FUNDAÇÃO GETÚLIO VARGAS
ESCOLA DE MATEMÁTICA APLICADA

CRISTIANO DE ANDRADE GONÇALVES

Análise de Sentimentos em Reclamações

Uma aplicação no maior site de Reclamações do Brasil

Rio de Janeiro, RJ
2016

Ficha catalográfica elaborada pela Biblioteca Mario Henrique Simonsen/FGV

Gonçalves, Cristiano de Andrade
Análise de sentimentos em reclamações: uma aplicação no maior site de reclamações do Brasil / Cristiano de Andrade Gonçalves. – 2016.
75 f.

Dissertação (mestrado) – Fundação Getulio Vargas, Escola de Matemática Aplicada.
Orientador: Flávio Codeço Coelho.
Inclui bibliografia.

1.Comportamento do consumidor – Modelos matemáticos. 2. Processamento da linguagem natural (Computação). 3. Mineração de dados (Computação). I. Coelho, Flávio Codeço. II. Fundação Getulio Vargas. Escola de Matemática Aplicada. III. Título.

CDD – 006.35

CRISTIANO DE ANDRADE GONÇALVES

Análise de Sentimentos em Reclamações

Uma aplicação no maior site de Reclamações do Brasil

Dissertação apresentada ao Programa de Pós–Graduação do ESCOLA DE MATEMÁTICA APLICADA da FUNDAÇÃO GETÚLIO VARGAS, como requisito parcial para obtenção do título de Mestre em MESTRADO EM MODELAGEM MATEMÁTICA DA INFORMAÇÃO.

Área de concentração: .

Orientador: Prof. FLÁVIO CODEÇO COELHO

Rio de Janeiro, RJ
2016

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador(a).

CRISTIANO DE ANDRADE GONÇALVES

Graduou–se em Administração de Empresas na PUC-RIO - Pontifícia Universidade Católica do Rio de Janeiro e Estatística na UERJ - Universidade do Estado do Rio de Janeiro. Trabalha na Petrobras desde 2006. Atualmente trabalha na área de Riscos Empresariais na Petrobras na gerência de Estudos e Modelos para Análise Quantitativa de Risco.

CRISTIANO DE ANDRADE GONÇALVES

ANÁLISE DE SENTIMENTOS EM RECLAMAÇÕES – UMA APLICAÇÃO NO MAIOR SITE DE RECLAMAÇÕES DO BRASIL

Dissertação apresentada ao Curso de Mestrado em Modelagem Matemática da Informação da Escola de Matemática Aplicada da Fundação Getulio Vargas para obtenção do grau de Mestre em Modelagem Matemática da Informação.

Data da defesa: 22/06/2016.

ASSINATURA DOS MEMBROS DA BANCA EXAMINADORA

Flavio Codeço Coelho
Orientador (a)

Renato Rocha Souza

Ligia Arruda Café

Vitor Teixeira de Almeida

Dedico esta dissertação à minha esposa Priscila, pelo apoio incondicional e constante incentivo.

Ao meu filho Bento, que apesar de pequeno, me motiva a concluir esta dissertação.

E a todos os professores da EMAP-FGV, que foram tão importantes na minha vida acadêmica e no desenvolvimento desta dissertação.

Agradecimentos

A Petrobras e área de Recursos Humanos por patrocinador e autorizar a efetivação deste mestrado, representados por **Antônio Sérgio Oliveira Santana** e **Gustavo André Dunzer**.

Ao **Leonardo de Almeida Matos Moraes** por entender e compreender a necessidade de conclusão desta dissertação.

Aos professores com quem tive aula **Alexandre Rademaker, Antônio Carlos Saraiva Branco, Asla Medeiros e Sá, Eduardo Fonseca Mendes, Flávio Codeço Coelho, Hugo A. de La Cruz Cansio, Moacyr Alvim Horta Barbosa da Silva, Pierre-Alexandre Bliman, Renato Rocha Souza, Vincent Gérard Yannick Guigues** pelo conhecimento, pela parceria e pela oportunidade de crescer e melhorar.

