UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCELO ZIGLIOLI

ANÁLISE DE SENTIMENTO UTILIZANDO FERRAMENTAS OPEN SOURCE

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

MARCELO ZIGLIOLI

ANÁLISE DE SENTIMENTO UTILIZANDO FERRAMENTAS OPEN SOURCE

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof.ª Ms. Alessandra Bortoletto Garbelotti Hoffmann

MEDIANEIRA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

ANÁLISE DE SENTIMENTO UTILIZANDO FERRAMENTAS OPEN SOURCE

Por

Marcelo Ziglioli

Este Trabalho de Diplomação (TD) foi apresentado às 13:50 h do dia 20 de novembro de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Manutenção Industrial, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof.ª Me. Alessandra Bortoletto Garbelotti Hoffmann
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Marcio Matté
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Arnaldo Cândido Junior
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Me. Jorge Aikes Junior
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

ZIGLIOLI, Marcelo. **Análise de sentimento utilizando ferramentas open source**. 2015. Trabalho de Conclusão de Curso de Tecnologia em Análise e Desenvolvimentos de Sistemas - Universidade Tecnológica Federal do Paraná (UTFPR).

RESUMO

As empresas têm interesse na opinião das pessoas sobre seus produtos ou serviços, e até mesmo da sua imagem. Por meio da internet é possível obter informações que servem como base para esse tipo de análise e identificando a opinião comum de um grupo de usuários a respeito de um assunto, as quais podem ser utilizadas para auxílio na tomada de decisões estratégicas. Entretanto, os dados amplamente disponíveis na rede estão em formatos textuais não estruturados e inadequados para uso direto nesse processo. Nesse contexto, o objetivo deste trabalho é apresentar e aplicar as técnicas para o pré-processamento de textos provenientes de redes sociais com intuito de construir um conjunto de dados adequado para o uso com algoritmos de aprendizado de máquina e extração de padrões. Como resultado, foi desenvolvida uma ferramenta computacional para a realização dessa tarefa.
Palavras-chave: preparação de dados; mineração em redes sociais.

ZIGLIOLI, Marcelo. **Sentiment analysis using open source tools**. 2015. Monograph for Degree in Technology Systems Analysis and Developments. - Federal University of Technology – Paraná (UTFPR).

RESUMO EM LINGUA ESTRANGEIRA

Companies are frequently worried about customer's opinion on their products and offered services, on public interests or on what these people are publishing on social networks. Using the Internet it is possible to get many unstructured data carrying latent information that may give insight on public opinion about a specific subject. These insights, in turn, could be used as base information for the strategic decision-making process. However, the highly available data over Internet is usually unstructured textual information, which is inadequate for the process of pattern extraction and knowledge discovery. In this context, the aim of this paper is to present and apply several text pre-processing techniques on data gathered from social networks and convert them into an adequate input data set for machine learning algorithms. As a result, we have developed a computational tool to ease this entire process.
Key-words: data preparation; mining social networks.

LISTAS DE FIGURAS

LISTA DE SIGLAS

AC	Aquisição do Conhecimento
API	*Application Programming Interface*
ARFF	*Attribute-Relation File Format*
AM	Aprendizado de Máquina
HTTP	*Hypertext Transfer Protocol*
JDBC	*Java Database Connection*
JMS	*Java Message Services*
JPA	*Java Persistence API*
JSF	*Java Server Faces*
JSON	*JavaScript Object Notation*
MD5	*Message Digest 5*
MVC	*Model View Controller*
POA	Programação Orientada a Aspectos
SBC	Sistema Baseado em Conhecimento
SQL	*Structured Query Language*
SUN	*Sun Microsystems*
XML	*Extensible Markup Language*

LISTAS DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A análise de sentimento é utilizada para identificar as opiniões e ou sentimentos dos usuários de um determinado assunto (como lugar, pessoa, produto específico) em um determinado conjunto de dados.

A cada dia mais pessoas estão divulgando seus pensamentos e opiniões em redes sociais, como o *Twitter*, *Facebook* e outros. Esses pensamentos positivos, negativos ou neutros retratam as vidas de milhões de usuários que os compartilham em diferentes aspectos todos os dias. Esses dados podem ser utilizados por empresas, governo e indústrias para direcionar ações visando maior lucratividade ou atendimento às expectativas dos usuários. Uma indústria poderia observar a aceitação de um novo produto ou ainda o governo poderia observar qual o nível de satisfação destes em relação a sua gestão. Os assuntos e possibilidades de pesquisa são imensas levando-se em consideração a quantidade de dados que são lançados nas redes sociais diariamente. Porém, a análise humana e coleta dos dados significativos à pesquisa seria improvável sem a utilização de meios computacionais. Utilizando uma ferramenta computacional pode-se aplicar linguagens de programação e inteligência artificial para percorrer toda esta informação com intuito de coletar e estudar somente as informações pertinentes. A análise de sentimento proposta neste trabalho prevê a identificação e tratamento das expressões naturais do ser humano que podem ser expostas de maneiras diferentes, tais como a escrita de texto.

1.1 OBJETIVO GERAL

Desenvolver uma solução computacional para analisar dados provenientes de redes sociais e obter a análise de sentimento desses dados.

1.2 OBJETIVOS ESPECÍFICOS

- Obter os dados através de Api's especificamente desenvolvidas para acesso às redes sociais via Web;
- Aplicar técnicas de Pré-processamento de dados para filtrar e organizar os dados obtidos;
- Construir modelos de classificação a partir dos dados coletados na etapa anterior;
- Identificar e entender padrões relacionados aos rótulos de modelos que são utilizados pelos algoritmos;
- Obter a análise sentimental dos dados para um determinado assunto;
- Desenvolver uma interface Web para a utilização do projeto.

1.3 JUSTIFICATIVA

Cada vez mais as empresas estão buscando saber o que as pessoas estão falando a respeito de um assunto que estas pessoas estão interessadas em saber, adquirir ou somente expressando suas opiniões. Através da internet é possível buscar informações em *blogs*, redes sociais e outros meios que possam, de alguma maneira, serem utilizadas por alguém interessado no que as pessoas estão discutindo via *Web*.

As utilizações de dados comparativos podem ser utilizadas pelo comércio, indústrias e ainda pelo governo. Com base nesta informação seria de grande importância para o governo saber o que os milhões de usuários de redes sociais estão pensando a respeito de um fato que aconteceu, está acontecendo ou irá acontecer, ou, até mesmo quais são as reações demonstrados pelos usuários de algum *site* de compra quando lhes são mostradas ofertas relativas as suas características de consumidor. Com base nos resultados obtidos por um sistema de análise de sentimento, os interessados (empresas, indústrias e governo, por exemplo) podem direcionar ações que agreguem valor às suas atividades fins.

1.4 ESTRUTURA DO TRABALHO

Desenvolveu-se esse projeto seguindo algumas etapas bem definidas. São elas:

1. Etapa de estudo e levantamento bibliográfico
    - Entender os algoritmos de aprendizado de máquina utilizados para análise de sentimento;
    - Estudar as bibliotecas *open source* disponíveis para análise de sentimento;
    - Estudar bancos de dados *open source* não relacionais e como podem ser utilizados no contexto de armazenamento de dados de redes sociais;
    - Estudar *Api's open source* que serão empregadas na coleta de dados a partir das redes sociais;
    - Estudar e implementar uma *Api* própria para coleta de dados em *WebServices*.
2. Modelagem do Sistema
    - Elaborar diagramas necessários para o desenvolvimento do protótipo.
3. Desenvolvimento
    - Implementar uma solução.
4. Avaliação e Testes
    - Fazer a utilização do sistema com dados não reais para verificar possíveis erros;
    - Analisar resultados obtidos pelo sistema e comparar seu desempenho com as principais medidas de avaliação.
5. Publicação dos Resultados
    - Resultados obtidos serão publicados na monografia de conclusão de curso e, possivelmente, em artigos para eventos de iniciação científica.

2 SISTEMAS INTELIGENTES

Este capítulo apresenta a fundamentação teórica, necessária para a compreensão do trabalho proposto. Nesta etapa são apresentadas as definições de um sistema inteligente, tanto quanto como este sistema adquire conhecimento, e ainda quais são as técnicas utilizadas para o aprendizado.

2.1 SISTEMAS BASEADOS EM CONHECIMENTO

Os Sistemas Baseados em Conhecimento (SBC) são programas computacionais que usam o conhecimento representado explicitamente para resolver problemas, manipulam informações e conhecimento de forma inteligente, podem resolver problemas antes resolvidos somente por seres humanos (REZENDE, 2003).

Segundo Rezende (2003) a vantagem dos computadores em relação aos seres humanos é devido a sua velocidade, quantidade de processamento e consistência com que executam determinadas funções. São compostos por uma base de conhecimento, mecanismos de inferência e interface com o mundo. Na construção de um SBC, o conhecimento dos membros do projeto necessita ser capturado, organizado e disponibilizado na Base de Conhecimento. Uma vez este conhecimento armazenado, ele torna-se acessível e pode ser utilizado por alguma aplicação. Pode-se observar o processo de desenvolvimento de um SBC na Figura 1, onde é possível observar o fluxo do processo, partindo do Planejamento, Aquisição do Conhecimento (AC), Implementação até chegar na Verificação e Refinamento. Este processo possui um formato de ciclo aonde poderá retornar para o início conforme necessitado.

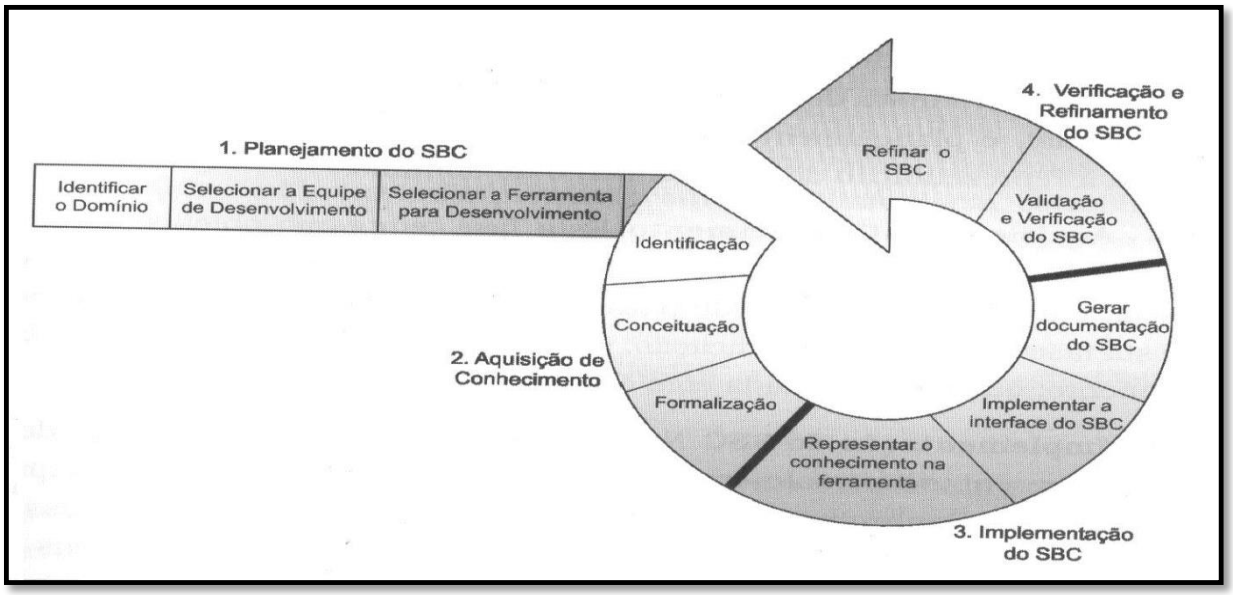


Figura 1 - Um Processo de desenvolvimento de um Sistema Baseado em Conhecimento.
Fonte: (REZENDE, 2003) .

2.2 AQUISIÇÃO DE CONHECIMENTO

Segundo Rezende (2003) o processo de AC é conhecido como o gargalo na construção de um SBC. É considerado um processo de interação entre humanos, conhecido como Engenheiro de Conhecimento, responsável pela construção do SBC e a fonte humana de conhecimento, o especialista no assunto. Estes processos podem ser classificados como manuais, semiautomáticas e automáticas.

- Manuais: são atribuídas pelo Engenheiro do Conhecimento;
- Semiautomáticas: fornece aos especialistas as ferramentas necessárias que possibilitam a criação dos sistemas minimizando a necessidade de um Engenheiro de Conhecimento;
- Automáticas: minimizam a participação humana, utilizam técnicas de mineração de dados ou mecanismos de inferência que permitem o aprendizado de máquina automático.

Pode-se observar as fases do processo de Aquisição do Conhecimento na Figura 2, divide-se em cinco fases, identificação, conceituação, formalização, implementação e testes. Este possui um processo interativo e evolucionário aonde pode-se retornar ou avançar para uma nova fase sem dependência de fases.

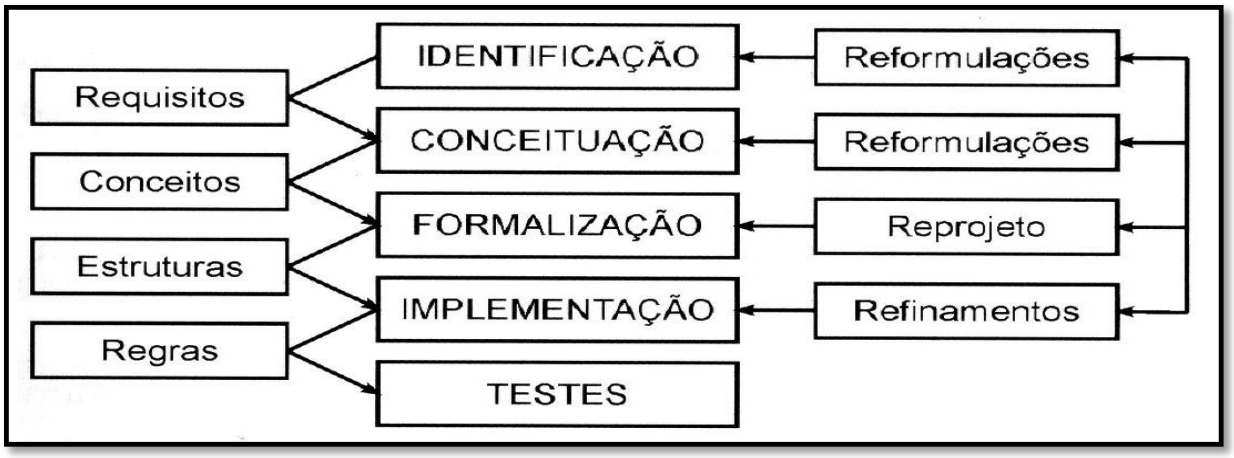


Figura 2 - Fases do processo de Aquisição de Conhecimento.
Fonte: (REZENDE, 2003) .

2.3 APRENDIZADO DE MÁQUINA

Com a crescente complexidade dos problemas a serem tratados computacionalmente e do grande volume de dados, tornou-se necessário o desenvolvimento de ferramentas computacionais mais sofisticadas e que fossem mais autônomas, reduzindo a intervenção humana e dependência de especialista no assunto. Para isso essas técnicas deveriam criar automaticamente, a partir de experiências adquiridas, uma hipótese capaz de resolver o problema. Esse processo de indução de uma hipótese a partir da experiência passada dá-se o nome de Aprendizado de Máquina (AM) (FACELI *et al.*, 2011).

Em AM, através do princípio da indução, no qual se obtêm conclusões genéricas a partir de um conjunto particular de exemplos, os computadores são programados para aprender com a experiência passada. São capazes de resolver problemas a partir de um conjunto de dados que representam instâncias do problema a ser resolvido (FACELI *et al.*, 2011).

2.3.1 Indução de Hipóteses

O objetivo da indução de hipóteses é aprender a partir de um subconjunto dos dados, denominado conjunto de treinamento, um modelo ou hipótese capaz de relacionar os valores

dos atributos de entrada de um objeto do conjunto de treinamento ao valor de seu atributo de saída. Tem a finalidade de induzir hipóteses de classificação para novos dados a partir dos dados utilizados para montar a regra de decisão. Uma vez induzida uma hipótese, é desejável a sua capacidade de generalização para que seja também possível utilizá-la em outros objetos do mesmo domínio (FACELI *et al.*, 2011).

2.3.2 Tarefas de Aprendizado

As tarefas de aprendizado podem ser divididas em Preditivas e Descritivas.

O objetivo das tarefas de previsão (Preditivas) é encontrar o modelo ou hipótese a partir dos dados de treinamento que possa ser utilizado para prever um rótulo ou valor que caracterize um novo exemplo, com base nos valores de seus atributos de entrada, devendo possuir, atributos de entrada e saída para cada objeto do conjunto de treinamento. Os algoritmos de tarefas preditivas seguem o paradigma de aprendizado supervisionado, aonde é necessário a presença do engenheiro do conhecimento para analisar o rótulo de cada exemplo.

