma diferença entre os algoritmos, foram utilizados o teste de Friedman, e, no caso de existir, o teste de *Nemenyi* para detetar em quais existe diferença.

Os dados foram divididos por tempo de teste, treino, *accuracy*, *precision*, *recall* e *f-measure* e em cada um destes foi aplicado o teste de *Friedman*. Este revelou que existe uma diferença entre os algoritmos no tempo de treino, teste, *precision*, *recall* e *f-measure*, mas não na *accuracy* (ou seja os valores de *accuracy* de todos os algoritmos são semelhantes).

Foi então aplicado ao tempo de treino, teste, precision, recall e f-measure o teste de Nemenyi, em que foi testada a premissa *q-stat > q-crit* (Gomes 2016), sendo que *q-crit* é um valor fixo e igual a 4,03 neste caso.

Tabela 8 - Teste de *Nemeyi*: tempo de treino

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		5,1	V	5,46	V	0,35	F	6,79	V	5,58	V
NB				10,56	V	4,75	V	1,69	F	10,68	V
SVM						*5,81*	*V*	12,25	V	0,12	F
KM+DT								6,44	V	5,93	V
KM+NB										12,38	V
KM+ SVM											

Na tabela anterior (Tabela 8) podemos ver que para os casos identificados como Falso (F) a premissa *q-stat > q-crit* não se confirma e por isso rejeitamos a hipótese de que o algoritmos têm *performances* semelhantes, exceto no caso do DT e KM + DT, NB e KM+ NB, e SVM e KM+SVM, isto é os pares de algoritmos têm comportamento semelhante no que toca ao tempo de treino.

Tabela 9 - Teste de *Nemenyi*: tempo de teste

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		1,05	F	2,76	F	6,56	V	6,6	F	8,93	F
NB				1,71	F	7,61	V	7,65	V	9,98	V
SVM						9,32	V	9,36	V	11,69	V
KM+DT								0,04	F	2,37	F
KM+ NB										2,33	F
KM+ SVM											

Já para o caso do tempo de teste (Tabela 9), podemos ver que os algoritmos sem *cluster* (DT, NB e SVM) são semelhantes entre si, assim como os outros (com *cluster*).

Tabela 10 - Teste de *Nemenyi*: *precision*

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		6,73	V	3,83	F	4,01	F	5,26	V	3,82	F
NB				2,89	F	2,72	F	1,47	F	2,91	F
SVM						*0,17*	F	1,4	F	0,02	F
KM+DT								1,25	F	0,19	F
KM+NB										1,44	F
KM+ SVM											

Na precisão (Tabela 10), os algoritmos são semelhantes entre si excetuando os algoritmos DT e NB, e DT e KM+NB (tal como podemos concluir da Tabela 7).

Tabela 11 - Teste de *Nemenyi*: *recall*

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		3,06	F	4,43	V	4,43	V	4,4	V	5,43	V
NB				1,37	F	1,37	F	1,34	F	2,36	F
SVM						*0*	F	0,03	F	1	F
KM+DT								0,03	F	1	F
KM+NB										1,03	F
KM+ SVM											

O mesmo podemos concluir no *Recall* (Tabela 11) e *F-Measure* (Tabela 12): excetuando DT combinado com todos os outros algoritmos (menos o NB) as combinação são semelhantes.

Tabela 12 - Teste de *Nemenyi*: *f-measure*

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		3,76	F	4,5	V	4,42	V	4,06	V	4,6	V
NB				0,74	F	0,66	F	0,31	F	0,84	F
SVM						0,08	F	0,43	F	0,1	F
KM+DT								0,35	F	0,18	F
KM+NB										0,53	F
KM+ SVM											

Dos resultados anteriores podemos concluir que os algoritmos têm comportamentos semelhantes em aspetos como a *precision*, o *recall* e *f-measure*. Isto revela que a utilização de algoritmos de *clustering* para tentar filtrar os dados melhora pouco a performance dos algoritmos, afetando muito mais o seu tempo de treino e teste.

Voltando a analisar a Tabela 7, podemos concluir também que o algoritmo TF-IDF com o classificador *Naive Bayes* são a melhor combinação. Os seus resultados são os que menos variam e estão de acordo com a probabilidade de $\frac{1}{n}$ (Blackstock n.d.), sendo que a precisão do algoritmo é superior a 50%.

Por causa desta generalidade do *dataset* de comédia este algoritmo (*Naive Bayes*) será implementado através da utilização de sete classificadores (um para cada género literário). Os géneros encontrados serão guardados num *array* de géneros de forma a mitigar o problema da generalidade do género comédia e de encontrar todos os géneros literários numa história.

6.2.2 Conteúdo Impróprio – Algoritmos de Classificação

Da mesma forma foram analisados os algoritmos para o sub-dataset de conteúdo.

Aos indicadores de *performance* (tempo de teste, treino, *accuracy*, *precision*, *recall* e *f-measure*) foram aplicados o teste de *Friedman*, tendo sido obtido o mesmo resultado que nos

géneros literários (revela diferença entre os algoritmos no tempo de treino, teste, *precision*, *recall* e *f-measure*, mas não na *accuracy*).

A esta aplicação dos algoritmos também foi aplicado o teste de *Nemenyi*, testando a mesma premissa (*q-stat > q-crit)*, sendo que *q-crit* é igual ao dos géneros literários (4,03).

Tabela 13 - Teste de *Nemenyi*: tempo de treino

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		5,62	V	3,48	F	1,62	F	6,29	V	1,33	F
NB				10	V	4	F	0,67	F	6,95	V
SVM						5,1	V	9,76	V	2,15	F
KM+DT								4,67	V	7,62	V
KM+NB										1,03	F
KM+ SVM											

Na tabela anterior (Tabela 13) podemos ver que para os casos identificados como Falso (F) a premissa *q-stat > q-crit* não se confirma e por isso rejeitamos a hipótese de que o algoritmos têm *performances* semelhantes, exceto no caso do DT com todos os algoritmos exceto o NB e KM+NB, NB e KM+DT, NB e KM+NB, SVM e KM+SVM e KM+NV e KM+SVM.

Tabela 14 - Teste de *Nemenyi*: tempo de teste

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		2,46	F	2,48	F	2,97	F	5,3	V	6,52	V
NB				0,02	F	7,42	V	7,76	V	9,98	V
SVM						*7,44*	V	7,78	V	6,54	V
KM+DT								0,33	F	0,91	F
KM+NB										1,24	F
KM+ SVM											

Já no caso do tempo de teste (Tabela 14), rejeitamos todas as combinações (os algoritmos não têm *performances* semelhantes) exceto DT e NB, DT e KM+DT, NB e SVM, KM+DT e KM+NB, KM+DT e KM+ SVM, e KM+NB e KM+ SVM.

Tabela 15 - Teste de *Nemenyi*: *precision*

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		6,47	V	0	F	0	F	3,55	F	0	F
NB				6,47	V	6,47	V	2,91	F	6,47	V
SVM						*0*	F	3,55	F	0	F
KM+DT								3,55	F	0	F
KM+NB										3,55	F
KM+ SVM											

Tabela 16 - Teste de *Nemenyi*: *recall*

	DT	NB		SVM		KM + DT		KM + NB		KM +SVM	
DT		6,17	V	0	F	0	F	0	F	0	F
NB				6,17	V	6,17	V	6,17	V	6,17	V
SVM						*0*	*F*	*0*	*F*	*0*	*F*
KM+DT								0	F	0	F
KM+NB										0	F
KM+ SVM											

Tabela 17 - Teste de *Nemenyi*: *f-measure*

	DT	NB		SVM		KM + DT		KM + NB		KM+SVM	
DT		6,46	V	0	F	0	F	3,57	F	0	F
NB				6,46	V	6,46	V	2,89	F	6,46	V
SVM						*0*	F	3,57	F	0	F
KM+DT								3,57	F	0	F
KM+NB										3,57	F
KM+ SVM											

Analisando a precisão (Tabela 15), *recall* (Tabela 16) e *f-measure* (Tabela 17), podemos ver que a maior parte dos algoritmos são semelhantes.