Ao meu orientador **Flávio Codeço Coelho** pela sua simplicidade, pelo seu exemplo como educador e pelas suas idéias e proposições construtivas.

Ao professor e ex-coordenador do curso **Renato Rocha Souza** pelo incentivo e orientação durante o Mestrado.

À FGV, instituição com a qual vou ter uma dívida de gratidão impagável, por ter me possibilitado evoluir e desenvolver.

There is no true interpretation of anything; interpretation is a vehicle in the service of human comprehension. The value of interpretation is in enabling others to fruitfully think about an idea.

Andreas Buja,
The Elements of Statistical Learning.

Resumo

GONÇALVES, CRISTIANO DE ANDRADE. **Análise de Sentimentos em Reclamações**. Rio de Janeiro, RJ, 2016. 75p. Dissertação de Mestrado. ESCOLA DE MATEMÁTICA APLICADA, FUNDAÇÃO GETÚLIO VARGAS.

A análise de sentimentos é uma ferramenta com grande potencial, podendo ser aplicada em vários contextos. Esta dissertação tem com o objetivo analisar a viabilidade da aplicação da técnica numa base capturada do site de reclamações mais popular do Brasil, com a aplicação de técnicas de processamento de linguagem natural e de aprendizagem de máquinas é possível identificar padrões na satisfação ou insatisfação dos consumidores.

Palavras–chave

Comportamento do Consumidor, Análise de Sentimentos, Processamento de Linguagem Natural e Aprendizagem por máquinas

Abstract

GONÇALVES, CRISTIANO DE ANDRADE. **T**. Rio de Janeiro, RJ, 2016. 75p. MSc. Dissertation. ESCOLA DE MATEMÁTICA APLICADA, FUNDAÇÃO GETÚLIO VARGAS.

The sentiment analysis is a tool with high potential and can be applied in various contexts. The aim of this work is to analyze the feasibility of the technique applying on a database webscaped from the most popular website complaints in Brazil, with the application of natural language processing techniques and machine learning can identify patterns in satisfaction or dissatisfaction of consumers.

Keywords

Consumer Behavior, Sentimet Analysis, Natural Language Processing, Machine Learning

Sumário

Lista de Figuras 13

Lista de Tabelas 14

Lista de Códigos de Programas 16

1 Introdução 17

2 Revisão Bibliográfica 19
- 2.1 Trabalhos Relacionados 19
- 2.2 Conceitos e Fórmulas matemáticas 22
  - 2.2.1 Tf-idf 22
  - 2.2.2 Latent Semantic Analysis (LSI) 23
  - 2.2.3 Regressão Logística 24
  - 2.2.4 Ridge Classifier 24
  - 2.2.5 Support Vector Machine - SVM 24
  - 2.2.6 SGDClassifier - Gradiente Descendente Estocástico 25
  - 2.2.7 Cross-validation e GridSearchCV 27

3 Metodologia 28
- 3.1 Captura dos dados 28
  - 3.1.1 Coleta dos dados: Primeira Etapa 29
  - 3.1.2 Coleta dos dados: Segunda Etapa 31
  - 3.1.3 Coleta dos dados: Terceira Etapa 37
- 3.2 Tratamento dos Dados 50
  - 3.2.1 Correção Ortográfica 50
  - 3.2.2 Construção da Matriz LSI 57

4 Pesquisa 60
- 4.1 Pesquisa 60
- 4.2 Análise Exploratória dos Dados 60
- 4.3 Definição das Classes 66
- 4.4 Modelos de Classificação 67
  - 4.4.1 Primeiro Modelo 68
  - 4.4.2 Segundo Modelo 69
  - 4.4.3 Terceiro Modelo 69
  - 4.4.4 Quarto Modelo 71
  - 4.4.5 Quinto Modelo 71