Em tarefas de descrição (Descritivas) é explorar ou descrever um conjunto de dados, esses algoritmos não utilizam atributos de saída e seguem o paradigma de aprendizado não supervisionado, aonde a sua tarefa é encontrar regras de associação que possam relacionar de alguma maneira os grupos de atributos (REZENDE, 2003). Pode-se observar a hierarquia de aprendizado de acordo com os tipos de tarefas de aprendizado na Figura 3.

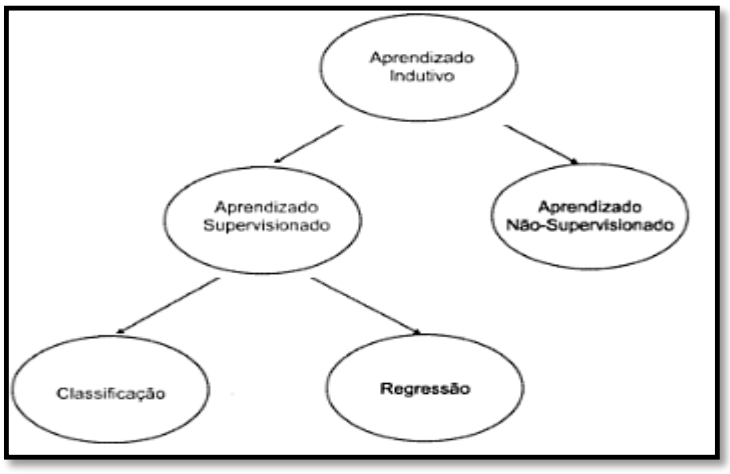


Figura 3 - Hierarquia de aprendizado.
Fonte: (REZENDE, 2003) .

3 MINERAÇÃO DE DADOS

A mineração de dados consiste no resultado de algumas etapas aplicadas aos dados, tais como, coleta dos dados originais, pré-processamento, extração de padrões, pós-processamento e uso do conhecimento (REZENDE, 2003).

A etapa de coleta de dados refere-se à obtenção dos dados para utilização no restante do processo. Trata-se da captura dos dados em original formato, sem nenhum tipo de processamento e padronização de formatos (REZENDE, 2003).

O pré-processamento consiste na tarefa de adequar os dados para a análise. Diversas transformações podem ser executadas nesta etapa, entre elas: (a) limpeza para remoção ruídos ou informações errôneas; (b) remoção de *Stopwords*, o qual consiste em remover algumas palavras que não interferem significativamente no processo ou termos que não traduzem a essência dos textos; (c) *Stemming*, que consiste na identificação das diferentes inflexões referentes à mesma palavra e sua substituição por um radical comum; e (d) cortes de *Luhn*, que consistem em excluir os termos mais comuns e os considerados raros, com baixíssima frequência, pois esses não serão úteis para discriminar os padrões latentes nos dados (REZENDE, 2003).

A extração de padrões, terceira etapa do processo de mineração de textos, compreende à tarefa de construção dos modelos de aprendizado, na qual algoritmos são utilizados para analisar os conjuntos de dados e descobrir os padrões neles embutidos, possibilitando a descoberta de conhecimento (REZENDE, 2003).

O pós-processamento e avaliação do conhecimento consiste na validação e avaliação de qualidade do modelo construído, assim como na adaptação do conhecimento extraído de modo que seja possível utilizá-lo posteriormente. Observe que esse processo é realizado em ciclos, uma vez que se o conhecimento extraído for avaliado como insuficiente, o processo pode ser refeito a partir de uma etapa anterior com o intuito de melhorar os resultados. Esse processo todo pode ser aplicado com o objetivo específico de identificar as percepções de um grupo de pessoas com relação à um tema específico (REZENDE, 2003)

O uso do conhecimento é a última etapa da Mineração de Dados, aonde os dados em questão foram coletados, analisados, processados e avaliados, poderão ser utilizados como fonte de conhecimento para avaliação de novos dados, se este processo obter um resultado satisfatório (REZENDE, 2003).

3.1 MINERAÇÃO DE TEXTOS

Mineração de textos é um conjunto de técnicas e processos para coleta de informações presentes em dados textuais expressos em linguagem natural (REZENDE, 2003). Esse processo é composto por etapas responsáveis pela execução de diferentes tarefas.

3.1.1 Técnicas de Mineração de Textos

Entre as diversas técnicas de mineração de texto, foram utilizadas neste trabalho as Limpeza, *Stopwords*, *Stemming* e cortes de *Luhn*.

Os algoritmos utilizados foram o *J48* e *Naive Bayes Multinomial* por serem amplamente utilizados na mineração de textos.

3.1.2 Limpeza

A limpeza consiste em retirar informações errôneas ou indesejáveis que possam interferir na interpretação do texto. Essas informações podem ser caracteres que poluem ou danificam o entendimento de um determinado texto, como por exemplo, nas redes sócias são comuns o uso de alguns caracteres que não possuem significado na língua portuguesa, entre eles, “@”, “#” , “http” entre outros. Pode-se observar a utilização desta técnica na Tabela 1.

Tabela 1 - Exemplo de utilização do método de limpeza.

TWITTER ORIGINAL	TWITTER APÓS LIMPEZA
RT @moraesluciano: #BetaLab Dilma se reúne com ministros para definir investimentos: http://t.co/Qd9r425n59	moraesluciano BetaLab Dilma se reúne com ministros para definir investimentos
RT @casa_dos_pobres: Dilma vira o jogo: http://t.co/vhwNTDzFze	casa_dos_pobres Dilma vira o jogo

3.1.3 *Stopwords*

O processo de remoção de *Stopwords*, é o processo aonde deverão ser removidas as palavras que não deverão influenciar no processamento dos dados. Estas palavras devem ser adicionadas pelo usuário no momento da criação do Modelo de Teste, assim o sistema irá rejeitar todas as palavras que estiverem na lista.

3.1.4 *Stemming*

Stemmer é parte de um processo composto de extração de palavras do texto e transformá-los em termos de índice em um sistema. Para cada palavra é criado uma equivalente base, por exemplo, a palavra provável e provavelmente, possuem o mesmo equivalente base "provável", este método de extração de uma palavra comum denomina-se Stemmer (PORTER, 2014). Pode-se observar mais exemplos de Stemmer na Tabela 2.

PTStemmer é um kit de ferramentas utilizado para o processo de extração de *Stemmers* no idioma Português, tendo como pressuposto o algoritmo *Snowball* (PORTER, 2014). *PTStemmer* pode ser utilizado em diversas linguagens, como Java e Python. Produzido e disponibilizado em código aberto por Pedro Oliveira.

Tabela 2 - Exemplo da utilização de Stemmer.

STEMMER	PALAVRAS
Lemb	lembrada, lembrança, lembrar
Invest	investimento, investir, investido
Temp	tempo, temporario, temporariamente

3.1.5 Cortes de *Luhn*

Consiste em excluir termos e palavras não relevantes, os termos que excedem o corte superior são considerados comuns e os abaixo do corte inferior são considerados raros, portanto, estes dados não contribuem significativamente na discriminação dos documentos.

3.1.6 *Naive Bayes*

Naive Bayes é um classificador probabilístico simples baseado na aplicação de teorema de Bayes. É um dos mais eficientes e eficazes algoritmos de aprendizagem indutivos para aprendizado de máquina e mineração de dados. É a forma mais simples de rede bayesiana, em que todos os atributos são considerados independentes do valor da variável de classe. Isto é chamado de independência condicional (ZHANG, 2004).

Multinominal Naive Bayes (MNB), é um método de classificação que representa as frequências com que certos eventos foram gerados por uma multinominal, normalmente utilizados para classificação de documentos, os valores de recursos são as frequências prazo, geradas por um multinominal que produz um certo número de palavras.

Pode-se observar a formula de classificação utilizada pelo MNB é dado por:

$$\Pr(c|ti) = \frac{\Pr(c)\,\Pr(ti|C)}{\Pr(\text{ti})}, c \in C \qquad (1)$$

Onde C é o conjunto de classes, Pr(c) é a classe de precedente, $Pr(t_i|c)$ é a probabilidade de obter um documento com t_i, atribui um documento ti teste para a classe que tem a maior probabilidade Pr $(c|t_i)$ e $Pr(t_i)$ é o fator de normalizando (KIBRIYA *et al*., 2008).

Pode-se observar a estrutura do algoritmo Naive Bayes na Figura 4.

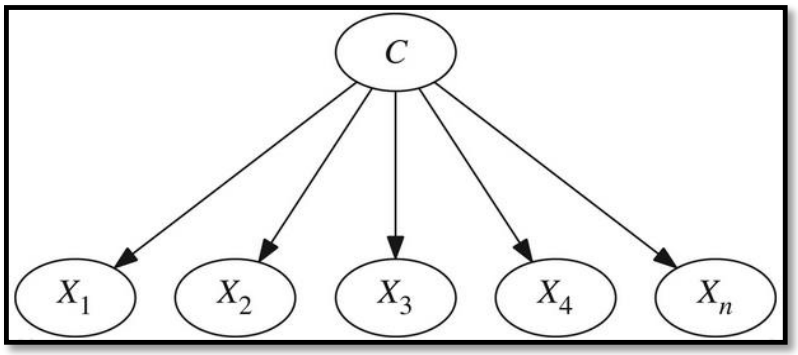


Figura 4 - Estrutura do algoritmo Naive Bayes.
Fonte: http://rspa.royalsocietypublishing.org/content/465/2109/2927.

3.1.7 Árvores de decisão J48

J48 é um algoritmo para classificação baseado em árvores de decisão, é uma forma de previsão com dois nós filhos conhecidos como, (1) decisão e (2) classe. Cada nó contém basicamente duas informações, (a) onde um índice de pureza passa a informação de que todos os casos pertencem à mesma classe, (b) é o tamanho total de dados em questão. O algoritmo J48 constrói uma árvore de decisão com base nos atributos obtidos a partir do conjunto de treinamento. O classificador extrai os atributos que podem classificar os casos, tais atributos são chamados de atributos discriminantes, um modelo de árvore de decisão é construído usando as instâncias de formação e seleção do atributo (VENUGOPAL e PATNAIK, 2011). Pode-se observar a estrutura do algoritmo J48 na Figura 5.

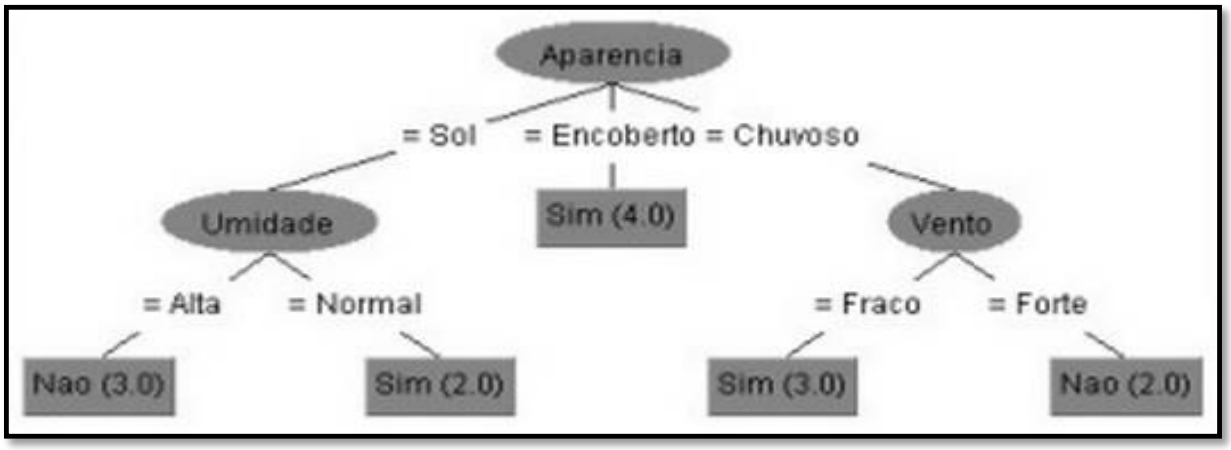


Figura 5 - Estrutura de árvore de decisão J48.
Fonte: http://pt.slideshare.net/iaudesc/algoritmoid3ec45gilcimar.

4 MATERIAS E MÉTODOS

Como ferramenta proposta, foram desenvolvidas duas aplicações, um serviço *Web Service* como servidor e um cliente *Web*.

Como servidor, foi desenvolvida uma aplicação baseada em serviços, retornando como resposta *Json Object*. Entretanto como cliente, fez-se um sistema *Web* para consumir os serviços disponíveis e utilizá-los na aplicação cliente.

Para a realização dos mesmos foram utilizadas várias tecnologias e *frameworks* que serão demonstradas a seguir.

4.1 FRAMEWORKS E TECNOLOGIAS

Este capitulo irá descrever todos os *frameworks* e tecnologias utilizadas na elaboração deste projeto.

Estas ferramentas são utilizadas para auxiliar na criação e desenvolvimento de projetos, possuem funcionalidades especificas e utilizam padrões na sua arquitetura.

4.1.1 Java

Java é uma plataforma de desenvolvimento criada pela SUN e teve seu lançamento em 1995, possui como fonte principal, a sua portabilidade entre os sistemas e mantendo uma linguagem de programação simplificada, segura, orientada a objetos e com extensa API. Java adotou a ideia de utilizar uma máquina virtual que executa um código máquina próprio, aonde esta máquina traduz as instruções para uma plataforma específica, tornando assim a plataforma independente (SILVEIRA *et al.*, 2012).

4.1.2 Api Public Streams

A API Public Streams possui um processo de streaming que recebe os *tweets* de entrada e executa qualquer análise, filtragem e / ou agregação necessário antes de armazenar o resultado em um arquivo de dados. O processo de manipulação HTTP consulta o armazenamento de dados para resultados em resposta às solicitações dos usuários, obtendo assim um fluxo em tempo real de dados *tweet* para fazer a integração com outros aplicativos (TWITTER DEVELOPERS, 2014). Pode-se observar o funcionamento desta API na Figura 6.

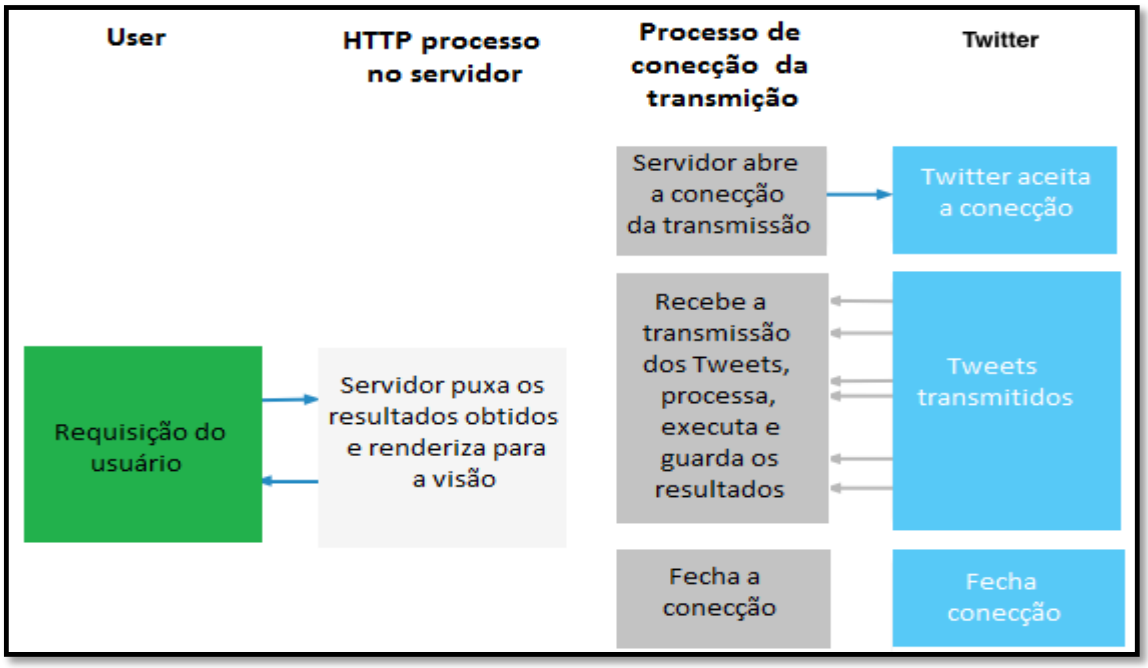


Figura 6 - Funcionamento da API Public Streams.
Fonte: https://dev.twitter.com/docs/api/streaming.