Dos resultados anteriores podemos concluir, que tal como nos géneros literários e salvo exceções, os algoritmos têm comportamentos semelhantes nas medidas (*precision*, *recall* e *f-measure*), mas são significativamente diferentes no que toca a tempos de treino e teste.

No entanto a semelhança encontrada nos algoritmos é significativamente pior do que nos géneros literários. Como podemos ver na Tabela 18, apenas o *Naive Bayes* (com e sem *cluster*) conseguiu identificar alguns textos positivos (com conteúdo impróprio). No entanto e por ser um campo muito mais sensível do que os géneros literários (uma vez que vai censurar textos impróprios para algumas pessoas), o *recall* é uma medida muito importante (tal como já foi explicado anteriormente), pois o importante é censurar o que é impróprio, mesmo à custa do que é próprio para leitura.

Tabela 18 - Resultados dos testes: conteúdo impróprio – algoritmos de classificação

	Tempo de Treino	Tempo de Teste	Accuracy (%)	Precision (%)	Recall (%)	F-Measure (%)
Decision Tree	40,95	2,56	45,45	0	0	0
	±11,95	±1,14	±9,48	±0	±0	±0
Naive Bayes	16,32	1,82	48,64	56,57	33,33	38,42
	±0,76	±0,25	±18,48	±36,57	±25,36	±25,2
SVM	22,17	2,45	45,45	0	0	0
	±22,17	±2,44	±0	±0	±0	±0
K-Means + DT	35,38	14,72	45,45	0	0	0
	±4,47	±2,23	±0	±0	±0	±0
K-Means + NB	16,33	15,14	42,73	22,25	14,17	16,49
	±1,71	±2,81	±13,22	±26,5	±22,47	±22,98
K-Means + SVM	234,28	18,04	45,45	0	0	0
	±14,47	±4,28	±0	±0	±0	±0

Por este facto, nenhum algoritmo foi escolhido e foi então implementada uma segunda solução que possa dar uma resposta melhor.

6.2.3 Conteúdo Impróprio – Lista de Palavras

Para encontrar uma forma melhor para classificar conteúdo como próprio e impróprio. A *Google*, no seu motor de busca, utiliza um conjunto de palavras[17] que não aparecem neste (palavras com conteúdo sexual, ofensivo, etc.).

Esta lista foi utilizada para encontrar conteúdo impróprio no texto (através da comparação das palavras).

Foram então definidos um conjunto de testes, podendo estes ser divididos em número absoluto de palavras (número de palavras impróprias presentes num texto) e percentagem de palavras impróprias.

[17] https://gist.github.com/jamiew/1112488

Tabela 19 - Resultados dos testes: conteúdo Impróprio – lista de palavras

	VP	FP	VN	FN	Accuracy	Recall	Precision	F-Measure
100 Palavras	0	127	108	0	45,95745	0	0	0
80 Palavras	0	127	108	0	45,95745	0	0	0
50 Palavras	0	127	107	1	45,53191	0	0	0
20 Palavras	6	121	104	4	46,80851	4,724409	4,724409	4,7244094
10 Palavras	17	110	101	7	50,21277	13,38583	13,38583	13,385827
9 Palavras	19	108	100	8	50,6383	14,96063	14,96063	14,96063
8 Palavras	19	108	88	20	45,53191	14,96063	14,96063	14,96063
7 Palavras	23	104	81	27	44,25532	18,11024	18,11024	18,110236
6 Palavras	29	98	87	21	49,3617	22,83465	22,83465	22,834646
5 Palavras	37	90	82	26	50,6383	29,13386	29,13386	29,133858
50% Palavras	0	127	108	0	45,95745	0	0	0
20% Palavras	0	127	108	0	45,95745	0	0	0
10% Palavras	0	127	108	0	45,95745	0	0	0
7% Palavras	0	127	108	0	45,95745	0	0	0
5% Palavras	0	127	108	0	45,95745	0	0	0
1% Palavras	18	109	105	3	52,34043	14,17323	14,17323	14,173228
0.5% Palavras	45	82	79	29	52,76596	35,43307	35,43307	35,433071
0.1% Palavras	89	38	57	65	62,12766	70,0787402	70,0787402	70,0787402

Como podemos ver na Tabela 19, o *recall* dos testes com o número absoluto de palavras é bastante baixo, variando entre os 0% e os 29,13%.

É nas percentagens de palavras que encontramos a solução: 0,1% de palavras num texto tem um *recall* de 70.09%. Esta percentagem é obtida às custas de um número elevado de falsos positivos.

No entanto e como foi dito anteriormente, um falso positivo, neste caso é menos prejudicial que um falso negativo, uma vez que um texto próprio classificado como impróprio pode ser lido por alguém responsável e nada acontecer, mas o contrário (um texto impróprio classificado como próprio lido, por exemplo, por uma criança) pode ser muito prejudicial.

7 Testes

7.1 Planeamento

Para além dos testes aos algoritmos, também foram realizados testes ao sistema em si. Estes testes foram realizados tendo em conta os casos de uso apresentados na Secção 2.1.2 para o caso da aplicação para análise automática (Figura 3).

Analisando cada um dos casos de uso, podemos então fazer os seguintes testes (Tabela 20):

Tabela 20 - Testes a realizar

Use Case	Teste	Como Testar
1- Vê Estatísticas	A. Analisar se o número de histórias para cada género literário corresponde ao número de histórias com esse género na base de dados.	Comparar os dados existentes na base de dados com os dados que a aplicação apresenta
	B. Analisar se o número de histórias para cada idioma corresponde ao número de histórias com esse idioma na base de dados	
	C. Analisar se o número de histórias para conteúdo próprio e impróprio corresponde ao número de histórias com esse conteúdo na base de dados	
2 – Encontra Novos Géneros e Idiomas	D. Analisar se o número de histórias guardadas para análise é o mesmo que os guardados na base de dados sem classificação	Comparar os dados existentes na base de dados com os dados guardados localmente para análise
3 – Atualiza as Informações das Histórias	E. Analisar se uma história é sincronizada da base de dados SQL para a NoSQL	Inserir uma história na base de dados SQL e verificar se foi copiada para a base de dados NoSQL
	F. Analisar se é adicionado à história um resumo	Dada uma história, verificar se é criado um resumo
	G. Analisar se é adicionado à história um conjunto de personagens	Dada uma história, verificar se são encontradas todas as personagens
	H. Analisar se é adicionado à história a classificação como conteúdo próprio/impróprio	Dada uma história, verificar se é classificada como conteúdo próprio/ impróprio
	I. Analisar se é adicionado à história um conjunto de géneros	Dada uma história, verificar se é adicionado um ou mais géneros
	J. Analisar se é adicionado à história um idioma	Dada uma história, verificar se é adicionado o idioma
	K. Analisar se uma história é sincronizada da base de dados NoSQL para a SQL	Verificar se a história processada na base de dados NoSQL é copiada para a base de dados SQL

4- Recomenda Histórias Baseando-se nas Leituras dos Utilizadores	L.	Analisar se uma visualização é sincronizada da base de dados SQL para a NoSQL	Inserir uma visualização na base de dados SQL e verificar se foi copiada para a base de dados NoSQL
	M.	Analisar se é criada uma recomendação	Verificar se é criada uma recomendação na base de dados
	N.	Analisar se a recomendação é única	Verificar se não existe mais nenhuma recomendação igual, isto é que nenhuma história seja recomendada duas vezes ao mesmo utilizador, pela mesma razão
	O.	Analisar se uma recomendação é sincronizada da base de dados NoSQL para a SQL	Inserir uma recomendação na base de dados NoSQL e verificar se foi copiada para a base de dados SQL

Como dados de *input* serão utilizados dados presentes nas bases de dados e a história presente no Anexo 6. Desta história são conhecidas as suas personagens, idioma, género literário e se é conteúdo impróprio ou não.

Os testes anteriores serão realizados do ponto de vista do utilizador, ou seja, será recriada uma utilização normal da aplicação. Através deste testes serão analisados cada um dos pontos anteriores com o objetivo de perceber se o caso de uso é satisfeito.