5 Conclusão **73**
5.1 Conclusão 73

Referências Bibliográficas **74**

Lista de Figuras

4.1 ViolinPlot: Distribuição das Notas por empresa 61
4.2 ViolinPlot: Distribuição das Notas e Resolução da reclamação por empresa 63
4.3 ViolinPlot: Distribuição do Tempo e Resolução da reclamação por nota 64
4.4 ViolinPlot: Distribuição da Nota e Resolução da reclamação por Estado 64
4.5 ViolinPlot: Distribuição Exclamações e Resolução da reclamação por nota 65
4.6 ViolinPlot: Distribuição de Tokens e Resolução da reclamação por nota 65
4.7 ViolinPlot: Distribuição do Vocabulário e Resolução da reclamação por nota 66
4.8 ViolinPlot: Distribuição do Termo Justiça e Resolução da reclamação por nota 66

Lista de Tabelas

4.1 Quantidade de Reclamações por empresas 62
4.2 Média e Desvio-padrão das Notas por empresas 62
4.3 Agrupamento das Notas por empresa 67
4.4 Parâmetros utilizados no GridSearchCV por algoritmo de aprendizagem de máquinas 68
4.5 Ridge com 500 vetores + duração 70
4.6 SVM 50 a 500 vetores LSI + resolução + duração 70
4.7 SVM com 100 vetores + resolução + duração 70
4.8 Algoritmos de classificação com 100 vetores + resolução + duração 70
4.9 Regressão Logística com 100 vetores + resolução + duração 70
4.10 Algoritmos de classificação com 100 vetores + resolução + duração + exclamação 71
4.11 Regressão Logística com 100 vetores + resolução + duração + exclamação 71
4.12 Algoritmos de classificação com 100 vetores + resolução + duração + exclamação 72
4.13 Regressão Logística com 100 vetores + resolução + duração + justiça 72

Lista de Códigos de Programas

3.1 Algoritmo de Captura - Nível 1: Importação dos pacotes 29
3.2 Algoritmo de Captura - Nível 1: Conexão com o Sqlite 29
3.3 Algoritmo de Captura - Nível 1: Criação da Tabela link_l1 29
3.4 Algoritmo de Captura - Nível 1: Instância do Navegador 30
3.5 Algoritmo de Captura - Nível 1: Identificação e captura 30
3.6 Algoritmo de Captura - Nível 2: Carregamento dos Pacotes 32
3.7 Algoritmo de Captura - Nível 2: Conexão com o Sqlite 32
3.8 Algoritmo de Captura - Nível 2: Parse Datas e Localidade 33
3.9 Algoritmo de Captura - Nível 2: Fila 34
3.10 Algoritmo de Captura - Nível 2: Captura Parte 1/2 35
3.11 Algoritmo de Captura - Nível 2: Captura Parte 2/2 36
3.12 Algoritmo de Captura - Nível 3: Carregamento dos Pacotes 38
3.13 Algoritmo de Captura - Nível 3: Parse Datas 39
3.14 Algoritmo de Captura - Nível 3: Conexão ao Sqlite 39
3.15 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 1/3 40
3.16 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 2/3 41
3.17 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 3/3 42
3.18 Algoritmo de Captura - Nível 3: Fila 42
3.19 Algoritmo de Captura - Nível 3: Parse do html 42
3.20 Algoritmo de Captura - Nível 3: Função de Busca dos elementos XPATH 43
3.21 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 1/3 43
3.22 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 2/3 45
3.23 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 3/3 46
3.24 Algoritmo de Captura - Nível 3: Instanciamento do MongoDB 47
3.25 Algoritmo de Captura - Nível 3: Salvando no MongoDB 48
3.26 Algoritmo de Captura - Nível 3: Scraping nível 3 49
3.27 Correção Ortográfica: Carregamento dos Pacotes 50
3.28 Correção Ortográfica: Instanciamento do MongoDB 51
3.29 Correção Ortográfica: Consulta ao MongoDB 51
3.30 Correção Ortográfica: Tokenização das Reclamações 52
3.31 Correção Ortográfica: Verificação Ortográfica - Parte 1/3 53
3.32 Correção Ortográfica: Verificação Ortográfica - Parte 2/3 54