4.1.3 Http

O HTTP é o protocolo padrão para troca de informações via Internet e utiliza a porta 80 como padrão para a comunicação *Web*. Foi o caminho encontrado pelas organizações para uma nova geração de padrões de comunicação, onde a evolução das hipermídias e do HTPP permitiu que este superasse muitos outros protocolos já existentes (SILVEIRA *et al*., 2012).

4.1.4 *Json*

Json (*JavaScript Object Notation*) Notação de Objetos *JavaScript* é um formato leve de intercâmbio de dados, fácil tanto para seres humanos quanto para máquinas, entender e analisar. Ele é baseado em um subconjunto da linguagem de programação JavaScript, completamente independente do idioma. É construído em duas estruturas:

- Uma coleção de pares nome / valor: objeto, registro, *struct* entre outros;
- Uma lista ordenada de valores: *array*, vetor, lista ou sequência.

Estas são estruturas de dados universais, praticamente todas as linguagens de programação modernas utilizam estas estruturas, portanto este tipo de dado é considerável portável, sendo assim, pode ser utilizado em diferentes linguagens (JSON, 2014).

4.1.5 Hosebird Client (Hbc)

O Hbc é um pacote de bibliotecas que foram desenvolvidas para a plataforma Java com o intuito de um cliente HTTP consumir transmissões API do Twitter.

O cliente é dividido em dois módulos: HBC-core e HBC-Twitter4J. O módulo de HBC-núcleo que utiliza uma fila de mensagens, onde o consumidor pode pesquisar as mensagens, enquanto o módulo de HBc-twitter4j utiliza objetos do tipo Twitter4J para prover uma camada de análise para os dados (LIU e OLIVER, 2014).

4.1.6 ARFF

ARFF é um arquivo de texto ASCII que descreve uma lista de instâncias que compartilham um conjunto de atributos. Arquivos ARFF foram desenvolvidas pelo Projeto de Aprendizado de Máquina no Departamento de Ciência da Computação da Universidade de *Waikato* para uso com o software de aprendizado de máquina Weka (WAIKATO, 2014).

Pode-se observar a estrutura de um arquivo ARFF na Figura 7.

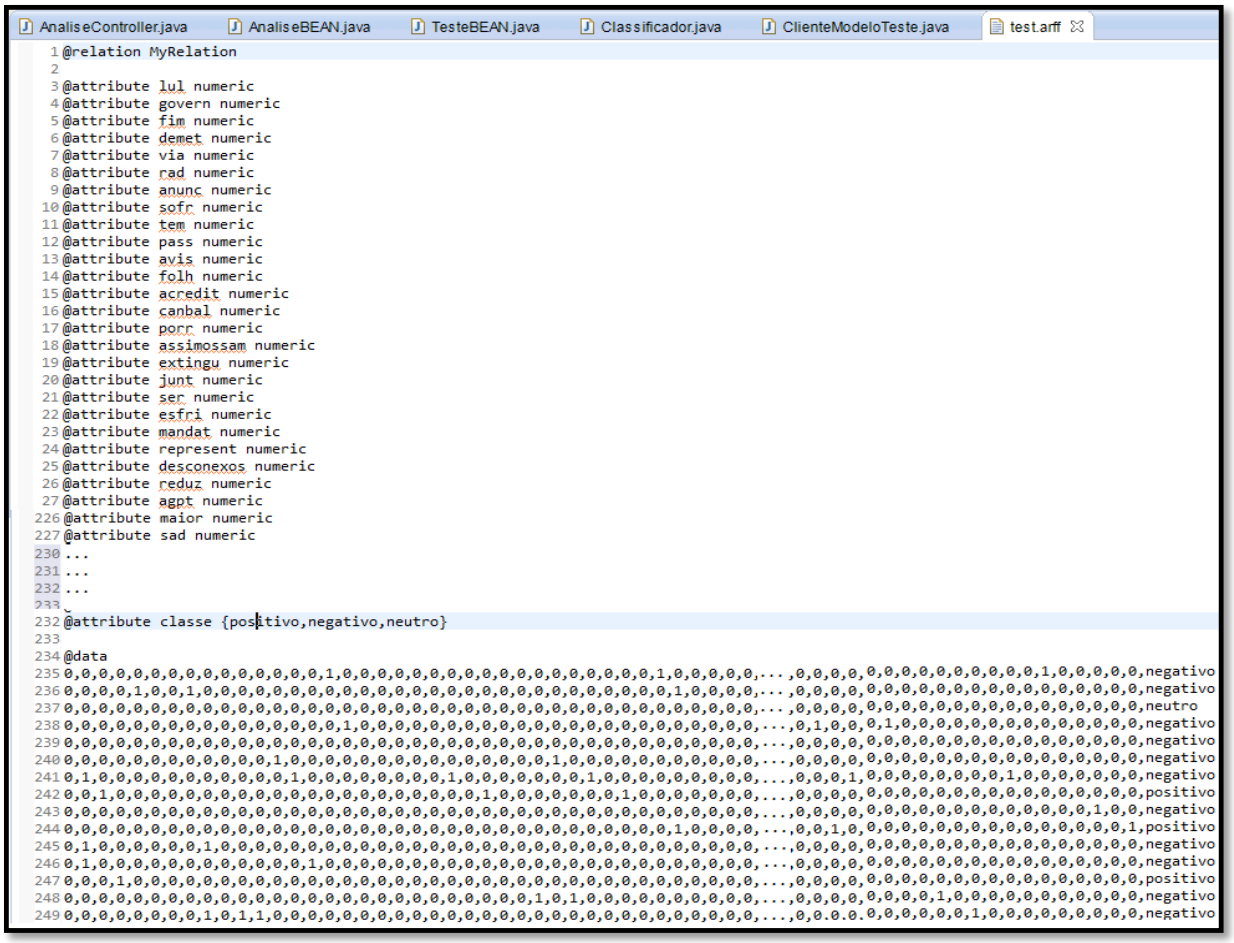


Figura 7 - Estrutura de um arquivo ARFF.
Fonte: Autoria Própria.

4.1.7 Weka

É uma coleção de algoritmos de aprendizado de máquina para tarefas de mineração de dados. Os algoritmos podem ser aplicados diretamente a um conjunto de dados ou chamado a partir de seu próprio código Java. Weka contém ferramentas para pré-processamento de dados, classificação, regressão, *clustering*, regras de associação e visualização. Ele também é bem adequado para o desenvolvimento de novos sistemas de aprendizado de máquina (WAIKATO, 2014).

4.1.8 MySQL

O MySQL é o mais popular SQL banco de dados de código aberto, foi desenvolvido e distribuído por MySQL AB. O Banco de dados é uma coleção de estruturas de dados, estes dados podem ser qualquer coisa, simples como uma lista de compras, ou um grande número de informações de uma grande corporação.

Para adicionar, acessar e processar dados guardados em um banco de dados, faz-se necessário a utilização de um *database management system*, ou seja, sistema de controle de dados, como o MySQL SERVER, que é um sistema de banco de dados relacional (WIDENIUS e AXMARK, 2002).

4.1.9 *Web Services*

Web Services é uma maneira de expor as funcionalidades de um sistema de informação e fazer com que as informações sejam disponíveis através do padrão das tecnologias *Web*. O uso desses padrões, faz com que diminua-se heterogeneidade da aplicação facilitando a integração entre aplicações. É uma aplicação baseada na URI, e possui o suporte de troca de mensagem em formatos XML, JSON, texto via protocolos da internet, que são eles GET, POST, PUT e DELETE (ALONSO *et al.*, 2004).

4.1.10 Spring Framework

O Spring Framework fornece um modelo abrangente de programação e configuração de aplicativos baseados em Java EE, consiste em uma ferramenta para auxiliar os desenvolvedores nas tarefas de baixo nível dos aplicativos corporativos para que as equipes possam concentrar na lógica de negócios em nível de aplicativo. Entre as suas principais características estão a Injeção de Dependência, POA, Spring MVC, JDBC, JPA, JMS entre outros (THE SPRING TEAM, 2015).

4.1.11 Spring Security

Spring Security é uma estrutura de autenticação e de controle de acesso baseado no Spring Framework. Concentrada em fornecer autenticação e autorização para aplicativos Java. Entre suas principais características estão o grande suporte em autenticação e autorização, proteção contra ataques como fixação de sessão, *clickjacking*, *cross site request forgery*, entre outros e ainda a Integração opcional com o Spring Web MVC (THE SPRING TEAM, 2015).

4.1.12 JavaMail API

A JavaMail é uma API responsável em adicionar facilmente a capacidade de enviar mensagens através de correio eletrônico em uma aplicação Java. Esta API possui classes que são projetadas para desenvolver funções e protocolos de correio eletrônico. Entre suas características estão a criação de classes, que podem interagir com outras API's Java, interfaces de manipulação de e-mail básicas para qualquer aplicação e ainda pode funcionar com uma variedade de formatos, tipos de dados e de acesso e transportes protocolos (ORACLE, 2015).

4.1.13 Hibernate

Hibernate ORM (*Object/Relational Mapping*) é uma ferramenta de persistência de dados, utilizado para persistir objetos em banco de dados de maneira automatizada e transparente. Também é uma implementação da especificação JPA, pode ser facilmente utilizado em qualquer ambiente de suporte JPA (JBOSS COMMUNITY, 2014).

4.1.14 Apache Maven

É uma ferramenta de gerenciamento de projetos de software e compreensão. Baseado no conceito de um modelo de objeto do projeto (POM), pode gerir a construção de um projeto, elaboração de relatórios e documentação a partir de uma peça central de informações. Modelo de objeto do projeto ou POM é a unidade fundamental para um projeto Maven, este arquivo possui o formato XML que contém informações sobre os detalhes do projeto e configurações utilizadas pela ferramenta para construir o projeto (THE APACHE SOFTWARE FOUNDATION, 2014).

4.1.15 JSF

O JSF é uma tecnologia que nos permite criar aplicações Java para Web utilizando componentes visuais pré-prontos, de forma que o desenvolvedor não se preocupe com *JavaScript* e HTML. Os componentes podem ser adicionados facilmente ao projeto e eles serão rende rizados e exibidos em formato *html*. O estado dos componentes é sempre guardado automaticamente (*Stateful*). Trabalha com o modelo MVC, separando as camadas de modelo, visão e controladores (ORACLE, 2014).

4.1.16 Primefaces

PrimeFaces é uma biblioteca de componentes para JSF, é considerado uma biblioteca leve, de fácil configuração e possui muitos componentes para auxiliar no desenvolvimento *Web*. O Primefaces é uma coleção de ricos *JavaScript* e *widgets* baseado em *jQuery* e HTML5 (PRIMEFACES, 2014).

4.1.17 Apache Tomcat

O Apache Tomcat é um contêiner Java e um servidor *web*, é uma implementação de software de fonte aberta das tecnologias Java *Servlet* e JSP. Utilizado para publicar e servir páginas web produzidas em linguagem Java (THE APACHE SOFTWARE FOUNDATION, 2014).

4.1.18 Metro UI CSS

Metro UI CSS é uma ferramenta desenvolvida com o conselho da Microsoft para construir uma interface para o usuário, estão contidos neste conjunto de CSS, estilos gerais, grade, *layouts*, tipografia, componentes e ícones. Possui a licença para uso livre e tem modelo de licenciamento MIT (PIMENOV, 2015).

4.1.19 Criptografia MD5

O MD5 é um algoritmo de *hash* de 128 bits unidirecional, descrito na RFC 1321. Este algoritmo recebe como input uma mensagem e produz como output uma mensagem de 128 bits, esta nova mensagem não poderá ser transformada novamente na mensagem anterior devido a este algoritmo ser unidirecional (MIT LABORATORY FOR COMPUTER SCIENCE AND RSA DATA SECURITY, 2015).

4.2 MODELAGEM

O Diagrama de Classes: é utilizado para demonstrar a modelagem das classes no mundo real e especificar as relações entre elas (GUINWA, 2004). Na Figura 8 pode-se observar o Diagrama de Classes da aplicação.

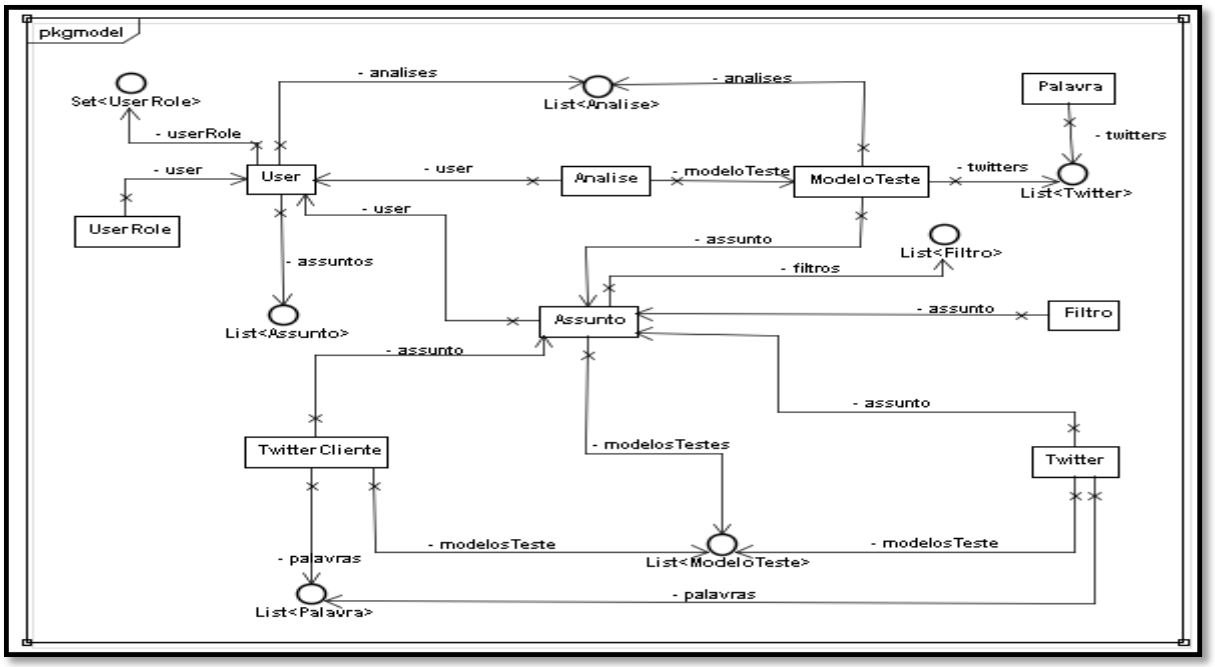


Figura 8 - Diagrama de Classes.
Fonte: Autoria Própria.

Diagrama de Caso de Uso: é uma representação gráfica das funcionalidades do sistema do ponto de vista do usuário (GUINWA, 2004). O sistema consiste no diagrama de caso de uso representado na Figura 9.

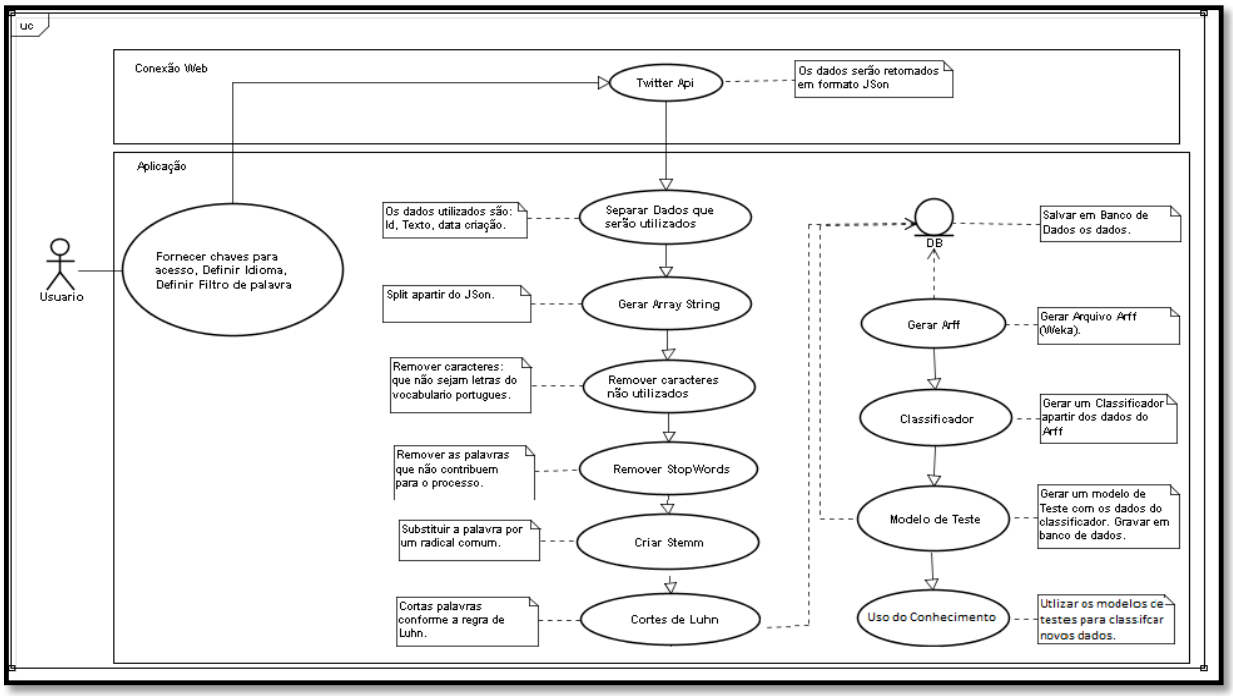


Figura 9 - Diagrama de Caso de Uso.
Fonte: Autoria Própria.