7.2 Resultados

As aplicações finais (aplicação para análise automática e a aplicação de sincronização) foram testadas em conjunto e tendo em conta o que foi mencionado na Secção 7.1. Os resultados foram os seguintes (Tabela 21):

Tabela 21- Resultados dos testes de sistema

Use Case	Teste	Dados de Entrada	Resultado Esperado	Resultado	Passou /Falhou
1	A	Três histórias com os géneros: **1.** Nan **2.** Nan **3.** Aventura, Comédia, Fantasia, Sci-Fi, Terror	2 histórias Nan 1 história Aventura 1 história Comédia 1 história Fantasia 1 história Sci-Fi 1 história Terror	Uma porção maior do gráfico como Nan e porções iguais para Aventura, Comédia, Fantasia, Sci-Fi, Terror	**PASSOU**
	B	Três histórias com os idiomas: **1.** outro **2.** inglês **3.** inglês	1 história "outro" 2 histórias "inglês"	Mais de metade do gráfico como inglês, o restante como outro	**PASSOU**
	C	Três histórias com conteúdo: **1.** Próprio **2.**Próprio **3.**Impróprio	2 histórias próprias 1 história imprópria	Mais de metade do gráfico como conteúdo próprio, o restante como impróprio	**PASSOU**
2	D	Duas histórias classificadas com o género Nan e uma classificada com o idioma "outro"	2 histórias guardadas em uma ou duas pastas criadas no *clustering* e uma história numa pasta com o nome do idioma a que pertence	2 histórias numa das dez pastas criadas durante o *clustering* e uma história numa pasta com o nome do idioma	**PASSOU**
3	E	Anexo 6	A história é copiada tal como apresentada no Anexo 6	A informação é apresentada como no Anexo 6	**PASSOU**
	F		Deve ser criado um resumo com frases do texto.	Under the gaze of the noon-day sun the land was quiet. The gentle whickering of a horse was the only sound to disturb the hot, heavy hush until it too became quiet. Silence ruled in this dusty place. At the edge of town a man approached through the rippling haze. He placed one foot in front of the other in the manner of one who has been beaten down by the incessant sun but not yet broken. Reward $100 to be collected in person from Mayor Silence. Hands twitching just	**PASSOU**

				above gun level the man took one last gulp of baked air, dropped his hands back to his sides and strode towards the bar. The barman's eyes flicked from the stranger at the door to the man at the bar while the fingers of his right hand fidgeted and fussed with his ample moustache. After much flicking, fidgeting and fussing the man in black grunted and tapped his glass ominously on the bar. "You the one they call Stormcrow boy?" His voice rang out like a colt-45 in a particularly echoey barrel.	
	G		Devem ser detetados, no mínimo, os seguintes nomes: McGraw, Mustang Sally, Minnie, Benny, John, Stormcrow, Billy	Foram detetados: Adams, TwoGuns, McGraw, Mustang Sally, Minnie, Benny, John Billy, Stormcrow, Hobbitssshhh	**PASSOU**
	H		Deve ser detetada como conteúdo impróprio	Detetada como conteúdo impróprio	**PASSOU**
	I		Deve ser classificada, mínimo, como "Aventura"	Classificada como "Aventura", "Comédia" e "Fantasia"	**PASSOU**
	J		Deve ser classificada como "inglês"	É classificada como "inglês"	**PASSOU**
	K		Devem ser copiados os itens enunciados anteriormente para a história original	A informação é copiada	**PASSOU**
4	L	Duas visualizações para o utilizador 2: **1.** {user_id : 2, story_id : 3} **2.** {user_id: 2, story_id : 1} Na base de dados existe mais uma	As visualizações anteriores devem ser copiadas para a base de dados NoSQL.	As visualizações são copiadas	**PASSOU**
	M		Deve ser recomendada a história com personagens iguais, mas não por idioma ou título	{story_id: 2, user_id : 2, reason : "Characters"}	**PASSOU**
	N		Depois de criada	Depois de criada esta	**PASSOU**

		história com algumas personagens iguais a uma das histórias O idioma destas histórias é diferente	esta recomendação não deve repeti-la	recomendação, a aplicação não a volta a recomendar	
	O		A recomendação deve ser copiada para a base de dados SQL	A recomendação é copiada	**PASSOU**

Dos testes anteriores, podemos constatar que o teste à deteção conteúdo impróprio, o teste falhou (à semelhança do que aconteceu nos testes unitários).

Analisando estes testes podemos então concluir que as aplicações estão a funcionar de acordo com os casos de uso pensados no início deste projeto.

8 Conclusão

Na área da escrita criativa já existem vários *websites* funcionais bastante conhecidos e frequentados, quer por escritores, quer por leitores, dos mais variados géneros literários. No entanto, nenhuma destas plataformas implementa técnicas de *data* e *text mining* para melhorar a experiência do utilizador.

Existem várias abordagens possíveis que podem fornecer à plataforma este incremento, tais como *Natural Language Processing* (NLP) e a utilização de algoritmos de *clustering*, extração e classificação.

Através da utilização de NLP, em particular, ou seja através da classificação morfológica das palavras presentes num texto, podemos obter nomes de personagens ou até criar um resumo.

Já com os algoritmos de extração e classificação podemos obter dados como o género literário, provando que é possível determinar o género literário num texto de forma automática é medianamente eficaz.

Com a aplicação destas abordagens foi possível criar uma aplicação que dá suporte à plataforma para escrita criativa, para que os seus utilizadores não tivessem a necessidade que preencher todas as informações da história.

Para além disso, a utilização de NLP propiciou novas formas de recomendar histórias aos utilizadores. Com a deteção de personagens, foi possível passar a recomendar histórias com os mesmos personagens, que é uma recomendação relevante, por exemplo, para leitores de *fanfiction*.

8.1 Limitações e Trabalho Futuro

Como já foi referido, a criação do *dataset* foi um desafio. Apesar dos contactos com várias entidades, nenhuma se mostrou interessada em ajudar a criar o *dataset*. Esta limitação fez com que o *dataset* criado não seja da dimensão desejada e consequentemente que os algoritmos não possam mostrar o seu verdadeiro potencial.

Para além desta limitação, existiu uma limitação ao nível do *website*: por razões exteriores a esta dissertação, o *website* em que deveriam ser utilizados os resultados obtidos na aplicação para análise automática não ficou finalizado.

Futuramente, está planeado um melhoramento do dataset, assim como um melhoramento da deteção do género literário. Para além disso, será terminado o *website* e lançada a plataforma.

Referências

(AXON 2010) AXON, S., 2010. Which Words Does Google Instant Blacklist? Available at: http://mashable.com/2010/09/28/google-instant-blacklist/#z.qT0294hZqY [Accessed May 23, 2016].

(Azad Naik n.d.) Azad Naik, k-means clustering algorithm - Data Clustering Algorithms. Available at: https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm [Accessed January 8, 2016].

(Bezerra n.d.) Bezerra, C.E.B. /Joã. P.R., The EM Algorithm for Gaussian Mixtures The EM Algorithm for Gaussian Mixture Models. , pp.1–4. Available at: http://www.ics.uci.edu/~smyth/courses/cs274/notes/EMnotes.pdf.

(Bakshi 2012) Bakshi, K. (2012). Considerations for big data: Architecture and approach. *IEEE Aerospace Conference Proceedings*, 1–7. http://doi.org/10.1109/AERO.2012.6187357

(Blackstock n.d.) Blackstock, A., Classifying Movie Scripts by Genre with a MEMM Using NLP-Based Features. , pp.1–23.

(Blei et al. 2000) Blei, D.M., Jordan, M.I. & Ng, A.Y., 2000. 10.1162/jmlr.2003.3.4-5.993. *CrossRef Listing of Deleted DOIs*, 1(4–5), pp.993 – 1022. Available at: http://jmlr.csail.mit.edu/papers/v3/blei03a.html [Accessed November 30, 2015].

(Carnevale & Pruitt 1992) Carnevale, P.J. & Pruitt, D.G., 1992. Negotiation And Mediation. *Annual Review of Psychology*, 43(1), pp.531–582.