3.33 Correção Ortográfica: Verificação Ortográfica - Parte 2/3 56
3.34 LSI: Carregamento dos Pacotes 57
3.35 LSI: Carregamento das Stopwords 58
3.36 LSI: Criação do Dicionário 58
3.37 LSI: Construção do Bag-of-Words e Tf-idf 59
3.38 LSI: Construção da Matriz LSI 59

CAPÍTULO 1

Introdução

Com o desenvolvimento e disseminação na internet no Brasil, as relações comerciais entre consumidores e empresas sofreram transformações. Os comentários e reclamações que antes eram gerados e restritos a troca de informações entre os círculos de amizade, agora com o compartilhamento em redes sociais e sites especializados estes conteúdos extrapolam estes círculos, possibilitando consumidores obterem informações prévias sobre produtos e serviços para sua tomada de decisão de consumo.

As empresas preocupadas como sua reputação e imagem perante ao mercado buscam atender melhor seus clientes evitando assim denegrir sua imagem, o que em um mercado competitivo seria crucial para a sobrevivência do empreendimento.

Como avaliar individualmente cada comentário sobre produtos ou serviços de uma determinada empresa pode se tornar uma tarefa árdua e demorada, muitos consumidores recorrem a sites especializados para buscarem informações precisas sobre as empresas, com base em reclamações e elogios de outros consumidores.

Estas reclamações e elogios são armazenados e disponibilizados no sites especializados, permitindo potenciais consumidores avaliarem se vale a pena ou não consumir aquele produto ou serviço desejado, verificando se a reclamação é solucionada, o tempo médio de resposta da empresa e nota atribuída àquela reclamação.

Sites especializados como o ReclameAqui disponibilizam gratuitamente aos consumidores a possibilidade abrirem uma reclamação contra uma empresa, além de permitir acompanhar o andamento e resolução do problema aos demais usuários do site. O site gera e disponibiliza também rankings com as empresas mais reclamadas, as que mais solucionam os problemas dos clientes e outras estatísticas interessantes para a tomada de decisão do consumidor.

O objetivo deste trabalho, com base nas informações disponibilizadas no site

ReclameAqui, é criar um modelo de classificação, como base em algoritmos de aprendizagem de máquinas, com o objetivo de identificar padrões de o porquê os consumidores avaliam com notas baixas (insatisfação) ou notas altas (satisfação) as reclamações cadastradas.

Para isto serão utilizados métodos de captura de informações das reclamações do ReclameAqui, sendo este o site mais popular do país. O armazenamento destas informações se dará num banco de dados não relacional (NoSQL) para posterior tratamento das informações e construção de modelos de classificação.

Este modelo permitirá através do modelo de classificação elaborado, quais ações a empresa deve tomar com aquela reclamação, podendo optar por resolver num curto espaço de tempo ou simplesmente não resolvê-la, pois a satisfação do cliente não será alterada dependendo do tipo de reclamação que for realizada.

A análise de sentimentos dessas informações permitirá o consumidor a escolher melhor seus produtos e serviços e também permitirá as empresas melhorarem seus produtos e serviços oferecidos.

CAPÍTULO 2

Revisão Bibliográfica

2.1 Trabalhos Relacionados

A análise de sentimento é um método de classificação que tem como o objetivo avaliar a percepção e julgamento do avaliador sobre determinado tema ou conteúdo expressos de forma textual [13].

A análise de sentimento tem como objetivo determinar a atitude de um escritor com relação a algum tema ou a polaridade contextual geral de um documento. A atitude pode ser o seu julgamento ou avaliação, estado afetivo (ou seja , o estado emocional do autor quando a escrita) , ou a comunicação emocional pretendida (ou seja , o efeito emocional do autor deseja ter no leitor).

A análise de sentimento vem sendo estudada com o objetivo de identificar padrões em revisões e comentários em sites especializados na internet. As primeiras pesquisas sobre o tema datam o ano de 2002 nos artigos publicados por Turney[10] e Pang [8].