5 ANÁLISE DOS RESULTADO

O sistema proposto atendeu aos requisitos propostos pela monografia, sendo sua evolução e resultados descritos a seguir. O desenvolvimento do projeto deu-se, na construção de dois projetos distintos.

5.1 WEBSERVICE REST

O primeiro projeto foi o desenvolvimento de um *WebService* REST. Este sendo desenvolvido na linguagem Java e utiliza-se de vários frameworks descritos na etapa de materiais e métodos.

Entre os materiais utilizados estão o Spring Framework utilizado na construção dos serviços REST, sendo este publicado em um servidor Apache Tomcat, Spring Security utilizado para controlar as requisições aos serviços, Apache Maven para o gerenciamento de dependências, o Http como protocolo de transferência de informações entre cliente-servidor, e como banco de dados foi utilizado o MySQL.

Pode-se observar no Apêndice na Figura 22 o código de criação de serviços que gerencias as chamadas para as análises do sistema.

Como parte de segurança do servidor foi utilizado o Spring Security utilizando arquivos xml como arquivo de configuração, o arquivo referente pode ser verificado no Apêndice na Figura 23.

5.2 CLIENTE WEB

O segundo projeto desenvolvido foi um cliente *Web*, responsável em buscar e gerenciar as informações retornadas pelo servidor, através de requisições de Http. O cliente conta com toda a estrutura visual do sistema e ainda com parte de regras de negócio. Entre as tecnologias utilizadas estão JSF responsável pelo MVC do projeto, o Primefaces e Metro UI utilizados na parte gráfica, Weka e Arff para a criação de classificadores e modelos de teste, e

ainda API Public Streams e Hosebird Client para obter os dados a serem estudados, o MD5 foi utilizado para criptografar a senha do usuário para a transferência e armazenamento seguro no servidor e o JavaMail utilizado para enviar ao usuário do sistema um *link* para a troca de senha via e-mail.

A página inicial da aplicação é a "index.xhtml", aonde o usuário previamente cadastrado, deverá autenticar-se com e-mail e senha para entrar no sistema, esta pode ser visualizada na Figura 10 e o código referente no Apêndice na Figura 24. O usuário pode cadastrar-se através da página de cadastro que pode ser exibida ao clicar no botão cadastra-se. O usuário pode ainda solicitar uma a criação de uma nova senha através do botão recuperar senha via e-mail Figura 11.

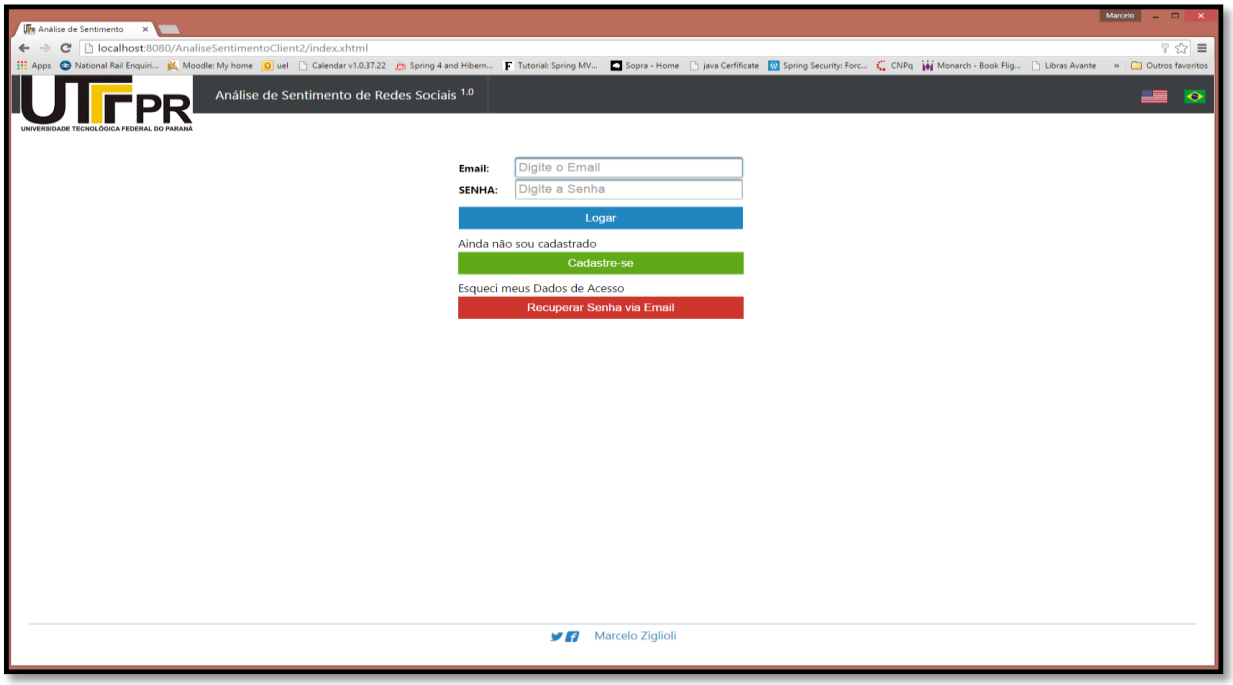


Figura 10- Página de *Login*. Opção de mudar o idioma.
Fonte: Autoria Própria.

As senhas dos usuários são criptografadas utilizando MD5, o que não possibilita a sua recuperação em caso de esquecer a senha. Para a troca de senha foi utilizado o sistema de criar uma nova senha e atualizar o registro do usuário com esta senha, a autenticação do usuário 'e feita através de um *token* de acesso, o qual 'e enviado ao e-mail do usuário, conforme Figura 12. O Código referente a este processo pode ser visualizado no Apêndice nas Figuras 25 e 26.

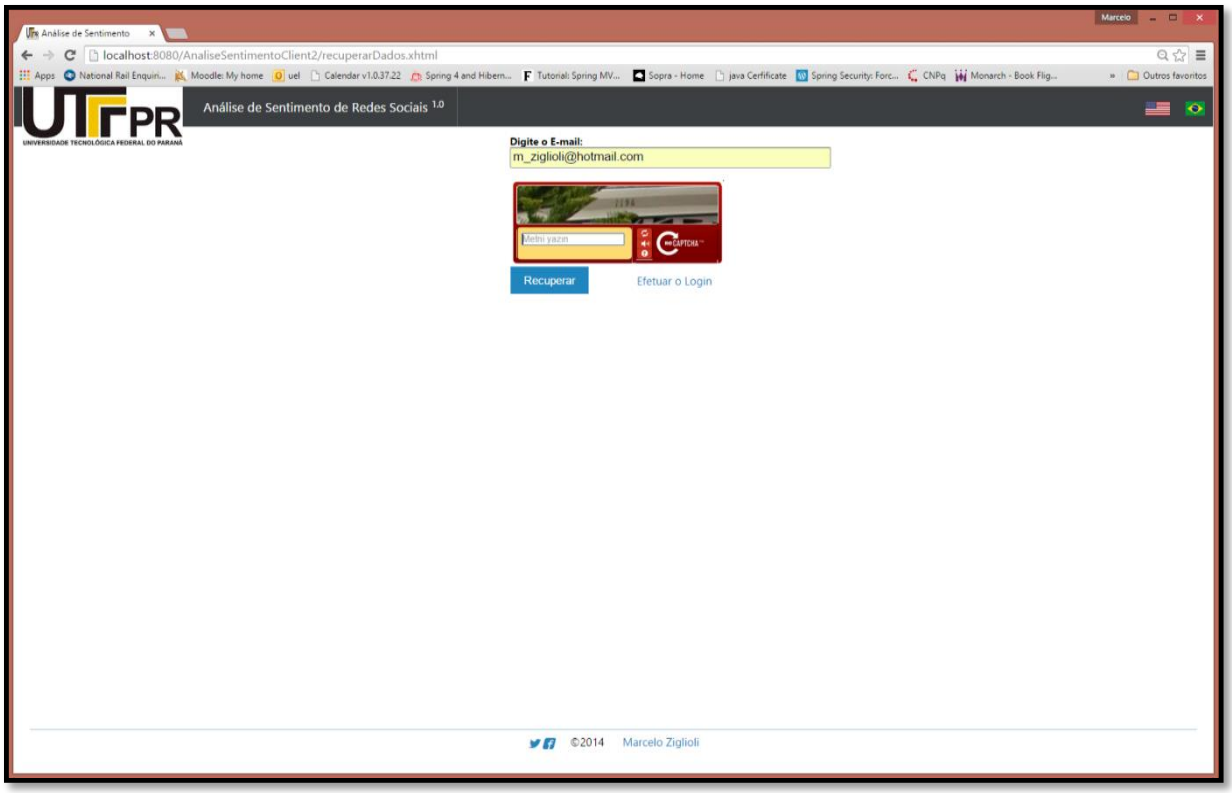


Figura 11 - Página Recuperar Dados.
Fonte: Autoria Própria.

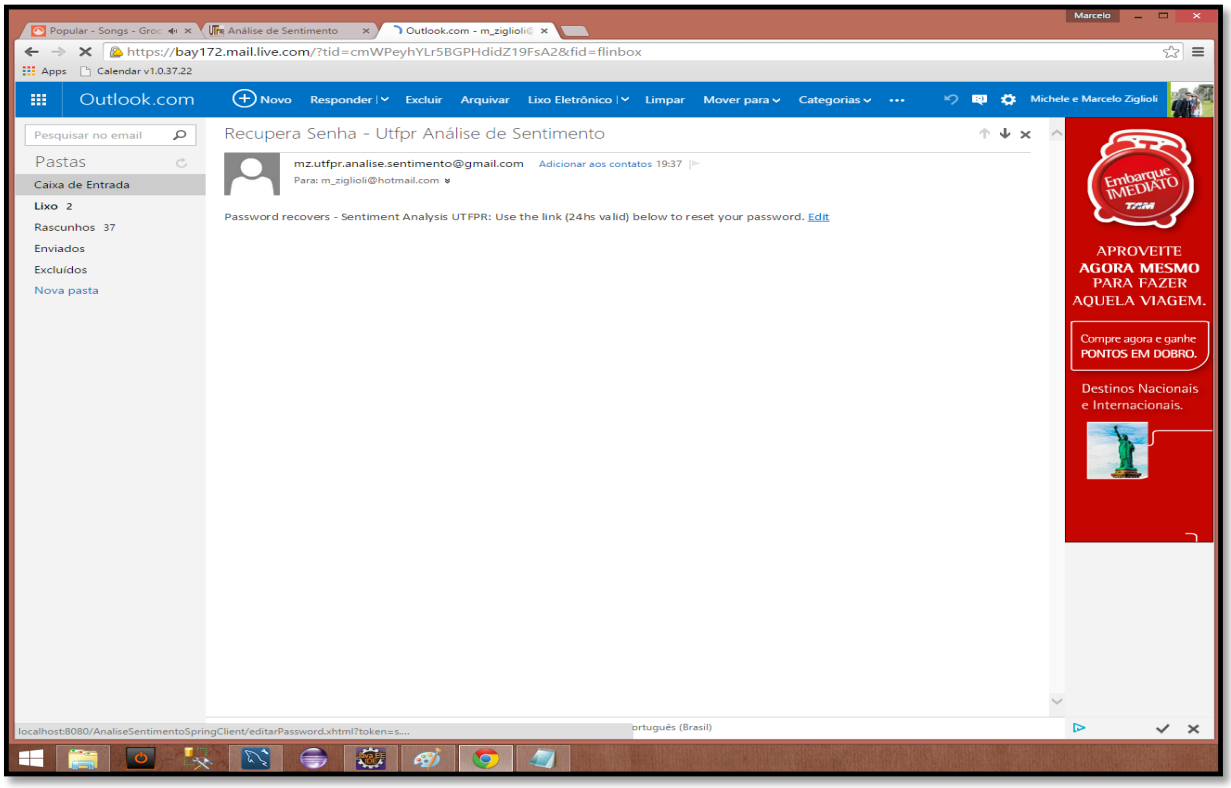


Figura 12 - Verificação do recebimento do e-mail, com o link para o click.
Fonte: Autoria Própria.

A página *home*, “home.xhtml”, é onde o usuário tem acesso ao *menu* com as opções disponibilizadas pela aplicação, conforme Figura 13.

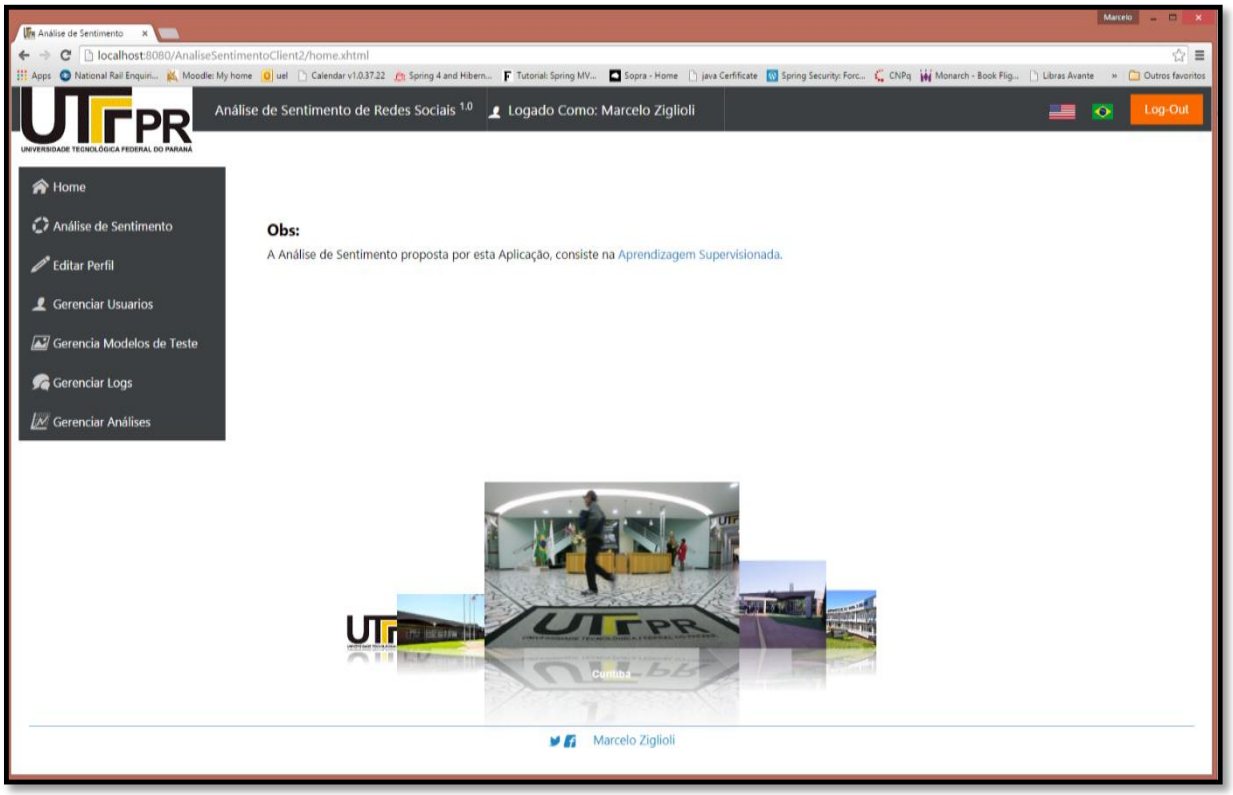


Figura 13 - Página Home.
Fonte: Autoria Própria.

A principal funcionalidade do sistema está na página “analiseSentimento.xhtml”, é onde o usuário poderá efetuar uma nova análise de dados dos assuntos “públicos” ou dos seus próprios assuntos, ou ainda criar um novo assunto para ser analisado. As opções de busca, são o banco de dados ou o Twitter API, conforme pode-se observar na Figura 14.