(Cavalcanti n.d.) Cavalcanti, L. (n.d.). Leitura nos gêneros digitais: abordando as fanfics A tecnologia e os novos gêneros.

(Cemgil 2009) Cemgil, A. T. (2009). Bayesian inference for nonnegative matrix factorisation models. Computational Intelligence and Neuroscience, 2009(2), 785152. http://doi.org/10.1155/2009/785152

(Chen et al. 2012) Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business Intelligence and Analytics: From Big Data To Big Impact. Mis Quarterly, 36(4), 1165–1188. http://doi.org/10.1145/2463676.2463712.

(DELL 2015) DELL, 2015. Text Mining. Available at: http://documents.software.dell.com/statistics/textbook/text-mining.

(EINOV 2014) EINOV, 2014. Modelo de Negócio Modelo de Negócio ? é.

(Fan & Bifet 2013) Fan, W., & Bifet, A. (2013). Mining Big Data : Current Status , and Forecast to the Future. ACM SIGKDD Explorations Newsletter, 14(2), 1–5. http://doi.org/10.1145/2481244.2481246.

(Filzmoser & Vetschera 2008) Filzmoser, M. & Vetschera, R., 2008. A classification of bargaining steps and their impact on negotiation outcomes. *Group Decision and Negotiation*, 17(5), pp.421–443.

(Gomes 2016) Gomes, E.F., 2016. Experimentação e Avaliação.

(Hegland 2003) Hegland, M. (2003). Data Mining – Challenges, Models, Methods and Algorithms. Methods.

(Ibrahim Naji 2013) Ibrahim Naji, 2013. 10 Tips to Improve your Text Classification Algorithm Accuracy and Performance | Thinknook. Available at: http://thinknook.com/10-ways-to-improve-your-classification-algorithm-performance-2013-01-21/ [Accessed May 23, 2016].

(Javier Couto 2015) Javier Couto, 2015. The Definitive Guide to Natural Language Processing.

(Kanungo et al. 2002) Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient k-means clustering algorithm: analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7), 881–892. http://doi.org/10.1109/TPAMI.2002.1017616

(Landset et al. 2015) Landset, S. et al., 2015. A survey of open source tools for machine learning with big data in the Hadoop ecosystem. *Journal of Big Data*, 2(1), p.24. Available at:

http://www.journalofbigdata.com/content/2/1/24 [Accessed November 8, 2015].

(Lobo 2007) Lobo, V., 2007. Classificação Bayesiana Tema central : Dados completos. , pp.1–24.

(Luís Reis 2010) Luís Reis. (2010). A Invenção da Escrita - Um Novo Rumo para a Humanidade. Retrieved from http://www.triplov.com/novaserie.revista/numero_07/luis_reis/index.html.

(Martin & Azmi-Murad 2005) Martin Porter, 2006. Porter Stemming Algorithm. Available at: http://tartarus.org/martin/PorterStemmer/ [Accessed May 23, 2016].

(Michael Gilleland n.d.) Michael Gilleland, M.P.S., Levenshtein Distance. Available at: http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein Distance.htm [Accessed June 4, 2016].

(Minnaar n.d.) Minnaar, A., Word2Vec Tutorial Part I: The Skip-Gram Model. Available at: http://alexminnaar.com/word2vec-tutorial-part-i-the-skip-gram-model.html [Accessed January 11, 2016a].

(Minnaar n.d.) Minnaar, A., Word2Vec Tutorial Part II: The Continuous Bag-of-Words Model. Available at: http://alexminnaar.com/word2vec-tutorial-part-ii-the-continuous-bag-of-words-model.html [Accessed January 11, 2016b].

(Moniruzzaman & Hossain 2013) Moniruzzaman, a B.M. & Hossain, S.A., 2013. NoSQL Database : New Era of Databases for Big data Analytics- Classification , Characteristics and Comparison. *International Journal of Database Theory and Application*, 6(4), pp.1–13.

(Nicola 2016) Nicola, S., 2016. Análise de valor de negócio.

(OSTERWALDER, 2011) OSTERWALDER, Alexander; PIGNEUR, Yves. Business Model Generation: Inovação em Modelos de Negócios. Rio de Janeiro, RJ: Alta Books, 2011

(OSTERWALDER, 2003) OSTERWALDER, Alexander; PIGNEUR, Yves. An ontology for e-business models. Wendy Currie : Value Creation from E-Business Models, 2003.

(Piotr Kowalczyk 2016) Piotr Kowalczyk, 15 most popular fanfiction websites. Available at: http://ebookfriendly.com/fan-fiction-websites/ [Accessed January 4, 2016].

(Reynolds 2008) Reynolds, D. a, 2008. Gaussian Mixture Models. *Encyclopedia of Biometric Recognition*, 31(2), pp.1047–64. Available at: http://www.ncbi.nlm.nih.gov/pubmed/21856417\nhttp://extwebprod.ll.mit.edu/mission/communications/publications/publication-files/full_papers/0802_Reynolds_Biometrics-GMM.pdf.

(Rhiannon Tuffield 2016) Rhiannon Tuffield, 2016. Surviving as a Writer in the Digital Age. Available at: http://www.writersedit.com/survive-writer-digital-age/.

(Richard K. Lomotey and Ralph Deters 2012) Richard K. Lomotey and Ralph Deters, 2012. D Ata M Ining and I Ntegrating. , 2(May), pp.17–29.

(RosettaCode 2016) RosettaCode, 2016. Levenshtein distance - Rosetta Code. Available at: https://rosettacode.org/wiki/Levenshtein_distance#Java [Accessed June 4, 2016].

(Sam 2015) Sam, 2015. You probably shouldn't use a JavaScript Framework | Snugug. Available at: https://snugug.com/musings/you-probably-shouldnt-use-javascript-framework/ [Accessed January 21, 2016].

(SAS n.d.) SAS, Machine Learning - What it is & why it matters. Available at: http://www.sas.com/en_id/insights/analytics/machine-learning.html.

(Shuyo Nakatani n.d.) Shuyo Nakatani, Cybozu Open Source: Language Detection Library for Java. Available at: http://developer.cybozu.co.jp/archives/oss/2010/10/language-detect.html [Accessed May 25, 2016].

(Spark n.d.) Spark, T.D., Feature Extraction and Transformation - spark.mllib - Spark 1.6.0 Documentation. Available at: http://spark.apache.org/docs/latest/mllib-feature-extraction.html [Accessed January 10, 2016].

(Vattani 2011) Vattani, A. (2011). k-means Requires Exponentially Many Iterations Even in the Plane. Discrete & Computational Geometry, 45(4), 596–616.

http://doi.org/10.1007/s00454-011-9340-1

(Verma et al. 2012) Verma, M., Srivastava, M., Chack, N., Diswar, A. K., & Gupta, N. (2012). A Comparative Study of Various Clustering Algorithms in Data Mining Manish Verma , Mauly Srivastava , Neha Chack , Atul Kumar Diswar , Nidhi Gupta, 2(3), 1379–1384.

(Wang & Grimson 2007) Wang, X., & Grimson, E. (2007). Spatial Latent Dirichlet Allocation. Proceedings of Neural Information Processing Systems Conference (NIPS).

(Witten et al. 2011) Witten, I. H., Frank, E., & Hall, M. a. (2011). Data Mining: Practical Machine Learning Tools and Techniques (Google eBook). Complementary literature None. Retrieved from http://books.google.com/books?id=bDtLM8CODsQC&pgis=1

(Wu et al. 2008) Wu, X., Kumar, V., Ross, Q. J., Ghosh, J., Yang, Q., Motoda, H., … Steinberg, D. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems (Vol. 14). http://doi.org/10.1007/s10115-007-0114-2

(Yogesh Raja n.d.) Yogesh Raja, Gaussian Mixture Models. Available at: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAJA/CV.html [Accessed January 21, 2016].

(Zhang et al. 2014) Zhang, H., 2004. The optimality of naive Bayes. *Aa*. Available at: http://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf.