Nestas publicações, a análise de sentimento tinha como objetivo criar classificadores utilizando técnicas da área de processamento de linguagem natural (NLP) como variáveis ou *features* para explicar o porquê do conteúdo analisado ter uma avaliação positiva ou negativa.

No artigo [10], utiliza-se como base de dados informações obtidas do site Epinions criando um classificador automático dos comentários e avaliações de produtos. Implementando técnicas de processamento de linguagem natural, utilizou-se de adjetivos e advérbios contidos nos documentos para pontuar a polaridade, onde cada termo possui uma polaridade única, calculada através da orientação semântica do termo, sendo este o log na base 2 da razão das probabilidade de a frase ser "excelente"dado que ela é "ruim"sobre a frase ser "ruim"dado que ela é "excelente"[10] apud [1] . Ao final calcula-

se todas as polaridades positivas menos as polaridades negativas e se obtém a análise de sentimento do documento. Turney não propôs nenhum método de precisão ou verificação de sua classificação.

A identificação de padrão nas avaliações torna-se necessária a medida que o conteúdo disponível na internet cresce rapidamente, exigindo técnicas que resumam e classifiquem o conteúdo de forma automática e com um grau de precisão aceitável. Para verificar e comprovar que os humanos não identificam padrões de maneira assertiva, Pang [8] realiza um teste com dois de seus alunos da ciência da computação e solicita-os para elaborarem uma lista com palavras negativas e positivas para realizar uma comparação com sua avaliação de filmes, obtendo uma precisão de 58% e 64%. Comprova-se que humanos possuem determinada dificuldade em identificar padrões, e sugere-se utilizar modelos de aprendizagem de máquinas para realizar determinada tarefa. Propôs utilizar três algoritmos para gerar os modelos de classificação: naive-bayes, entropia máxima e máquinas de suporte de vetores (SVM). Enquanto, [10] unicamente se propôs a calcular a a polaridade total do documento não realizando predições.

Para realizar a classificação, Pang [8] utilizou técnicas de processamento de linguagem natural para extrair variáveis a serem utilizadas em seu modelo, empregando técnicas como: bag-of-words com unigramas e bigramas, além de part of speech ou parte do discurso (POS) , e a posição onde o termo se encontrava no documento, (exemplo: primeiro quarto do documento, último quarto do documento etc). Após a realização de tratamentos de pontuação, não utilizou-se de técnicas de stemming, processo de redução da flexão do termo, e remoção de stopwords, palavras sem significados. Fez uso em suas variáveis a utilização dos valores de sua frequência ou somente a presença da palavra do documento.

Em seu artigo Pang [8] utiliza a base de avaliações e comentários da IMDb http://www.reviews.imdb.com/Reviews/, selecionando 20 avaliações por autor com classificação, obtendo uma coleção de 752 avaliações negativas e 1301 positivas, excluindo da base as avaliações neutras, realizando uma classificação binária. Ao final o autor conclui que encontrou dificuldades em criar um classificador preciso para realizar determinada classificação.

A necessidade de criar classificadores torna-se cada vez mais indispensável, Yu [14] em seu artigo propõe separar fatos de opiniões de artigos publicados nos 6 principais jornais dos Estados Unidos, extraindo da base de dados TREC artigos publicados entre os anos de 1987 a 1992 . Em seu estudo selecionou 2.000 artigos de cada seção (Editorial,

Cartas ao Editor, Negócios e Notícias), classificando primeiramente os documentos entre fatos e opiniões, adotando e empregando um padrão ouro para verificação, garantindo a precisão do classificador. Após esta separação realizou a análise de sentimento nos documentos classificados como opiniões identificando a polaridade com base no algoritmo proposto por Turney[10], treinando um modelo naive-bayes com diferentes composições de features (unigramas, bigramas, trigramas, part-of-speech e polaridade) para realizar a classificação dos documentos. Obteve uma precisão de 91% na separação de fatos de opiniões, e 90% na classificação dos documentos entre positivo, negativo e neutro nas opiniões.