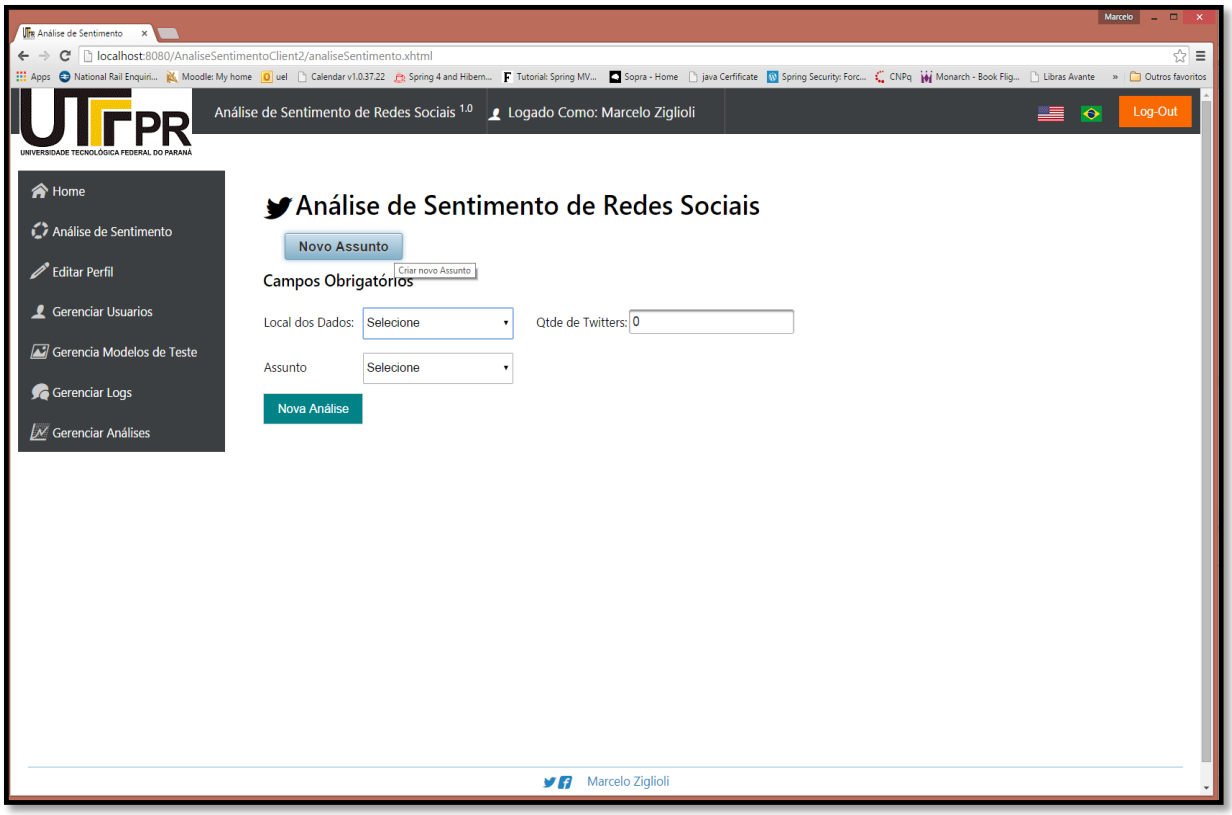


Figura 14 - Página Nova Análise de Sentimento.
Fonte: Autoria Própria.

Para criar um novo assunto, o usuário deverá fornecer alguns dados conforme a Figura 15. Ao clicar no botão criar novo assunto o sistema dá início ao processo de maior custo computacional do sistema, promovendo o download e processamento dos dados recebidos.

A Figura 16 mostra como o sistema de forma visual possibilita ao usuário acompanhar as etapas do sistema. Através da Figura 27 no Apêndice é possível visualizar o código de busca dos *tweets* e atualização da barra de progresso e ainda para cada *tweet* baixado dar-se-á a etapa de pré-processamento no Apêndice (Figura 28, 29 e 30) e posteriormente a persistência dos objetos necessários.

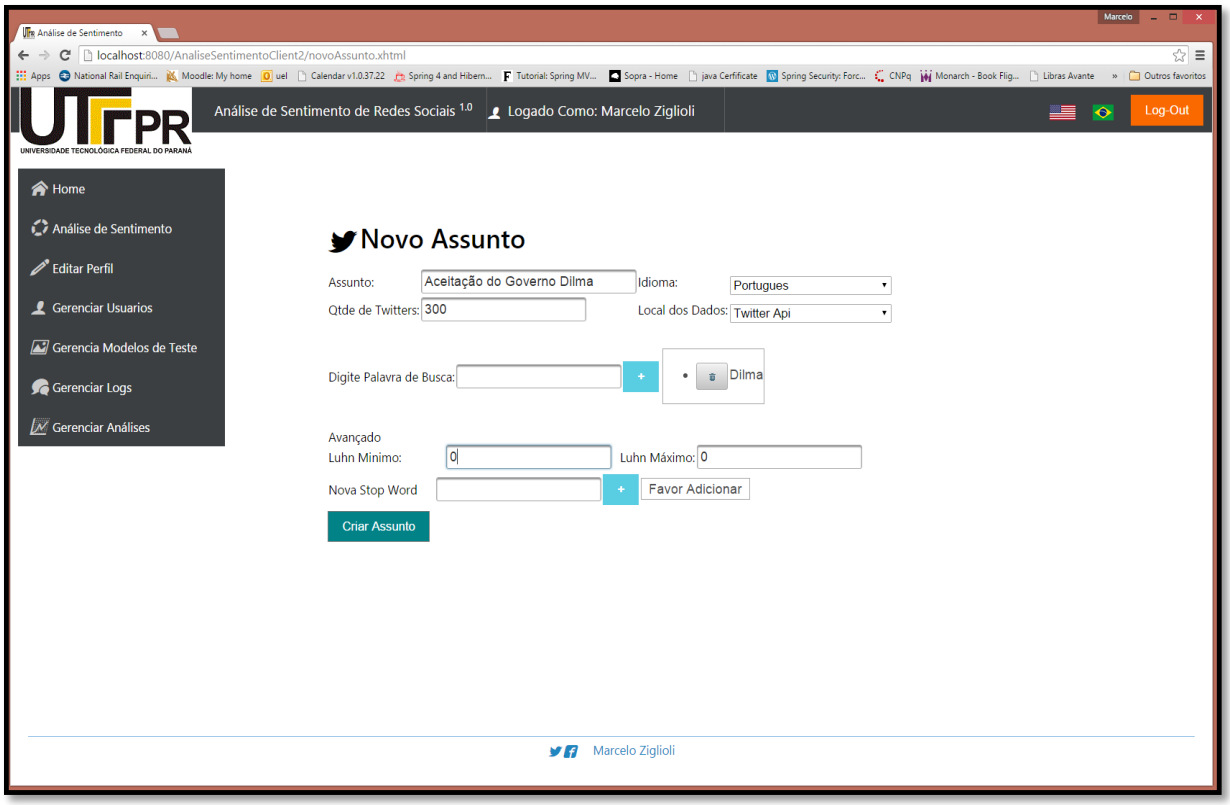


Figura 15 - Página Criar Novo Assunto.
Fonte: Autoria Própria.

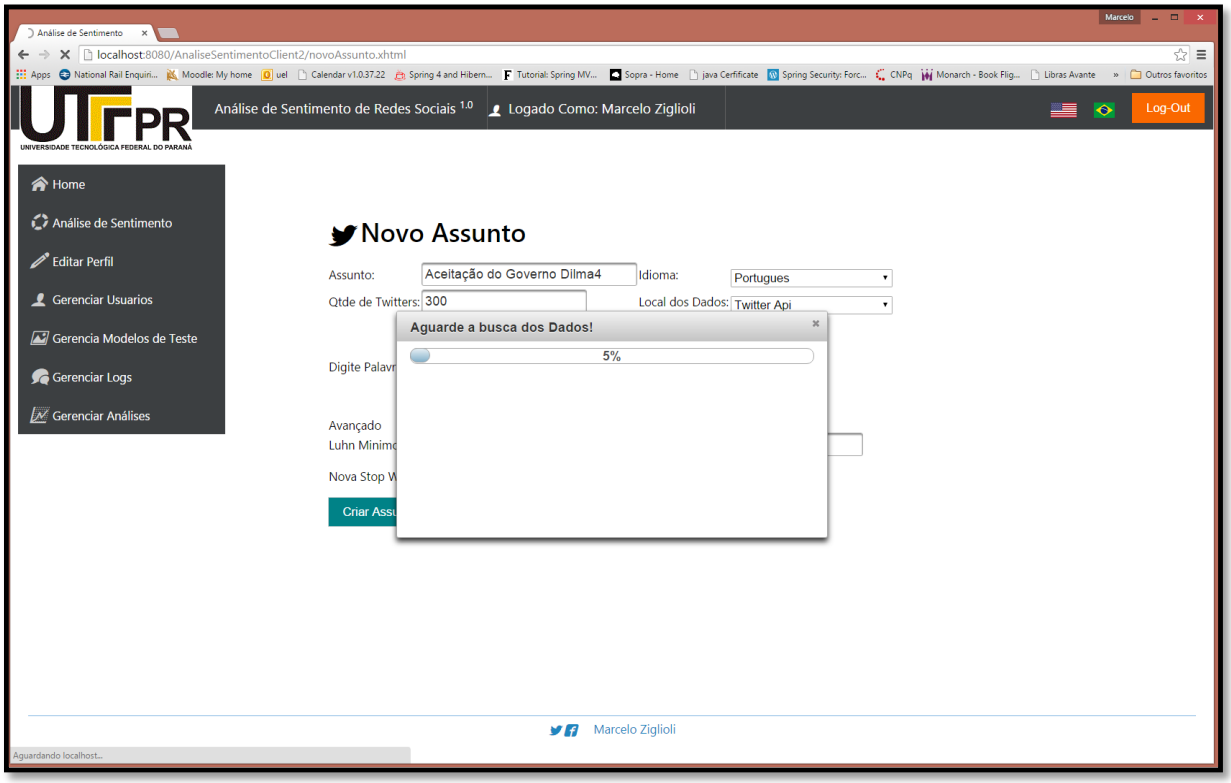


Figura 16 - Página Criar Novo Assunto. Baixando os Dados.
Fonte: Autoria Própria.

Após o término do processo será aberto em forma de tabela, os *tweets* obtidos pelo sistema. Estes *tweets* deverão ser manualmente classificados, conforme a Figura 17.

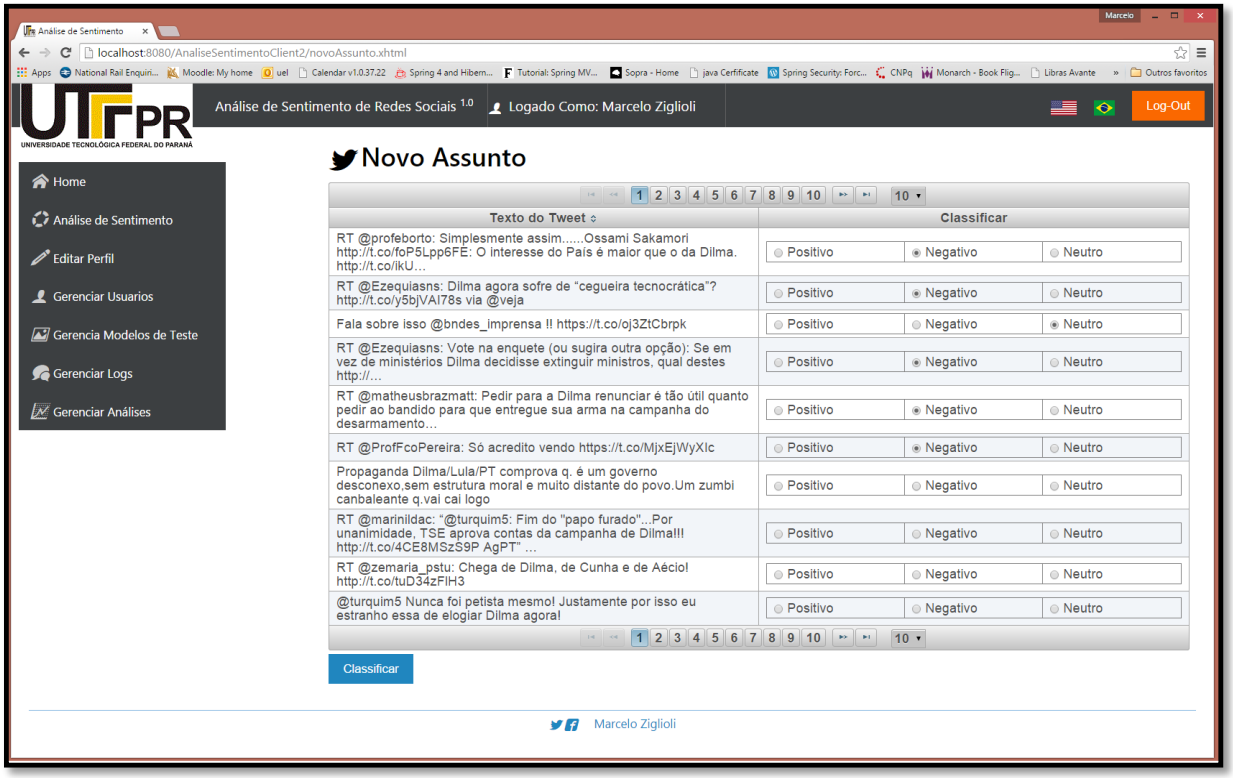


Figura 17 - Página Criar Novo Assunto. Classificar os *tweets*.
Fonte: Autoria Própria.

Após o usuário classificar os *tweets* e clicar no botão classificar o sistema dá início a etapa de pós-processamento e avaliação do conhecimento. Este processo envolve quatro etapas:

Primeira: o sistema cria de um arquivo ARFF no Apêndice conforme Figura 31 para ser utilizado pelo novo classificar através do *framework* WEKA.

Segunda: o sistema cria um novo classificador, utilizando o arquivo criado pela primeira etapa, o código desta fase pode ser visualizado no Apêndice na Figura 32.

Terceira: validação da nova instância do classificador. Esta validação é responsável por verificar os erros e acertos do classificador utilizando o processo de validação cruzada. O código desta etapa pode ser visualizado no Apêndice na Figura 33.

Quarta: verificar a quantidade de cada “classe” classificada na etapa anterior. Para este estudo foram utilizadas, positivo, negativo e neutro como atributos classe para cada *tweet*

classificado manualmente. Para quantificar esses atributos foi utilizada uma matriz de confusão conforme a Figura 34 no Apêndice.

Por meio da Figura 18 pode-se observar o resultado obtido após o pós-processamento e avaliação do conhecimento.

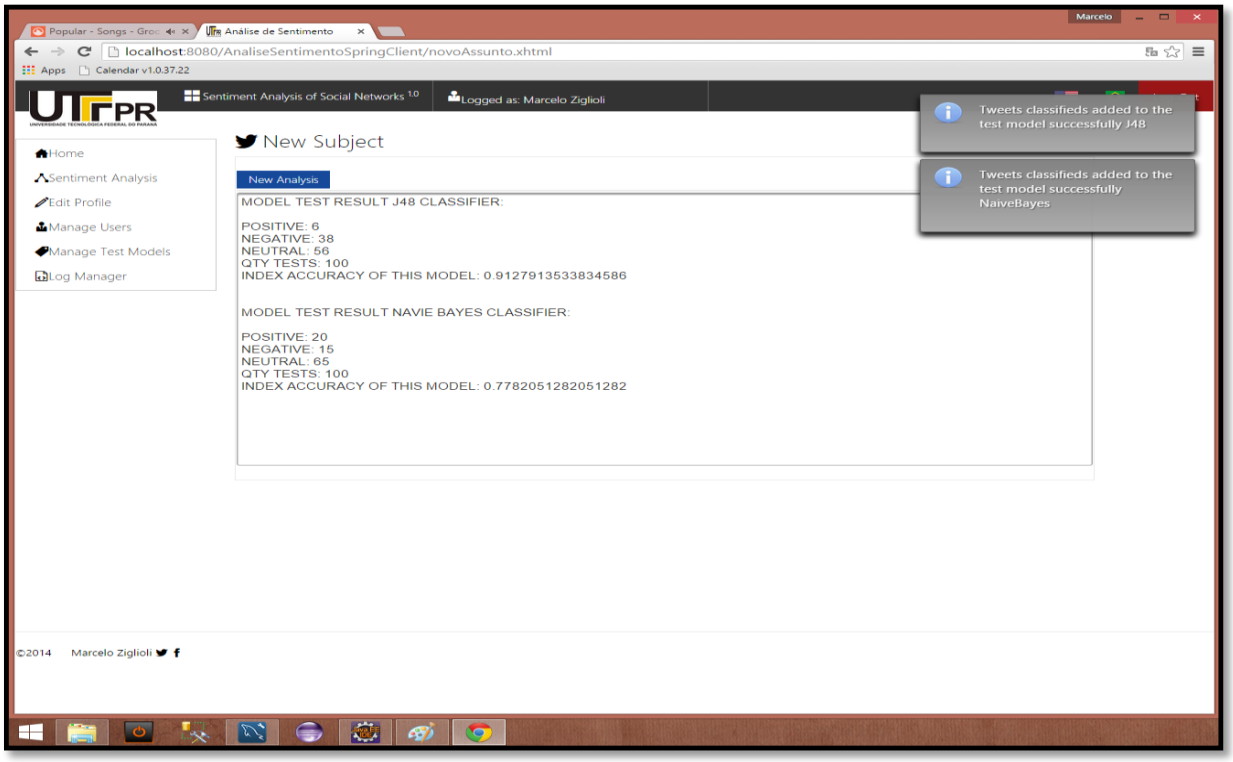


Figura 18 - Página Criar Novo Assunto. Criação do Modelo de Teste concluída.
Fonte: Autoria Própria.