Anexos

Anexo 1

Use Case Section	Comment
Use Case Name	1 - Register in the Platform (Regista-se na plataforma)
Scope	Website
Level	user-goal
Primary Actor	Visitor
Stakeholders and Interests	Visitor: Wants to enter the platform and access to its content.
Preconditions	
Success Guarantee	Visitor registers himself in the platform becoming an authenticated and registered User.
Main Success Scenario	A. Visitor arrives at the platform B. Visitor chooses the normal registration type C. Visitor fills the required information D. Visitor sends the information E. Visitor clicks confirmation link that was sent to email F. Visitor gets authenticated
Extensions	a) At any time, the Visitor can choose to register using a social network: i Visitor arrives at the platform ii Visitor chooses a social network iii Visitor agrees to terms and conditions of said network iv Visitor gets authenticated
Special Requirements	a) Connection to third party applications must be supported such as Facebook, Google +, Outlook. i Connection to Social Networks for register purposes. ii Connection to Email Servers for validation of email purposes (maybe).
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	2 - Edit Profile (Edita perfil)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: Wants to update his profile information
Preconditions	User is identified and authenticated.
Success Guarantee	User's profile information is updated
Main Success Scenario	A. User visits the edit profile page. B. User changes his personal information. C. Before saving the information User is required to provide his current password.
Extensions	a) At any time, the User can change his password:

	i User must provide the old password before changing the current password. b) At any time, the User can add a password: i When registering from Social Networks normally User doesn't get a password, he just gets a Token from the Social Network. In this case User should be able to set a password without providing his old password because it doesn't exist. ii At any time, the Server can fail while updating the new information: iii The User is notified and suggested to try again. c) The user interface maintains all the previous information expect for the passwords. d) At any time, the Server can deny a update because the password was wrong: i The User is notified and suggested to try again with a different password. ii The User interface maintains all the previous information expect for the passwords.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	**Comment**
Use Case Name	3 - Deactivate Account (desativa conta)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: Wants to be able to deactivate his account.
Preconditions	User is identified and authenticated.
Success Guarantee	User's account enters a Disabled state.
Main Success Scenario	A. User requests to deactivate account B. System asks for User password for security measures. C. System deactivates User account
Extensions	a) At any time, the System can deny the deactivation of the User account if the password is wrong: i The user is notified that he entered a wrong password and is asked to try again. b) At any time, the System can fail while deactivating the user account: i The user is notified of the failure and is asked to try again.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	4 - Invite Friends (Convida amigos)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: Wants to be able to invite friends to the website.
Preconditions	User is identified and authenticated.
Success Guarantee	Invites are generated and sent to targets.
Main Success Scenario	A. User visits the friends page. B. User introduces the e-mail of the friend. C. User sends the invite.
Extensions	a) At any time, the User can choose to login using a social network and use his friends list to send an invitation
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	5 - Validate account (Valida conta)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: receives an e-mail to validate account
Preconditions	User account created but not validated.
Success Guarantee	An e-mail is receivedUser with a link that upon opening will lead to the validation of the User account
Main Success Scenario	A. A Visitor tries to create an account. B. The System processes the creation and sends an e-mail to validate the account C. If the Visitor validates the account , the System creates officially the account D. If not, the temporary account is deleted
Extensions	a) If the Visitor is using a social network, the validation is done by the social network
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Anexo 2

Use Case Section	Comment
Use Case Name	1 - Write a Story (Escreve história)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: Wants an interface for writing a story that provides him easy to use features.
Preconditions	User is identified and authenticated
Success Guarantee	Upload associated documents to the server. Story is saved.
Main Success Scenario	A. User arrives at the creation page. B. User starts writing a story C. User selects the categories of the story. D. User uses the functionalities of the editor: a. User can alter the styling of the text. E. User selects optional functionalities: a. User can tell the system that the story is not finished. b. User can define a cover for the story. c. User can define the privacy of the story.
Extensions	a) At any time the User can save a draft of the story i Everything about the story at the current point is saved. ii The story is flagged as not finished. iii User is returned to the main page and receives a notification that the story draft was saved. b) In a set interval (1, 2, 5 minutes depending on the load of the system) the system automatically saves a draft: i Everything about the story at the current point is saved. ii The story is flagged with the previous state. c) At any time the Server fails while saving the story i The user is notified that the server had a problem saving the story. ii The user is asked whether to save the story information at the current point as cache, if he wants to try again or if he prefers to lose the content.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	2 - View a Story (Vê história)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	Visitor: Can view a story basic details. User: Can view a story basic details and can do the following: A. Can like the story B. Can dislike the story.

	C. Can share the story on several social networks D. Can comment / give feedback. E. Can suggest the story to other users. F. Can continue viewing a story at last known chapter of the story. Administrator: Can do everything that was stated before.
Preconditions	To access the User functionalities the User must be identified and authenticated. To access the Administrator functionalities the Administrator must be identified and authenticated.
Success Guarantee	All information was presented correctly.
Main Success Scenario	Visitor / User / Administrator: A. Requests a story to read. B. The story information is presented.
Extensions	a) At any time the User / Administrator can share the story: i The target platform is chosen. ii If the authentication on that platform is required it must be done. iii The story is shared. b) At any time the User / Administrator can cancel the share action: i The user cancels the share. ii The interface returns to its normal state. c) At any time the User / Administrator can like or dislike a story i The specific action is chosen (like or dislike) ii The action is saved on the server. iii The user interface is updated with a new count of likes / dislikes. d) At any time the User / Administrator can comment / give feedback on the story i A textbox is focused. ii The user writes on the textbox. iii The comment is saved. e) At any time the User / Administrator can cancel the comment they are writing. i The user cancels the comment. ii The interface returns to its normal state.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	3 - Bookmark a Story (Adiciona história aos favoritos)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: User wants a way to save his favorite stories. Administrator: Same as User.
Preconditions	User is identified and authenticated Story doesn't belong to current user. Story isn't on User's favorite list already.
Success Guarantee	Bookmark information is saved.
Main Success Scenario	A. User is viewing a story

	B. User bookmarks a story. C. Story is added to User's favorite list. D. Number of bookmarks of story updated accordingly E. The interface is updated with new icon.
Extensions	a) At any time, the Server fails while bookmarking the story: i User is notified of the failure. ii User can try again.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	**Comment**
Use Case Name	4 - Edit a Story (Edita história)
Scope	Website
Level	user-goal
Primary Actor	User
Stakeholders and Interests	User: User wants a way to edit his stories. Administrator: Same as User.
Preconditions	User is identified and authenticated Story belongs to current user.
Success Guarantee	Story is updated with edited content
Main Success Scenario	User/Administrator: A. Requests a story to edit. B. 2. Requested story information is presented in their respective container(s). C. User can edit information as he desires. D. Save updated information.
Extensions	a) At any time, the Server fails while editing the story: i User is notified of the failure. ii User can try again. b) At any time, the user can cancel his alterations. c) In a set interval (1, 2, 5 minutes depending on the load of the system) the system automatically saves a draft: i Everything about the story at the current point is saved. ii The story is flagged with the previous state.
Special Requirements	
Technology and Data Variations List	a) An intermediary data type that represents a story that was edited and saved but not finished so that the presented story is still the one that was finished before. Basically it's an "alternative" story.
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	**Comment**
Use Case Name	5 - View Statistics (Vê estatísticas)
Scope	Website
Level	user-goal
Primary Actor	User

Stakeholders and Interests	User: User wants to know relevant information about the popularity of his stories. Administrator: Same as User.
Preconditions	A. User is identified and authenticated. B. User has stories created.
Success Guarantee	User learns about his stories popularity.
Main Success Scenario	User/Administrator: A. User selects one or more stories from the list. B. Interface is updated with the statistics of the selected stories. C. User learns with that information.
Extensions	a) At any time, the Server fails while retrieving data for a story: i User is notified of the failure. ii User can choose to try again.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Anexo 3