Em 2004, Beineke [2] utiliza um método supervisionado utilizando a base do RottenTomatoes , um site especializado em críticas de cinema americano. Beineke propõe que o sentimento possa estar contido numa única passagem do documento onde está expressa a opinião do autor. Sendo assim, utilizou de técnicas estatísticas como o Naive Bayes e Regressão Logística com regularização para realizar a classificação. Para suas variáveis, localizou os termos buscados, com emprego de stemming, no resumo ou no primeiro parágrafo. Ao final, não obteve resultados satisfatórios, propõe novas formas mais sofisticadas de resumir a crítica que possam melhorar a predição.

O problema de identificação de variáveis capazes de explicar a classificação dos documentos é cada vez mais evidente [2], [14], [8] e [6], as variáveis extraídas por técnicas de processamento de linguagem natural anteriormente não somente são suficientes para aumentar o poder de classificação e precisão dos documentos. Neste sentido, Pang [6] propõe uma nova técnica extraindo de forma subjetiva trechos do textos, empregando técnicas de corte mínimos de grafos selecionando os trechos mais importantes para a análise. Para realizar esta comparação, utilizando Naive Bayes e SVM juntamente com cross-validation de 10-partições, os modelos foram avaliados com o documento todo e com trechos extraídos do método de cortes mínimos. Pang obteve bons resultados comparativos e conseguindo atingir bons níveis de precisão, comprovando que não é necessário a utilização de todo documento para modelar um bom classificador.

A busca por melhores variáveis explicativas ou features para aprimorar os modelos de classificação se envereda para o campo de sistemas de recuperação de informações (Information Retrieval) [5], [4] e [13]. Em 2010, Paltoglou [5] utiliza a matriz tf-idf (term frequency - inverse document frequency ou termo inverso da frequência nos documentos) com pesos para suavizar os valores. Para realizar o experimento, utilizou-se da mesma base testada por Pang em 2002 [8], comparando os resultados da frequência do termos, com a tf-idf tradicional, e a tf-idf com pesos obteve excelentes resultados em seu modelo

de classificação (83,4%, 88,45% e 96,9% respectivamente).

Estes resultados expostos acima incentivaram Maas [4] utilizar outras técnicas de recuperação de informação para extrair features para o seu modelo de classificação, capazes agora de capturar componentes semânticos para o seu modelo através de métodos não supervisionados capturando anotações sentimentais do documento. Para realizar o proposto utilizou-se de Latent Semantic Analysis (LSA) que tem com o objetivo analisar o relacionamento de um conjunto de documentos e seus termos que são capazes de criar um conjunto de conceitos relacionados, ou seja, identificar categorias entre os documentos, além de reduzir a dimensionalidade do problema.

A análise de sentimentos não é somente aplicada a críticas de base de cinemas, mas também em avaliação de produtos [13] e também em sites especializados de avaliações de produtos e serviços [10], mostrando-se ser uma técnica capaz de sumarizar e classificar as informações com base nas opiniões apresentadas textualmente. Vale ressaltar, todos os trabalhos mencionados a análise de sentimentos buscou obter classificações binárias, positivo ou negativo, satisfeito ou insatisfeito, porém Pang em seu trabalho [7] propõe classificar em três categorias: negativo, neutro e positivo, não limitando a aplicação.

2.2 Conceitos e Fórmulas matemáticas

2.2.1 Tf-idf

Em sistemas de recuperação de informação, tf-idf (termo frequência e inverso da frequência do documento, é uma medida que pretende refletir a relevância de uma palavra num documento em uma coleção ou corpus [12]. Ela é frequentemente usada como um fator de ponderação na recuperação de informação e de mineração de texto. O valor da tf-idf aumenta proporcionalmente ao número de vezes que uma palavra aparece no documento, mas é compensado pela freqüência da palavra no corpus, o que ajuda a ajustar para o fato de que algumas palavras aparecem com mais frequência em geral.