O sistema possibilita criar uma nova análise para um assunto que já foi previamente criado. Para isto basta preencher alguns campos e clicar no botão nova análise conforme a Figura 14. Esta ação busca no sistema o modelo de teste referente ao assunto selecionado e efetua a análise dos dados com o mesmo. O código referente a esta etapa pode ser visualizado na Figura 35.

Os resultados das análises de sentimento, que são a criação de modelos de teste e os respectivos resultados de análise, podem ser visualizadas nas Figuras 19 e 20.

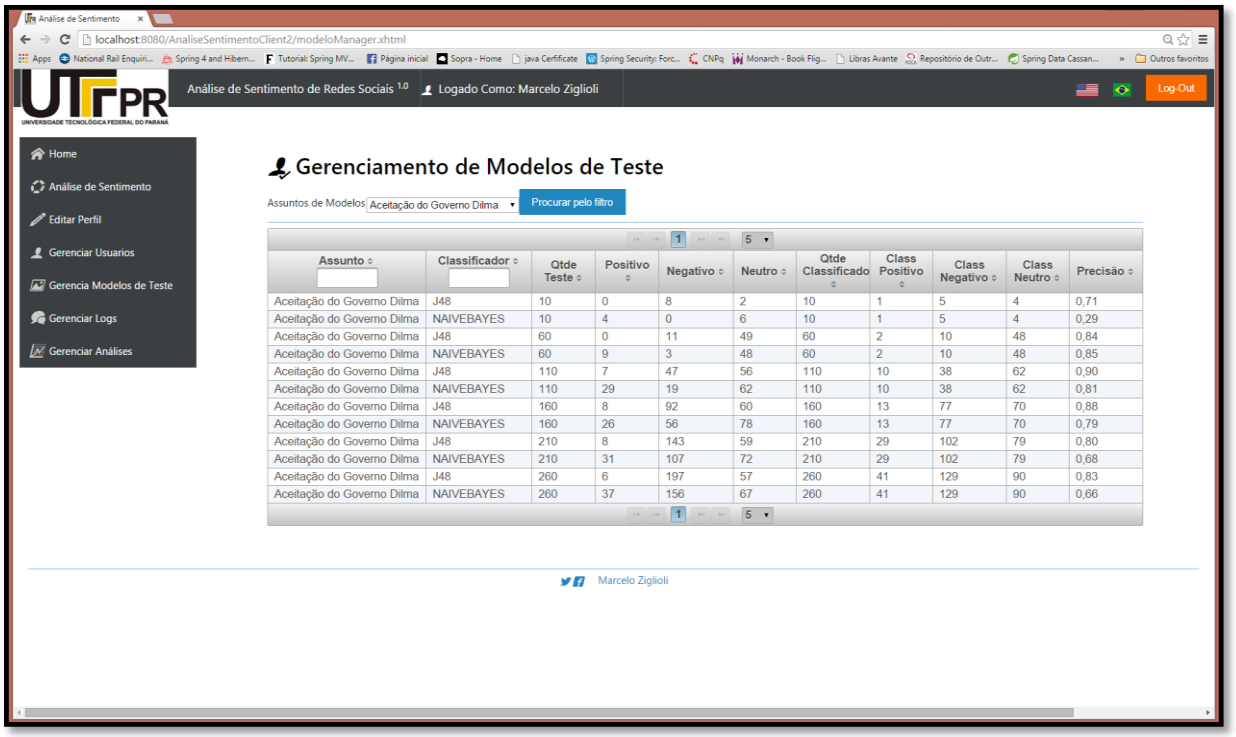


Figura 19 - Página Gerenciar Modelos de Teste.
Fonte: Autoria Própria.

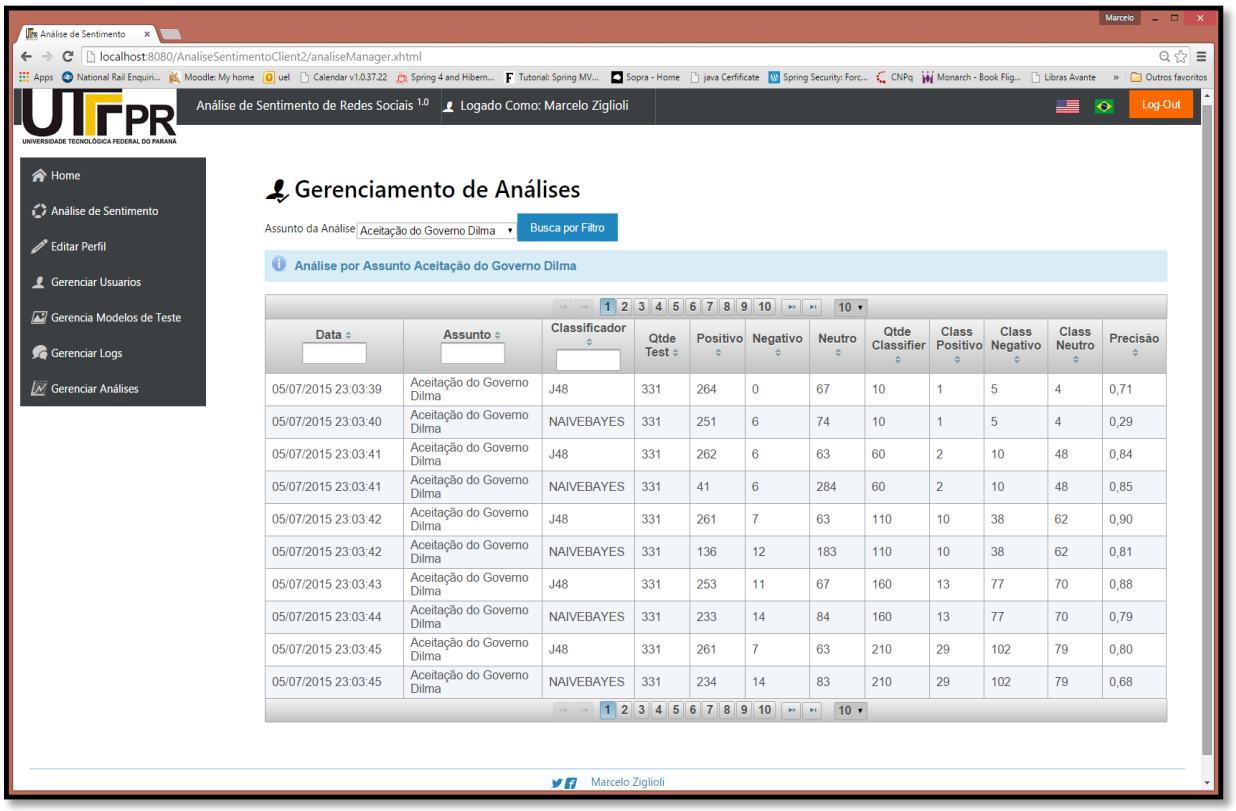


Figura 20 - Página Gerenciar Analises.
Fonte: Autoria Própria.

5.3 ESTUDO DE CASO

A aceitação do governo Dilma foi o tema selecionado como estudo de caso, através da rede social Twitter, foram adquiridos 384 *tweets* e foram classificados manualmente.

Os dados foram coletados e pré processados conforme a Figura 9. Após a coleta e pré-processamento dos dados o sistema em forma de uma tabela irá pedir ao usuário que classifique alguns dos *tweets* manualmente, conforme o critério informado em uma caixa de diálogo. Conforme o diagrama na Figura 21.

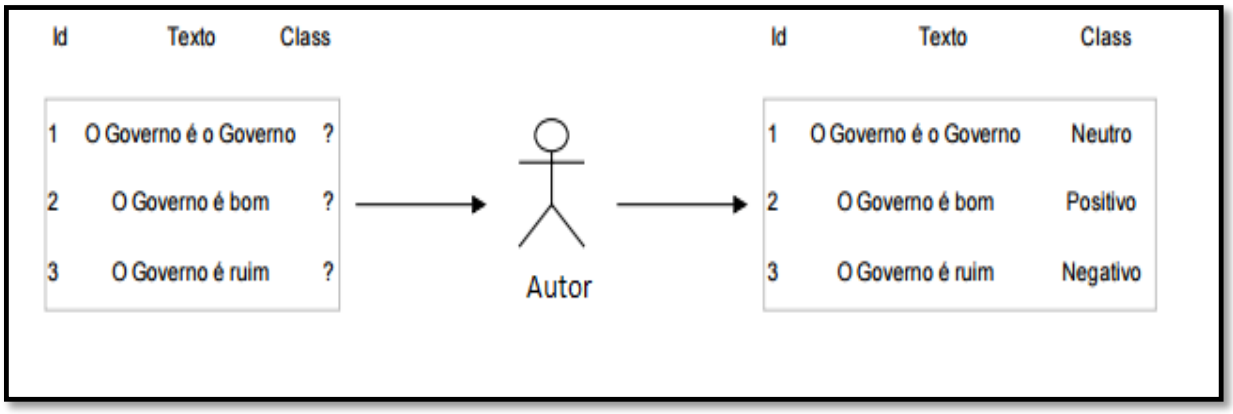


Figura 21 - Diagrama da Classificação dos *tweets.*
Fonte: Autoria Própria.

A partir dos *tweets* classificados o sistema irá criar várias instâncias de teste, através dos algoritmos de classificação Multinominal Naive Bayes e Árvores de decisão J48, que estão disponíveis no framework Weka. Estas instâncias Weka, chamadas de instâncias de treino, serão salvas em formato de arquivo com a extensão ".model" e serão utilizadas posteriormente na classificação de novos *tweets*. Para cada instância de treino o sistema irá criar um novo modelo de teste que será gravado no banco de dados e terá os dados relevantes ao processo de classificação, como a quantidade de *tweets* classificados, a quantidade de acertos, erros e a precisão do classificador, entre outros dados. O sistema possui uma página para gerenciamento dos modelos de teste, no qual o Administrador poderá visualizar todos os dados de cada classificador, assim podendo determinar qual algoritmo e qual instância irá gerar dados satisfatórios.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

As ferramentas estudadas e utilizadas nesta monografia, foram de grande importância para que o projeto tivesse êxito. Cada tecnologia utilizada dispunha de benefícios distintos e foram agrupadas para a realização do trabalho.

Entre os benefícios encontrados na utilização da análise de sentimento, está a rapidez de analisar uma grande quantidade de dados encontrados na rede social Twitter e obter os resultados esperados no AM. Percebeu-se ainda que com a implantação do software utilizado, tornou-se possível a identificação e aquisição de conhecimento para humanos, dos dados contidos em redes sociais. Dados estes que estão em grande quantidade e possivelmente difícil de serem analisados manualmente pelas pessoas.

Percebeu-se que o framework WEKA utilizado para o AM, parte mais objetiva do projeto, apresentou uma facilidade na sua utilização, pois possui um conjunto de bibliotecas que pode ser facilmente adicionado ao projeto Java, tornado possível a sua utilização no sistema. Esta ferramenta também demonstrou bons resultados perante aos dados utilizados, pode-se verificar que os algoritmos de classificação, J48 e Naive Bayes, possuem resultados de precisão equivalentes e satisfatórios sobre os dados, tornado os testes confiáveis e portanto poderá ser utilizado em um sistema em produção. Notou-se ainda que estes algoritmos não diferem na sua precisão quanto a quantidade de exemplos utilizados para efetuar a análise.

Outro benefício do sistema é a integração com o usuário, através de uma interface *web* de fácil utilização, onde o mesmo poderá efetuar análises de sentimento de qualquer assunto de interesse.

Como um todo o projeto proporcionou um grande conhecimento tanto na área de AM quanto na arquitetura e desenvolvimento de software através de *frameworks* amplamente utilizados no mercado de trabalho.

6.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Como trabalhos que poderão ser estudados após a apresentação deste trabalho estão a possibilidade de adicionar novos algoritmos de mineração de textos, e testar novos algoritmos que apresentem uma melhor precisão de acertos. Ainda existe a possibilidade de estender o aprendizado de máquina, adicionando o aprendizado não supervisionado ao sistema

Também como possível trabalho futuro está a troca do banco de dados relacional para um orientado a objetos, como por exemplo, o banco de dados Apache Cassandra, com o intuito de aumentar o volume de processamento em menor tempo.

REFERÊNCIAS BIBLIOGRÁFICAS

ALONSO, G., CASATI, F.; KUNO, H.; MACHIRAJU, V. **Web Services**. Zürich, Switzerland: Springer Berlin Heidelberg, 2004.

FACELI, K., LORENA, A.; GAMA, J.; CARVALHO, A. **INTELIGÊNCIA ARTIFICIAL Uma Abordagem de Aprendizado de Máquina**. Rio de Janeiro: LTC Livros Técnicos e Científicos Editora Ltda, 2011.

GUINWA, J. **UML by Example**. Cambridge: Cambridge University Press, 2004.
JBOSS COMMUNITY. Hibernate. **Hibernate ORM**, 17 maio 2014. Disponivel em: <http://hibernate.org/orm/>.

JSON. Introdução ao JSON. **Introdução ao JSON**, 17 maio 2014. Disponivel em: <http://www.json.org/json-pt.html>.

KIBRIYA, A., FRANK, E.; PFAHRINGER, B.; HOLMES, G. Multinominal Naive Bayes for text Categorization Revisited. **Department of Computer Science University of Waikato**, 06 nov. 2008.

LIU, S.; OLIVER, K. Hosebird Client (HBC). **GitHub**, 17 maio 2014. Disponivel em: <https://github.com/twitter/hbc>.

MIT LABORATORY FOR COMPUTER SCIENCE AND RSA DATA SECURITY. The MD5 Message-Digest Algorithm. **The MD5 Message-Digest Algorithm**, 06 jul. 2015. Disponivel em: <http://tools.ietf.org/html/rfc1321?ref=driverlayer.com>.

ORACLE. Java Persistence API. **Java Persistence API**, 17 maio 2014. Disponivel em: <http://docs.oracle.com/javaee/5/tutorial/doc/bnbpz.html>.

ORACLE. JavaMail API documentation. **JavaMail API documentation**, 05 jul. 2015. Disponivel em: <https://javamail.java.net/nonav/docs/api/>.

PIMENOV, S. https://metroui.org.ua/. **https:** //metroui.org.ua/, 06 jan. 2015. Disponivel em: <https://metroui.org.ua/>.

PORTER, M. F. Snowball: A language for stemming algorithms. **Snowball:** A language for stemming algorithms, 17 maio 2014. Disponivel em: <http://snowball.tartarus.org/texts/introduction.html>.

PRIMEFACES. PrimeFaces. **PrimeFaces**, 19 maio 2014. Disponivel em: <http://www.primefaces.org/>.

REZENDE, S. O. **Sistemas Inteligentes Fundamentos e Aplicações**. 1ª. ed. Barueri SP: Manole Ltda, 2003.

SILVEIRA, P., SILVEIRA, G.; LOPES, S.; MOREIRA, G. KUNG, Fabio et al. **Introdução à Arquitetura e Design de Software**. Rio de Janeiro: Elsevier, 2012.

SRIRANGAN. **Apache Maven 3 Cookbook**. Birmingham Uk: Packt Publishing Ltd, 2011.
TAYLOR AND FRANCIS GROUP, LLC. **Natural Language Processing**. Boca Raton, FL, USA: Chapman & Hall/CRC, 2010.

THE APACHE SOFTWARE FOUNDATION. Apache Maven Project. **Apache Maven Project**, 17 maio 2014. Disponivel em: <http://maven.apache.org/>.

THE APACHE SOFTWARE FOUNDATION. Apache Tomcat. **Apache Tomcat**, 17 maio 2014. Disponivel em: <http://tomcat.apache.org/>.

THE SPRING TEAM. Spring Framework. **Spring Framework**, 06 jul. 2015. Disponivel em: <http://projects.spring.io/spring-framework/>.

THE SPRING TEAM. Spring Security. **Spring Security**, 06 jul. 2015. Disponivel em: <http://projects.spring.io/spring-security/>.