Use Case Section	Comment
Use Case Name	1 - View Statistics (Vê estatísticas)
Scope	Mining App
Level	user-goal
Primary Actor	Administrator
Stakeholders and Interests	Administrator: Administrator wants to know relevant information about the stories inserted in the database.
Preconditions	
Success Guarantee	Administrator learns about his stories popularity.
Main Success Scenario	Administrator: A. Administrator selects the type of statistics he/she wants to see. B. Interface is updated with the statistics. C. Administrator learns with that information.
Extensions	a) At any time, the App fails while retrieving data: i Administrator is notified of the failure. ii Administrator can choose to try again.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	2 - Find New Genres and Languages (Encontra novos géneros e Idiomas)
Scope	Mining App
Level	user-goal
Primary Actor	Administrator
Stakeholders and Interests	Administrator: Administrator wants to find new languages and genres in the stories.
Preconditions	
Success Guarantee	Administrator learns if there are new languages and genres.
Main Success Scenario	Administrator: A. Administrator selects the type cluster he/she wants to do (genre or language). B. System finds the stories with no genres or language. C. System finds the new genres or languages. D. System archives the stories according the new genres or languages in folders.
Extensions	a) At any time, the App fails while retrieving data: i Administrator is notified of the failure. ii Administrator can choose to try again.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	Comment
Use Case Name	3 - Update Stories Information (Atualiza as informações das histórias)

Scope	Mining App
Level	system-goal
Primary Actor	System
Stakeholders and Interests	System: System detects genre, language, resume, characters and mature content in the stories and update its information.
Preconditions	
Success Guarantee	System finds new information about the stories.
Main Success Scenario	System: A. System finds new stories in the database. B. System detects the genre, language, resume, characters and mature content in the stories . C. System saves the new information in the database.
Extensions	a) At any time, the App fails while retrieving data: i System notifies about the failure.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Use Case Section	**Comment**
Use Case Name	4 - Recommend Stories Based on Users Views (Recomenda histórias baseando-se nas leituras dos utilizadores)
Scope	Mining App
Level	system-goal
Primary Actor	System
Stakeholders and Interests	System: System recommends stories to the users .
Preconditions	
Success Guarantee	System finds new information about the user's preferences.
Main Success Scenario	System: A. System gets the stories viewed by the users in the database. B. Based on that information, System recommends new stories. C. System saves the new information in the database.
Extensions	a) At any time, the App fails while retrieving data: i System notifies about the failure.
Special Requirements	
Technology and Data Variations List	
Frequency of Occurrence	Continuous
Miscellaneous	

Anexo 4

```
trainClassifiers();
ArrayList<Story> stories;

while (!stop) {
stories = noSQLDB.getNewStories();
if (!stories.isEmpty()) {
   for (Story story : stories) {
      Language language = Language.OUTRO;
      Mine mineLanguage = new MineLanguage(detector);
      mineLanguage.setData(story);
      boolean succeed = mineLanguage.operate();
      if (succeed) {
          TextResults result = mineLanguage.getData();
          language = result.getLanguage();
          story.setLanguage(language);
      }
      if (language != Language.OUTRO) {
          Thread lGenre = mineGenre(language, story);
          Thread age = mineAge(language, story);
          Thread chars = mineChars(language, story);
          Thread resume = mineResume(language, story);
          lGenre.start();
          age.start();
          chars.start();
          resume.start();

          lGenre.join();
          age.join();
          chars.join();
          resume.join();
          story.setGenres(TRgenre.getGenres());
          story.setOver18(TRage.isOver18());
          story.setChars(TRchars.getChars());
          story.setResume(TRresume.getResume());
      }
      noSQLDB.removeNewStory(story.get_story_id());
      noSQLDB.insertStory(story, language);
 }
} else {
 Thread.sleep(50000);
}
```

Anexo 5

```
ArrayList<User> users = SVDatabase.getDistinctArrayUsers();
for (User user : users) {
   if (!SVDatabase.getStoriesReaded(user).isEmpty()) {
      RecommendByIdiom lang = new RecommendByIdiom(SDatabase, SVDatabase);
      lang.setData(user);
      lang.operate();
      TRlang = lang.getData();
           Language language = lang.getLanguage();
           Thread chars = recommendChars(user);
           chars.run();
           Thread titles = recommendByTitle(user, language);
           titles.run();
           chars.join();
           titles.join();
           if (!TRlang.isEmpty())
               SRDatabase.insertMultipleStories(TRlang, "Language", user);
           if ((!TRchars.isEmpty()))
               SRDatabase.insertMultipleStories(TRchars,
                      "Characters", user);
           if (!TRtitle.isEmpty())
                SRDatabase.insertMultipleStories(TRtitle,
                      "Similar titles", user);
      }
   }
 }
  Thread.sleep(10000);
```

Anexo 6

{
user_id : 1,
story_id : 1,
title : "A Fistful Of Feathers",
chapters : [{
chapter_id : 1,
chapter_number : 1,
title : "A Fistful Of Feathers",
content : "Under the gaze of the noon-day sun the land was quiet. Nothing stirred save the occasional plume of dust coaxed from the ground by the warm, summer breeze. The gentle whickering of a horse was the only sound to disturb the hot, heavy hush until it too became quiet. Silence ruled in this dusty place.At the edge of town a man approached through the rippling haze. He placed one foot in front of the other in the manner of one who has been beaten down by the incessant sun but not yet broken. As he reached the middle of Main Street he paused next to the deserted undertakers with its half-finished coffins spilling out beyond the porch. He spat in to the dirt and crossed to the general store. Above the “back much later” sign on the door was the same wanted poster he’d seen in every one-horse, two-dog, three-times-a-lady outpost this side of Halfrow.Wanted: Dead or Alive. Reward $100 to be collected in person from Mayor Silence.The man nodded slowly. The reward had grown as fast as the legend and today was collection day.Turning back to the street a creaking sound to his right caught his attention. He stiffened as he tried to place it and then relaxed as he watched the rusting sign of the Sheriff’s Badge swinging above the saloon door as it caught the hint of a zephyr. Hands twitching just above gun level the man took one last gulp of baked air, dropped his hands back to his sides and strode towards the bar. As he entered the saloon Silence greeted him. He waved back at the mayor and then turned his attention to the brooding black-clad form hunkered over the bar. On the opposite side of the worn, wooden counter a nervous looking barkeep dabbed at the beads of sweat forming on his balding pate with a threadbare handkerchief. The barman’s eyes flicked from the stranger at the door to the man at the bar while the fingers of his right hand fidgeted and fussed with his ample moustache. After much flicking, fidgeting and fussing the man in black grunted and tapped his glass ominously on the bar. The barkeep flinched at the sound of glass striking wood, picked up the half empty bottle of rye and began to pour. “Leave it.”Deep in his cups the huge bear-like form paid no further mind to the stranger at the door or the recently retreated barkeep. He just sat there in his highly inappropriate midnight cloak, hunched over his liquor, thoughts unknown. If the previously encountered zephyr had been of a mind to wander (as only zephyrs do – real breezes generally trot) in to the saloon, then make its zephyry way to the bar and disturb the thick dark cloak the man wore, the unmistakable movement of feathers would have been witnessed by All and Sundry (resident drunkard and part time blacksmith respectively) as well as everyone else within the Sheriff’s Badge. Zephyrs however are not to be trusted and the one in question merely wandered along Main Street before dying an inevitable and windless death just outside Old Ma Adams’ place. The stranger at the entrance cared nothing for the zephyr’s demise or the ruffled feathers it could have been responsible for. His focus was on the man at the bar. From where he stood staring at his quarry it was difficult to tell where midnight cloak ended and ebony ten-gallon hat began. The effect was disconcerting and it took him a moment to find his voice. “You the one they call Stormcrow boy?” His voice rang out like a colt-45 in a particularly echoey barrel.Around the saloon an entire cast of unmentioned people stopped whatever it was they happened to be doing. Their previous actions didn’t amount to a hill of beans and certainly play no part in this story. The black figure ignored this complete lack of description, raised his glass, drained it and placed it back on the table all in one fluid yet precise motion. “Sit down boy, ” said Stormcrow in a voice that was somehow soft, compelling and amused all in one. His words gave rise to more tension as well as birthing the adverb socompelsedly. The man pulled up a stool and did as he was asked.

Stormcrow produced a second glass and, after filling it to the brim, passed it to the stranger. “Drink.”“Yes it is, ” replied the stranger immediately wishing he could take it back, “but I didn’t come here to drink with you. I came here to kill you and claim my reward from the mayor over there.” As he caught the dark look on Stormcrow’s face the cheery wave from Mayor Silence turned in to a shrugged and somewhat incongruous “What’s a guy gonna do?” Silence wilted under the intense scrutiny and a completely unnecessary “Fuhgeddaboudit” died on his lips. “Why don’t we just get this over with?” said the stranger pushing the still-full glass back towards Stormcrow. “We ain’t getting any younger.” Stormcrow drained both glasses and shook his head. “What’s your name son?” he asked in a weary tone.“Billy. Billy ‘Two-Guns’ McGraw.” “Why ‘Two-Guns’ Billy?” said Stormcrow without looking up.Billy looked embarrassed for a moment and fidgeted in his seat before anger rose in him and he slammed his hand down on the bar. “That’ll be on account of my two guns mister which you’d already know if you knew anything about the wilds of the west. Round these parts I’m considered the greatest two gun shooter of them all! Ain’t nothing I can’t hit at a hundred paces whether I’m blind drunk, sober as a priest or anywhere in between. I’ve killed more men in these parts than most of the fellas in here have had cold dinners or hot women. Mustang Sally, Minnie the Moocher, The Rhinestone Cowboy…I shot the lot of them! Hell, I took out Benny and the rest of his Jets gang on account of him whistling funny! I’ve shot John…” “You’ve only got one gun.”“…and the rest of the Ewing County bandits…wait a minute, whatchoo say?”“Gun, Billy. Singular.” “Well now you just hold on there for one lousy moment mister…I don’t know where you get off casting doubt on good folks and their rightfully earned names…” Billy’s shoulders slumped as some frantic tapping at his left hip confirmed the truth. Stormcrow met Billy’s eye and nodded, conveying as much sympathy as is possible with a single nod.“I don’t believe this, ” said Billy, “I was sure I had it when I came in to town.” His face took on a broken look and his voice assumed a strange English twang. “This was my big chance and I’ve only gone and blown it. I’ll be the bloody laughing stock I will!” Stormcrow considered this for a few moments as he filled and then drained another glass. When he set the glass down his speech was a little slurred, not that you would tell from the way it’s written. “Not necessarily.”Billy looked up at him, eyes filled with hope.“Think, ” said Stormcrow, “Where was the last place you remember having it?” Billy tugged at a nondescript earlobe. “Well…I definitely had it when I left Halfrow…I didn’t stop at Bullethole Ridge or Freckle Gorge…” Billy slapped his forehead, “Of course! I stopped for a quick snack just short of town next to that little grove of oddly shaped cactuses and I must have left it behind.” “Cacti, ” said Stormcrow.“Huh?”“Cacti, ” repeated Stormcrow.Billy glanced down at his chest. “But I’m not wearing one.” “One more of those and I’ll shoot you myself, ” said the barkeep who had re-appeared for no discernible reason.“Sorry, ” said Billy. His face broke in to a wide smile. “Perhaps I should be ‘Two-Puns’ from now on eh? Eh?” Billy laughed alone. “Oh come on, it wasn’t that bad…guys?” Billy stood up in a huff. “Fine, I’ll be off then.” An awkward moment passed fairly awkwardly before Billy let out a nervous little laugh. “So…um…don’t go anywhere then…coz…well…y’know…” Meeting Stormcrow’s bitter stare Billy realised that mimicking the click of a pistol cock, shaping both hands in to guns and hinting at a “PuhPOW!” motion was probably not his finest hour and he dropped his hands back to his sides and walked towards the door. As Billy left the bar and Stormcrow resumed drinking, a strong gust of wind galloped in to the bar ruffling the black feathered cloak draped over Stormcrow’s solid frame. With a look of horror on his face Mayor Silence leapt to his feet. “BILLY! WAIT! IT’S A TRAP! THE WIND ALWAYS BLOWS STRONGEST AT THIS TIME OF DAY AND…”A harsh, metallic grating sound followed by a brief scream, a plaintive gurgle, a moderately dull thud and a spasmodic scratching noise filled the air. “…I’ve been meaning to fix that overly large rusting metal sign above the saloon…” finished the mayor before slumping back in to his chair. “Probably built by bloody Hobbitssshhh, ” muttered Stormcrow before passing out."

}]

}

Anexo 7

{
user_id : 1,
story_id : 2,
title : "A Matter Of Time",
genres : [{ genre: "NaN”}],
over18 : true,
resume : "The silent underground station held all the charm and colour of a mortuary slab. Running between the two empty platforms were a series of circular wooden benches each separated from the other by dull, concrete constructions no doubt used in times past to advertise some useless gadget or a piece of music from the fad-band of the hour. Only one soul occupied the platform and she was oblivious to anything beyond the breath fogging in front of her face. “Weird how it does that isn’t it?”She flinched at the voice from directly behind her and wiped her hand self-consciously against her leg. ” The man barked out a laugh at the abruptness of the response. “You really don’t have any idea what’s happening, do you?” He paused for a moment sensing Lucy’s unseen and soundless shake of the head before adding, “At least tell me you’ve bought your ticket already?” “Ticket?” said Lucy. “And how am I supposed to pay for a ticket? I’m in my best green dress! I don’t have room in it for pockets or purses and my handbag is on the table in the kitchen where I left it after…well, wherever it was I went before I came here and…now what was I saying before I lost my train of thought…train! Ha! That’s funny with me being in the station and all. Facing the stairs she saw a worn sign for tickets pointing to the left of the staircase and as her eyes followed the sign’s instruction she saw a flickering bulb at the back of the platform highlighting a door in ragged white bursts. ",
isFF : false,
characters :[{name : "Lucy"}, {name : "Peters"}, {name : "Charlie"}, {name : "Harry"},],
chapters : [{
chapter_id" : 3,
chapter_number : 1,
title : "A Matter Of Time",
content "The silent underground station held all the charm and colour of a mortuary slab. Running between the two empty platforms were a series of circular wooden benches each separated from the other by dull, concrete constructions no doubt used in times past to advertise some useless gadget or a piece of music from the fad-band of the hour. Despite their purpose nothing was on display and not a single shred of historical eco-solvent, self-adhesive, rainforest-aware billboard paper remained to indicate they had ever carried out their allotted task. It mattered little. Only one soul occupied the platform and she was oblivious to anything beyond the breath fogging in front of her face. The lady in green was sitting on the first of the spherical benches with her back to the stairs, allowing her vision (if it were paying attention) the full run of the platform and the empty tracks on either side. She held her breath for a moment before puffing out the contents of her lungs with as much force as she could muster. As had happened with each previous exhalation her breath oozed out languidly before hanging in a viscous cloud an inch from her nose. She lifted a hand to her face and watched in fascination as tiny droplets of water formed on her fingertips and slid gently towards her palm. “Weird how it does that isn’t it?”She flinched at the voice from directly behind her and wiped her hand self-consciously against her leg. “I didn’t mean to startle you, ” the mellifluous male voice continued, “And no need to turn around, we can get along fine from right where we are.” On any given day Lucy Peters would have confronted the stranger and given him a piece of her mind, but something in her seventy four year old brain told her that today was far from any given day. She turned her head slowly back to face the length of the platform before she replied. “How we’ll get along remains to be seen. I don’t hold too much truck with strangers.” The man barked out a laugh at the abruptness of the response. “Atta-girl Lucy! Good to see you’ve got some fight left in you. I had a feeling you weren’t beaten just yet. Still, ” the man paused to consider his next words, “it’s only a matter of time.” “Life’s only a matter of time when you get right down to it so what’s the difference?” Lucy sighed and ran a hand through her short grey hair. She had favoured an elfin cut since Charlie had died four summers past. She had worn it long to