TWITTER DEVELOPERS. https://dev.twitter.com/docs/streaming-apis. **Twitter**, 2014. Disponivel em: <https://dev.twitter.com/docs/streaming-apis>. Acesso em: 19 mar. 2014.

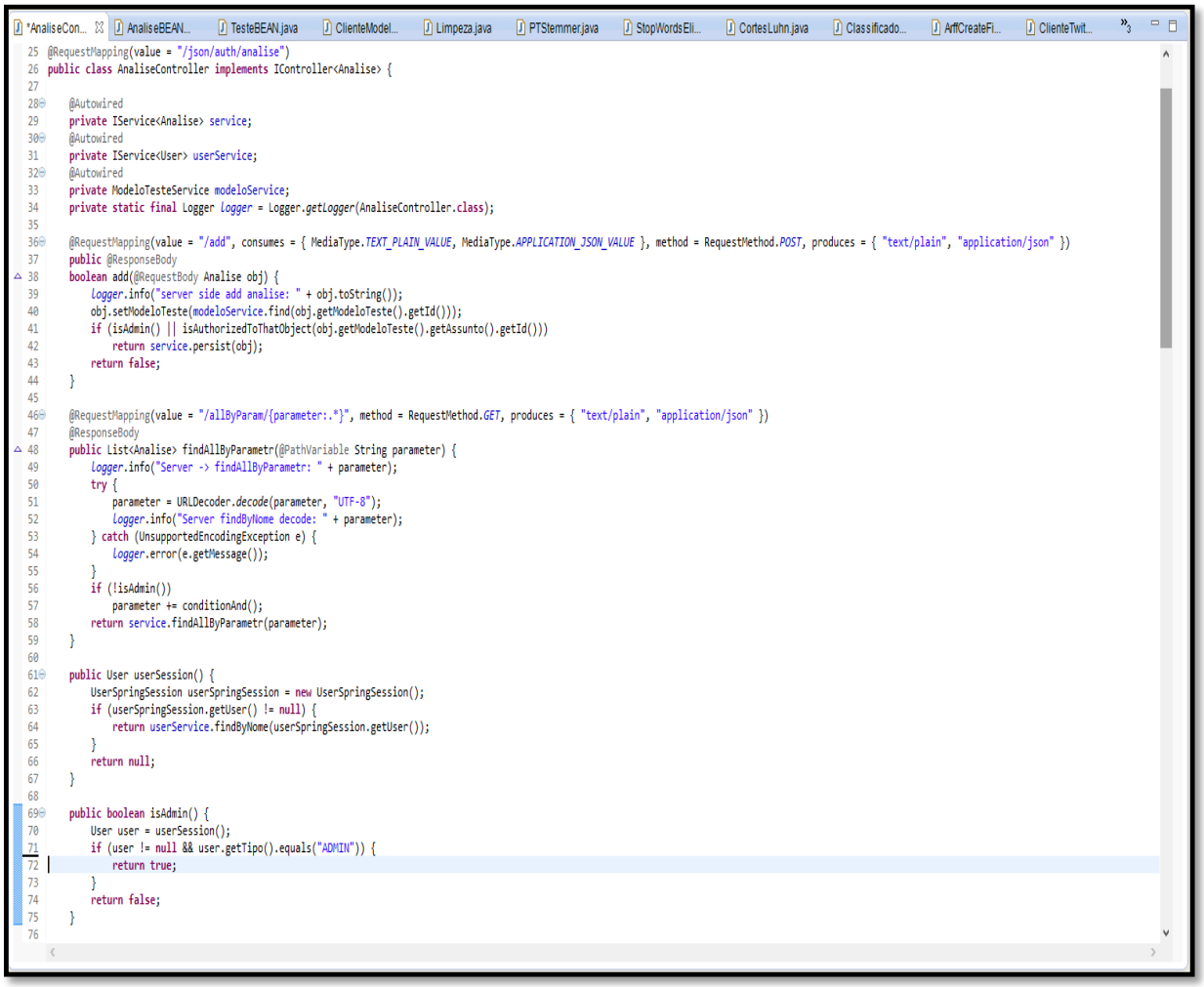
VENUGOPAL, K. R.; PATNAIK, L. M. **Computer Networks and Intelligent Computing**. Bangalore: Springer, 2011.

WAIKATO, M. L. G. A. T. U. O. Weka The University of Waikato. **Weka**, 2014. Disponivel em: <http://www.cs.waikato.ac.nz/ml/weka/>. Acesso em: 17 abr. 2014.

WIDENIUS, M.; AXMARK, D. **MySQL Reference Manual Documentation from the Source**. Sebastopol: O'Reilly Media, 2002.

ZHANG, H. The Optimality of Naive Bayes. **American Association for Artificial Intelligence (www.aaai.org)**, 2004.

APÊNDICE



```java
@RequestMapping(value = "/json/auth/analise")
public class AnaliseController implements IController<Analise> {

    @Autowired
    private IService<Analise> service;
    @Autowired
    private IService<User> userService;
    @Autowired
    private ModeloTesteService modeloService;
    private static final Logger logger = Logger.getLogger(AnaliseController.class);

    @RequestMapping(value = "/add", consumes = { MediaType.TEXT_PLAIN_VALUE, MediaType.APPLICATION_JSON_VALUE }, method = RequestMethod.POST, produces = { "text/plain", "application/json" })
    public @ResponseBody
    boolean add(@RequestBody Analise obj) {
        logger.info("server side add analise: " + obj.toString());
        obj.setModeloTeste(modeloService.find(obj.getModeloTeste().getId()));
        if (isAdmin() || isAuthorizedToThatObject(obj.getModeloTeste().getAssunto().getId()))
            return service.persist(obj);
        return false;
    }

    @RequestMapping(value = "/allByParam/{parameter:.*}", method = RequestMethod.GET, produces = { "text/plain", "application/json" })
    @ResponseBody
    public List<Analise> findAllByParametr(@PathVariable String parameter) {
        logger.info("Server -> findAllByParametr: " + parameter);
        try {
            parameter = URLDecoder.decode(parameter, "UTF-8");
            logger.info("Server findByNome decode: " + parameter);
        } catch (UnsupportedEncodingException e) {
            logger.error(e.getMessage());
        }
        if (!isAdmin())
            parameter += conditionAnd();
        return service.findAllByParametr(parameter);
    }

    public User userSession() {
        UserSpringSession userSpringSession = new UserSpringSession();
        if (userSpringSession.getUser() != null) {
            return userService.findByNome(userSpringSession.getUser());
        }
        return null;
    }

    public boolean isAdmin() {
        User user = userSession();
        if (user != null && user.getTipo().equals("ADMIN")) {
            return true;
        }
        return false;
    }

}
```

Figura 22 - Código de Criação dos serviços de análises do sistema.
Fonte: Autoria Própria.

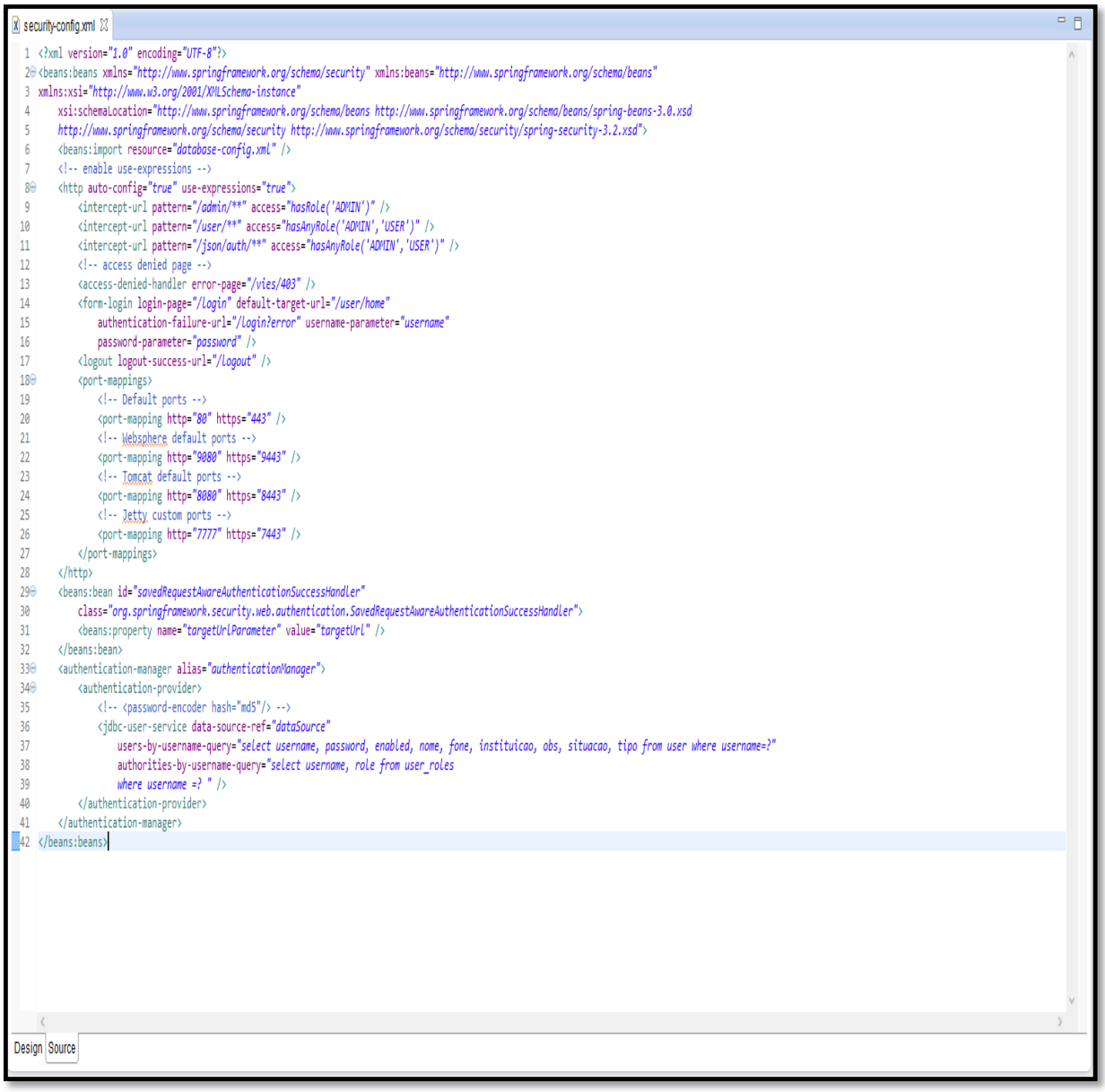


Figura 23 - Arquivo xml para a configuração do Spring Security.
Fonte: Autoria Própria.

LoginBEAN.java

```
145  public void login() {
146      System.out.println("teset");
147      logger.info(msg.getString(MessageBundleHelper.LOGIN_SUCESSFULLY));
148      if (email != "" && senha != "") {
149          user = restUser.login(email, HashMd5.md5(senha));
150          if (user != null) {
151              logger.info(" user: " + user.getUsername() + " : " + user.getPassword());
152              UserSession.setUserSession(user);
153              iniciar();
154              try {
155                  UserSession.getExternalContext().redirect(PagesHelper.HOME);
156              } catch (IOException ioe) {
157                  logger.error(ioe.getMessage());
158                  throw new FacesException(ioe);
159              }
160          } else {
161              if (restUser.enable(email)) {
162                  FacesMessageHelper.addErrorMessage(msg.getString(MessageBundleHelper.USUARIO_NAO_ENCONTRADO));
163                  logger.error(" cliente nao eee encontrado");
164              } else {
165                  FacesMessageHelper.addErrorMessage(msg.getString(MessageBundleHelper.USUARIO_BLOQUEADO));
166                  logger.error(" cliente bloqueado");
167              }
168              senha = "";
169              addTentativa();
170          }
171      } else {
```

RestUser.java

```
124  public User login(String username, String password) {
125      try {
126          User obj = new User();
127          obj.setUsername(username);
128          obj.setPassword(password);
129          HttpEntity<User> requestEntity = new HttpEntity<User>(obj,
130                  auth.buildHeader());
131          ResponseEntity<User> responseEntity = restTemplate.exchange(
132                  URL.USER_SERVER_URI_LOGIN, HttpMethod.POST, requestEntity,
133                  User.class);
134          logger.info(" User: "
135                  + auth.getUserString() + " login: " + username);
136          return responseEntity.getBody();
137      } catch (Exception e) {
138          logger.error(" User: "
139                  + auth.getUserString() + e.getMessage());
140      }
141      return null;
142  }
```

Figura 24 - Código referente ao login do usuário.
Fonte: Autoria Própria.

LoginBEAN.java

```
public void login() {
    System.out.println("teset");
    logger.info(msg.getString(MessageBundleHelper.LOGIN_SUCESSFULLY));
    if (email != "" && senha != "") {
        user = restUser.login(email, HashMd5.md5(senha));
        if (user != null) {
            logger.info(" user: " + user.getUsername() + " : " + user.getPassword());
            UserSession.setUserSession(user);
            iniciar();
            try {
                UserSession.getExternalContext().redirect(PagesHelper.HOME);
            } catch (IOException ioe) {
                logger.error(ioe.getMessage());
                throw new FacesException(ioe);
            }
        } else {
            if (restUser.enable(email)) {
                FacesMessageHelper.addErrorMessage(msg.getString(MessageBundleHelper.USUARIO_NAO_ENCONTRADO));
                logger.error(" cliente nao eee encontrado");
            } else {
                FacesMessageHelper.addErrorMessage(msg.getString(MessageBundleHelper.USUARIO_BLOQUEADO));
                logger.error(" cliente bloqueado");
            }
            senha = "";
            addTentativa();
        }
    } else {
```

RestUser.java

```
public User login(String username, String password) {
    try {
        User obj = new User();
        obj.setUsername(username);
        obj.setPassword(password);
        HttpEntity<User> requestEntity = new HttpEntity<User>(obj,
                auth.buildHeader());
        ResponseEntity<User> responseEntity = restTemplate.exchange(
                URL.USER_SERVER_URI_LOGIN, HttpMethod.POST, requestEntity,
                User.class);
        logger.info(" User: "
                + auth.getUserString() + " login: " + username);
        return responseEntity.getBody();
    } catch (Exception e) {
        logger.error(" User: "
                + auth.getUserString() + e.getMessage());
    }
    return null;
}
```

Figura 25 - Código para enviar um novo e-mail com o *link* para criar nova senha.
Fonte: Autoria Própria.