please him for forty six years but without him to tell her how beautiful it looked the point was lost. “So…seeing as you’re so familiar with my name how about you tell me yours Mr…? It doesn’t look like a train’s coming any time soon so you’ve got time enough to make one up.” This drew a warmer chuckle from the man. “You’ve always been feisty I’ll give you that. So let’s see…my name, my name, my name…” Lucy could hear the man drumming his fingers lightly against his cheek while he paused. “I’m not much for giving out names but seeing as you asked so nicely…I’m just plain old Harry.” “Plain old Harry, ” said Lucy with a hint of a smile. “Well, just plain old Harry would you mind awfully telling me what it is that you want so that I can get on with…” The sudden realisation that she had no idea what she had been doing before the fogging breath and the conversation set Lucy’s heart to an irregular thump against the cage of her ribs and her words fell away. “Don’t worry Lucy, ” the man’s voice held some sympathy. “It’s always strange at the start but you’ll settle in to it soon.” “Settle in to what exactly?” The man laughed but the hint of sympathy was still apparent. “You really don’t have any idea what’s happening, do you?” He paused for a moment sensing Lucy’s unseen and soundless shake of the head before adding, “At least tell me you’ve bought your ticket already?” “Ticket?” said Lucy. “A ticket to where exactly? I wasn’t planning on making a trip I’m sure and it’s a bit difficult to decide on a to when you don’t currently know the where part.” Her voice rose in both pitch and pace as her heart thudded against its bony prison. “And how am I supposed to pay for a ticket? I’m in my best green dress! I don’t have room in it for pockets or purses and my handbag is on the table in the kitchen where I left it after…well, wherever it was I went before I came here and…now what was I saying before I lost my train of thought…train! Ha! That’s funny with me being in the station and all. Well alright, I’ll play along. Where do I get a ticket then plain old Harry?” The stranger’s name reverberated through the tunnels on either side of where she sat. Each returning echo sounded closer to hurry than the one preceding it and Lucy felt a rising sense of urgency. She smoothed down her dress and rose to her feet. “Harry?” The bench behind her was empty save for a large golden coin. She walked around the wooden perimeter and retrieved it with a growing sense of unease. The coin was perfectly smooth on both sides, heavy and cold to the touch. For a brief moment Lucy was convinced it was a foil covered chocolate left over from some forgotten Christmas and she let out a sharp, too-loud giggle that did nothing to calm her nerves. Facing the stairs she saw a worn sign for tickets pointing to the left of the staircase and as her eyes followed the sign’s instruction she saw a flickering bulb at the back of the platform highlighting a door in ragged white bursts. “Guess I really am buying a ticket then, ” she muttered. The door was as worn as the sign. Below a small, square panel of opaque glass stood a once proud set of brass letters reduced now to a sorry looking ‘T KETS’. Lower still someone had knifed the words ‘dead reckoning’. Fighting down her rising fears Lucy gripped the door handle and stepped in to the room beyond. The space that greeted Lucy beyond the door was far bigger than it looked from the outside. More corridor than room, it was at first glance as visually unappealing as the platform she had left behind. She was aware of a sharp click as the door closed behind her and then a voice rang out. “Breaths taken two point seven billion six million four hundred and eighty three thousand and nine. Breaths released two point seven billion four million nine hundred and sixteen thousand three hundred and twenty two. You’d think it would be the same but it never is for some reason.” “Hello?” “Time spent on trains, twenty nine days four hours seven minutes twelve seconds. A bit below average but perfectly acceptable.” The emptiness of the room caused the droning, metallic voice to bounce around making it difficult for Lucy to pinpoint the source. The space before her was high-ceilinged, surgically lit with dazzling, fluorescent strips and improbably long. On either side of her were blank white walls that stretched forward for a hundred paces at least. At the far end of the room was another white wall empty save for the word tickets in a large childlike scrawl. “Total distance travelled on foot one hundred and twenty one thousand miles at an average moving pace of three point two seven miles per hour.” “What has this got to do with anything? What do you want?” Lucy waited for an answer but the voice continued its statistical monologue. “Pairs of shoes worn, three hundred and sixty eight at an adjusted relative cost of eighty three thousand…” “I’m leaving, this is ridiculous, ” said Lucy as the voice continued its even paced hum. As she turned back to the door she let out a moan and a fluttering surge of adrenalin coursed through her body. In place of the door was a bare white wall. “Doors opened seven hundred and ninety eight thou…” “Shut up! Shut up! What’s this all about?” Lucy could feel that she was close to tears and battled to keep them in check. The coin in her palm grew heavy and uncomfortable. “…closed nine hundred and sixty three thousand seven hundred…” “So what? What difference does it fucking make?” Lucy gasped as the unbidden expletive escaped her lips. “Fucks

uttered, sixteen…tickets purchased nine hundred and thir…" Ticket. I need to buy a ticket. The thought galvanised Lucy and she started to walk towards the far end of the room, hesitant at first but with increasing purpose. "…books read three hundred and twenty six…" "I never was much of a reader, " Lucy muttered to herself as she continued to walk forward. Charlie was the one for reading. "Sexual partners, four…time spent kissing three hundred and fourteen hours exactly…" Lucy picked up her pace. "Opportunities lost…six hundred and seven…hearts broken none…times cheated on by Charlie twenty seven…" Lucy let out a sob. "LIAR! Charlie was a good man. He was good to me. He was always good to me." She started to run. Sensing her urgency the voice quickened and lifted in tone. "Tears shed seven hundred and sixty eight thousand and nineteen…significant lies told seven…minutes spent laughing twenty nine and a half thousand…" She was almost at the wall and the voice was now deafening, more akin to a physical assault than a noise and it dropped Lucy to her knees. "Time wasted waiting for CharliegoodoldCharliehewasalwaysgoodtomeCharlie to come home five hundred and nine hours twelve mi…" "STOP! Please just make it stop." Lucy crawled the last six feet and used the wall to gain her feet. At about chest height was a dark recess in the shape of a downturned mouth. "Time wasted time wasted time wasted TIMEWASTEDTIMEWAST" Lucy fumbled the coin out of her trembling hand and thrust it in to the hole. By sheer instinct she jerked her fingers back out before the metal shutter slammed closed and the voice cut off. "Have a pleasant day further!" said a cheerful female voice. Nondescript piped music began to play. The metal grid slid upwards to reveal a white paper ticket. Lucy steadied herself and removed it from the alcove. It was unmarked except for the words "Valid – Serial Number: r1-V3r5T-yX" in small red letters at the bottom. Holding it between thumb and forefinger Lucy turned around to look for a way out. The door she had entered by was less than ten feet away. She closed her eyes and took a deep breath but when she opened her eyes again the room was just as small and the door just as solid. Confused and with tears still drying on her cheeks Lucy forced herself forward and stepped back out on to the platform. The wooden bench she had sat on earlier was still unoccupied. Lucy walked towards it feeling more than her seventy four years. She sat down and put her head in her hands with a deep sigh. Feeling a warm breeze in her hair she sat upright again. She stretched her arms to out to each side and could feel the same faint zephyr against both her hands. At the edge of her still-blunted hearing she could make out a faint, screeching whine. "Looks like the trains are finally coming." "You again?" said Lucy, "Why am I not surprised?" The man gave another of his short laughs. "You've figured it out, that's why. Sharp girl like you I'm surprised it took you so long." The breath of wind from the tunnel increased and the metallic shrieking sounded much closer. Lucy dropped her arms to her sides but said nothing. "We can talk face to face if you like, " said the man, "Maybe you've got some questions…?" "Not really, " said Lucy with a touch of sadness. "It was only a matter of time." "And there's not much of that left." The man had to shout over the rising howl of the approaching trains. Lucy sensed her unseen companion getting to his feet behind her but she stared straight ahead and ignored the urge to turn around. At the far end of the platform she could see two white spots of light coming out of the shadows and the wind increased another notch. The eyes in the dark grew as did the sound of screaming metal and within moments the heads of two black, windowless trains hurtled towards her on either side before grinding to an eventual halt. There was a brief silence and then two puffs of air as the doors level with her on each side slid open in unison. Both carriages were glaringly lit rendering the contents indistinct to Lucy's squinting gaze. Behind her she heard soft, steady footsteps. "Have you made your choice?" "Yes." Lucy turned her head to stare along the platform once again. "Then perhaps you should choose a carriage…?" "No. I think I'll just sit here for a while"}],
  language: "en"
}