ResetPasswordController.java

```
@RequestMapping(value = "/request/{email:.*}", method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public String request(@PathVariable String email) {
    logger.info("request: " + email);
    try {
        email = URLDecoder.decode(email, "UTF-8");
        logger.info("Server email decode: " + email);
    } catch (UnsupportedEncodingException e) {
        logger.error(e.getMessage());
    }
    email = email.replaceAll("'", "");
    Resetpassword rp = service.findByParameter(" email = '" + email + "' and isUsed = 0 and valid > sysdate()");
    logger.info("rp: " + rp);
    if (rp != null) {
        logger.info("rp: " + rp.isUsed() + " " + Calendar.getInstance() + " " + rp.getValid());
        logger.info(Calendar.getInstance().before(rp.getValid()));
        return "editarPassword.xhtml?token=s.a.utfpr_" + new Md5PasswordEncoder().encodePassword(String.valueOf(rp.getEmail()), null) + "t_"
                + rp.getToken();
    }
    logger.info("create a new one");
    // create a new one
    rp = createRequest(email);
    if (service.persist(rp)) {
        return "editarPassword.xhtml?token=s.a.utfpr_" + new Md5PasswordEncoder().encodePassword(String.valueOf(rp.getEmail()), null) + "t_"
                + rp.getToken();
    }
    return null;
}

private Resetpassword createRequest(String email) {
    Calendar c = Calendar.getInstance();
    Resetpassword rp = new Resetpassword();
    rp.setDate(c.getTime());
    rp.setEmail(email);
    rp.setUsed(false);
    c.add(Calendar.DATE, 1);
    rp.setValid(c.getTime());
    rp.setToken(new Md5PasswordEncoder().encodePassword(String.valueOf(Calendar.getInstance().getTimeInMillis()), null));
    return rp;
}
```

Figura 26 - Código de criação de um novo *token* para nova senha.
Fonte: Autoria Própria.

```
@SuppressWarnings("deprecation")
public void getTweets(int qtde) throws InterruptedException {
    // connectar ao api streaming
    Client client = construirClienteTwitter();
    client.connect();
    int twPersist = 0;
    // manipular as mensagens
    for (int msgRead = 0; twPersist < qtde; msgRead++) {
        String msg = decodeUtf8(queue.take().getBytes());
        if (msg != null) {
            // calcular a porcentagem concluida ate agora para mostrar na barra de progresso
            double barraProgressoDouble = (twPersist * 100) / qtde;
            if (barraProgressoDouble > 100)
                barraProgressoDouble = 100;
            int barraProgresso = (int) Math.round(barraProgressoDouble);
            logger.info(" barraProgresso: " + barraProgresso);
            SessionBarraProgresso.setBarraProgresso(barraProgresso);
            try {
                JSONObject json = new JSONObject(msg);
                if (json != null) {
                    String jsonText = json.getString("text");
                    String jsonData = json.getString("created_at");
                    Twitter twitter = new Twitter();
                    twitter.setText(jsonText);
                    twitter.setIdioma(filtroLinguas.toString());
                    twitter.setData(new Date());
                    twitter.setAssunto(assunto);
                    twitter.setDataTweeter(new Date(jsonData));
                    twitter.setFiltro(filtroPalavras.toString());
                    List<Palavra> aux = new ArrayList<Palavra>();
                    logger.info(" read: " + msgRead + " -> " + jsonText);
                    // limpeza das plavras
                    Limpeza textMinning = new Limpeza();
                    List<String> palavras = textMinning.efetuarLimpeza(jsonText);
                    // remover stopwords
                    if (removerStopword) {
                        StopWordsEliminator swe = new StopWordsEliminator();
                        if (addStopWords != null)
                            swe.setStopWordsByUser(addStopWords);
                        palavras = swe.removeStopWords(palavras);
                    }
                    // criar stemm
                    if (criarStemm) {
                        PTStemmer ptStemmer = new PTStemmer();
                        palavras = ptStemmer.getPorterStemmer(palavras);
                    }
                    // cortes de luhn
                    if (efetuarCortesLuhn) {
                        CortesLuhn cortesLuhn = new CortesLuhn(maxCortesLuhn, minCortesLuhn);
                        palavras = cortesLuhn.efetuarCortesDeLuhn(palavras);
                    }
                    for (String palavra : palavras) {
                        // persist possui validacao unique pelo nome da palavra
                        restPalavra.add(new Palavra(palavra, null));
                        // busca a palavra pelo nome para adicionar ao twitter
                        Palavra p = restPalavra.getByParameter(" palavra = '" + palavra + "'");
                        if (p != null) {
                            aux.add(p);
                        }
                    }
                    twitter.setPalavras(aux);
                    if (restTwitter.add(twitter)) {
                        twPersist++;
                        logger.info(" persistiu " + twPersist);
                    }
                }
            } catch (JSONException e) {
                logger.error(e.getMessage());
                e.printStackTrace();
            }
        }
    }
    client.stop();
}
```

Figura 27 - Código da busca de tweets.
Fonte: Autoria Própria.

```java
public class Limpeza {

    public Limpeza() {

    }

    public List<String> efetuarLimpeza(String texto) {
        List<String> listaPalavras = new ArrayList<String>();
        String[] splited = texto.split("\\s+");
        for (String st : splited) {
            // nao adicionar enderecos eletronicos nem palavras menor q 1 e null
            if (!st.contains("http") && !st.contains("www")) {
                // remove caracteres especiais
                String s = removeRegex(st);
                if (!s.equals("") && s.length() > 1)
                    listaPalavras.add(s);
            }
        }
        return listaPalavras;
    }

    private String removeRegex(String string) {
        // retirar todos os numeros da string
        string = string.replaceAll("\\d", "");
        // System.out.println("antes:2 " + string);
        Pattern pt = Pattern.compile("[^a-zA-ZçÇàáÁÀãÃõÕéÉÓóúÚôÔêÊ\\s]");
        Matcher match = pt.matcher(string);
        while (match.find()) {
            String s = match.group();
            if (s != null) {
                // retirar todos os regex declarados acima
                string = string.replaceAll("\\" + s, "");
            }
        }
        return string;
    }
}
```

Figura 28 - Código para a etapa de limpeza.
Fonte: Autoria Própria.

```java
public class StopWordsEliminator {

    private final String FILE_STOPWORDS = RealPath.getPATH() + "src/main/resources/stopwords.txt";
    private final String FILE_STOPWORDS_TWITTER = RealPath.getPATH() + "src/main/resources/stopwordstwitter.txt";
    private List<String> stopWordsByUser;

    public StopWordsEliminator() {}

    @SuppressWarnings("resource")
    public List<String> getStopWords(String sw) {
        BufferedReader reader;
        List<String> stopwords = new ArrayList<String>();
        try {
            reader = new BufferedReader(new FileReader(sw));
            String line = null;
            while ((line = reader.readLine()) != null) {
                stopwords.add(line);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return stopwords;
    }
    public List<String> removeStopWords(List<String> allWords) {
        Collection<String> allWordsLower = Collections2.transform(allWords, new Function<String, String>() {
            public String apply(String str) {
                return str.toLowerCase();
            }
        });
        Collection<String> stopWordsLower = Collections2.transform(getStopWords(FILE_STOPWORDS), new Function<String, String>() {
            public String apply(String str) {
                return str.toLowerCase();
            }
        });
        Collection<String> stopWordsTwitterLower = Collections2.transform(getStopWords(FILE_STOPWORDS_TWITTER), new Function<String, String>() {
            public String apply(String str) {
                return str.toLowerCase();
            }
        });
        allWordsLower.removeAll(stopWordsLower);
        allWordsLower.removeAll(stopWordsTwitterLower);
        if (stopWordsByUser != null) {
            allWordsLower.removeAll(stopWordsByUser);
        }
        return new ArrayList<String>(allWordsLower);
    }
    public List<String> getStopWordsByUser() {
        return stopWordsByUser;
    }
    public void setStopWordsByUser(List<String> stopWordsByUser) {
        this.stopWordsByUser = stopWordsByUser;
    }
}
```

Figura 29 - Código para a etapa de remoção das Stopwords.
Fonte: Autoria Própria.

```java
public class CortesLuhn {

    private int max;
    private int min;

    public CortesLuhn() {
        min = 1;
        max = 20;
    }
    public CortesLuhn(int max, int min) {
        this.max = max;
        if (max == 0)
            this.max = 20;
        this.min = min;
    }
    /**
     * Método responsavel por retornar uma lista de String contendo as palvaras
     * cortadas atraves do metodo de cortes de Luhn.
     * @author Marcelo Ziglioli
     * @return List
     */
    public List<String> getPalavrasCortadas(List<String> listaPalavras) {
        List<String> listaCortes = new ArrayList<String>();
        Set<String> uniqueSet = new HashSet<String>(listaPalavras);
        for (String temp : uniqueSet) {
            int freq = Collections.frequency(listaPalavras, temp);
            if (freq <= min || freq >= max) {
                listaCortes.add(temp);
            }
        }
        return listaCortes;
    }
    /**
     * Método responsavel por retornar uma lista de String contendo uma nova
     * lista sem as palvaras cortadas atraves do metodo de cortes de Luhn.
     * @author Marcelo Ziglioli
     * @return List
     */
    public List<String> efetuarCortesDeLuhn(List<String> listaPalavras) {
        List<String> listaCortes = new ArrayList<String>();
        Set<String> uniqueSet = new HashSet<String>(listaPalavras);
        for (String temp : uniqueSet) {
            int freq = Collections.frequency(listaPalavras, temp);
            if (freq > min && freq < max)
                listaCortes.add(temp);
        }
        return listaCortes;
    }
    /** GETTERS and SETTERS*/
```

Figura 30 - Código para a etapa de Cortes de Luhn.
Fonte: Autoria Própria.

```java
/**
 * Metodo responsavel em criar o Arff file com o map passado. O arff file tera elementos do tipo: positivo, negativo ou neutro.
 * @param mapPalavras Map<Long, List<String>>
 * @return Instances
 * */
private Instances makeArff(Map<Long, List<String>> mapPalavras) {
    // definir os elementos classificador
    FastVector fvClassVal = new FastVector(2);
    fvClassVal.addElement("positivo");fvClassVal.addElement("negativo");fvClassVal.addElement("neutro");
    // nova lista para guardar os atributos que nao se repetirao
    List<String> listaAtributos = new ArrayList<String>();
    // os elementos serao todas as palavras, ou seja, para cada linha repetir todas as palavras constantes
    // o FastVector e para criar uma coluna para cada palavra. colocar dentro de um hashset para eliminiar repeticoes
    palavras = new FastVector();
    for (String s : new HashSet<String>(allWords)) {
        palavras.addElement(new Attribute(s));
        listaAtributos.add(s);
    }
    // adicionar como ultimo atributo o elemento classificador
    Attribute ClassAttribute = new Attribute("classe", fvClassVal);
    palavras.addElement(ClassAttribute);
    // criar uma nova instancia do weka
    instance = new Instances("MyRelation", palavras, 10);
    // entrar em todos os elementos do map para preencher as colunas para cada twitter
    for (Entry<Long, List<String>> lista : mapPalavras.entrySet()) {
        // criar uma instancia do tamanho de todas as palavras
        Instance iExample = new Instance(palavras.size());
        // criar um array de boolean da quantidade das palavras, ou seja,
        // para cada palavra devera verificar se existe neste twitter, se existir true, senao false
        words = new boolean[palavras.size()];
        // percorrer cada palavra para atribuir o valor da coluna especificada
        for (int i = 0; i < palavras.size(); i++) {
            // este if verifica se a ultima coluna, se for tera que colocar a classificacao
            if (i == palavras.size() - 1) {
                // se for a ultima coluna, ir no banco e verificar qual a classificacao deste twitter
                Twitter twitter = findTwitterById(lista.getKey());
                if (twitter != null && twitter.getClassificacao() != null && twitter.getClassificacao() != "")
                    iExample.setValue((Attribute) palavras.elementAt(i), twitter.getClassificacao());
                else
                    // se nao conseguir encontrar a classificacao por algun motivo, set como neutro
                    iExample.setValue((Attribute) palavras.elementAt(i), "neutro");
            } else {
                // verificar se 'e true ou false, se for true coloca 1 se for false 0
                if (lista.getValue().contains(listaAtributos.get(i))) {
                    words[i] = true;
                    iExample.setValue((Attribute) palavras.elementAt(i), 1);
                } else {
                    words[i] = false;
                    iExample.setValue((Attribute) palavras.elementAt(i), 0);
                }}}
        instance.add(iExample);
    }
    // setar o classificador para a instancia
    instance.setClass(ClassAttribute);
    // escrever o arff em um file
    writeArff(instance);
    return instance;
}
```

Figura 31 - Código para criação do arquivo Arff.
Fonte: Autoria Própria.

```java
/**
 * Metodo responsavel em criar um classificador.
 * Ira criar um arquivo com a extenção .model
 * @param instance Instance
 * @param modeloTeste ModeloTeste
 * @param classificador boolean -> true = J48, false = NAIVE
 * @return modeloTeste
 * */
public ModeloTeste criarClassificador(ModeloTeste modeloTeste, Instances instance, boolean classificador) {
    try {
        instance.setClassIndex(instance.numAttributes() - 1);
        Classifier cModel;
        if(classificador)
            cModel = (Classifier) new J48();
        else
            cModel = (Classifier) new NaiveBayesMultinomial();
        cModel.buildClassifier(instance);
        SerializationHelper.write(SAVE_CLASSIFIER + modeloTeste.getClassificador() + "_" + modeloTeste.getAssunto().getId()
                + "_" + modeloTeste.getTwitters().size() + ".model", cModel);
        modeloTeste = crossValidation(modeloTeste, cModel, instance);
    } catch (Exception e) {
        logger.error(e.getMessage());
    }
    return modeloTeste;
}
```

Figura 32 - Código utilizado para criar um classificador.
Fonte: Autoria Própria.

```java
/**
 * Metodo responsavel em criar um validação cruzada da instancia de teste. Utiliza as classes Classifier e Instances do Weka para efetuar a validacao
 * @param modeloTeste
 * @param cModel
 * @param instance
 * @return ModeloTeste
 * */
public ModeloTeste crossValidation(ModeloTeste modeloTeste, Classifier cModel, Instances instance) {
    int folds = 10;
    // nao aceitar valor menor de folds doque de instancias
    if (instance.numInstances() < folds)
        folds = instance.numInstances();
    Instances data = new Instances(instance); // copia original
    if (data.classAttribute().isNominal())
        data.stratify(folds);
    Evaluation eval;
    try {
        eval = new Evaluation(data);
        for (int n = 0; n < folds; n++) {
            // instancia de treino
            Instances train = data.trainCV(folds, n);
            // instancia de teste
            Instances test = data.testCV(folds, n);
            // construir um classificador
            cModel.buildClassifier(train);
            // avaliar o modelo de teste cModel
            eval.evaluateModel(cModel, test);
        }
        // atribuir os valores da avaliacao para o modelo de teste que sera gravado no banco de dados
        modeloTeste.setAcertos((int) eval.correct());
        modeloTeste.setErros((int) eval.incorrect());
        modeloTeste.setQtdeTestes(modeloTeste.getAcertos() + modeloTeste.getErros());
        modeloTeste.setPrecisao(eval.weightedPrecision());
        // verificar a qtde de atributos classificados atraves da matrix de confusao
        double[] matrix = confusionMatrix(eval);
        if (matrix != null) {
            modeloTeste.setPositivos((int) matrix[0]);
            modeloTeste.setNegativos((int) matrix[1]);
            modeloTeste.setNeutros((int) matrix[2]);
        }
    } catch (Exception e) {
        logger.error(e.getMessage());
        e.printStackTrace();
    }
    return modeloTeste;
}
```

Figura 33 - Código da validação do Classificador.
Fonte: Autoria Própria

```java
/**
 * Metodo responsavel em criar uma matrix de confusao.
 * Utiliza Weka Evaluation class para criar a matrix.
 * @param evaluation
 * @return double[] -> [0] = negativo, [1] = positivo, [2] = neutro
 * */
public double[] confusionMatrix(Evaluation evaluation) {
    try {
        logger.info(" confusionMatrix: " + evaluation.toMatrixString());
        double[][] matrix = evaluation.confusionMatrix();
        double[] resultMatrix = new double[matrix.length];
        // zerar
        for (int i = 0; i < resultMatrix.length; i++) {
            resultMatrix[i] = 0;
        }
        for (int i = 0; i < matrix.length; i++) {
            for (int k = 0; k < matrix[i].length; k++) {
                resultMatrix[k] += matrix[i][k];
            }
        }
        return resultMatrix;
    } catch (Exception e) {
        logger.error(e.getMessage());
        e.printStackTrace();
    }
    return null;
}
```

Figura 34 - Código para a matriz de confusão.
Fonte: Autoria Própria.

```java
/**
 * @author Marcelo Metodo responsavel por retornar um modelo de teste com os resultados (acertos, erros...) onde
 * @param instanceTreino e a instancia recuperada dos testes
 * @param instanceNova e a instancia com os dados a serem analisados
 * */
public int[] classificador(ModeloTeste modeloTeste, String parameter, int qtdeTweets) {
    logger.info(" classificadorNaiveBayesMultinomial");
    // construir instancia de teste apartir do modelo de teste
    Instances instanceTreino = arff.makeArff(modeloTeste);
    // construir uma instancia nova com os parametros passados
    Instances instanceNova = arff.makeArffByParameter(parameter, qtdeTweets, instanceTreino);
    Classifier cModel;
    if (isClassificadorSerializable(modeloTeste)) {
        try {
            // buscar o classificador salvo
            String fileLocation = SAVE_CLASSIFIER + modeloTeste.getClassificador() + "_" + modeloTeste.getAssunto().getId()
                    + "_" + modeloTeste.getTwitters().size() + ".model";
            cModel = (Classifier) SerializationHelper.read(fileLocation);
            logger.info(" buscou o classificador no file -> " + fileLocation);
            return classificarNovaInstancia(cModel, instanceNova);
        } catch (Exception e1) {
            logger.error(e1.getMessage());
        }
    }
    return null;
}
public int[] classificarNovaInstancia(Classifier cModel, Instances instanceNova) {
    // set class attribute
    instanceNova.setClassIndex(instanceNova.numAttributes() - 1);
    // create copy
    Instances labeled = new Instances(instanceNova);
    logger.info(" -- classify Instances by " + cModel.getClass() + "--");
    for (int i = 0; i < instanceNova.numInstances(); i++) {
        try {
            double clsLabel = cModel.classifyInstance(instanceNova.instance(i));
            labeled.instance(i).setClassValue(clsLabel);
        } catch (Exception e) {
            logger.error(e.getMessage());
        }
    }
    int[] resultado = new int[3];
    for (int i = 0; i < labeled.numInstances(); i++) {
        String s = labeled.instance(i).stringValue(labeled.attribute(labeled.numAttributes() - 1));
        if (s.equals("negativo"))
            resultado[0]++;
        else if (s.equals("positivo"))
            resultado[1]++;
        else
            resultado[2]++;
    }
    logger.info(" positivo: " + resultado[0] + ", negativo: " + resultado[1] + ", neutro: " + resultado[2]);
    return resultado;
}
```

Figura 35 - Código referente à classificação de um novo Classificador.
Fonte: Autoria Própria.