Variações do sistema de ponderação tf-idf são muitas vezes utilizadas pelos motores de busca como uma ferramenta central na pontuação e classificação relevância de um documento a entregar uma consulta do usuário. A tf-idf pode ser utilizada com sucesso para palavras-parar de filtrar em vários campos, incluindo o resumo de texto e classificação.

Uma das mais simples funções de classificação é calculado pela soma dos tf-idf para cada termo da consulta; muitas funções de classificação mais sofisticados são

variantes deste modelo simples.

Termo Frequência é dado por:

$$\mathrm{tf}(t,d) = f_{t,d} \tag{2-1}$$

Onde o peso é dado pela frequência de ocorrência do termo no documento.

Enquanto o Inverso da Frequência do documento é:

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2-2}$$

Onde:

- N: Total de documentos no corpus $N = |D|$
- $|\{d \in D : t \in d\}|$: número de documentos onde o termo t ocorre (ex., $\mathrm{tf}(t,d) \neq 0$). Se o termo não estiver contido no corpus, isto implicará em uma divisão por zero. Por conseguinte, é comum ajustar o denominador para $1 + |\{d \in D : t \in d\}|$.

Logo a fórmula tf-idf é dada por:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D) \tag{2-3}$$

2.2.2 Latent Semantic Analysis (LSI)

O método de redução de dimensionalidade Latent Semantic Analysis, conhecida também como Latent Semantic Index (LSI) é calculada através da decomposição de valores singulares (SVD).

Dada uma matriz M mxn cuja valores são escalares k, neste caso uma matriz tf-idf, podemos fatorizá-la de M = U V*, onde U é uma matriz unitária mxm sobre k, a matriz é uma matriz diagonal mxn com números reais não negativos na diagonal, e V* uma matriz unitária nxn sobre k. denota a transposta conjugada de V.

O mapeamento entre o espaço, nas linhas (palavras) e nas colunas (contexto) . A baixa dimensionalidade do mapeamento captura o significado latente (oculto) nas palavras e dos contextos. Limitando-se o número de dimensões latentes (k < r) forçando uma maior correspondência entre as palavras e contextos. Esta correspondência forçada entre palavras e contextos melhora a similaridade auferida. [11] apud [3].

2.2.3 Regressão Logística

A regressão logística é um modelo linear utilizado para a classificação também conhecido como logit,. Neste modelo, as probabilidades que descrevem os possíveis resultados são calculadas de uma única vez, ou seja sem iterações.

Como é um problema de otimização, regressão logística penalizada pela norma L2 para uma classe binária minimiza a seguinte função de custos [9]:

$$\min_{w,c} \quad \frac{1}{2}w^T w + C\sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1) \tag{2-4}$$

2.2.4 Ridge Classifier

Ridge aborda alguns dos problemas dos Mínimos Quadrados Ordinários, impondo uma penalidade do tamanho de coeficientes. Os coeficientes do Ridge minimizam a penalização da soma residual dos quadrados,

$$\min_{w} ||Xw - y||_2{}^2 + \alpha||w||_2{}^2 \tag{2-5}$$

$$\tag{2-6}$$

Aqui , $\alpha \geq 0$ é um parâmetro complexo que controla a quantidade de encolhimento : quanto maior for o valor de α , maior será a quantidade de encolhimento e, portanto, os coeficientes de se tornar mais robusto a colinearidade.

2.2.5 Support Vector Machine - SVM

A máquina de suporte de vetores ou SVM pode ser utilizada para classificação (SVC) quanto para regressão (SVR).

A máquina de suporte de vetores contrói um hiperplano ou um conjunto de hiperplanos capazes gerar modelos de classificação e regressão. Para a máquina de suporte de vetores uma boa separação é constituída de uma maior margem , dada pela distância entre as observações (pontos de treino) e o hiperplano que separa a classe. Quanto maior esta margem, maior é poder de generalização do modelo [9].

Dados os vetores de treinamento $x_i \in \Re^P, i = 1, ..., n$, em duas classes, e um vetor $y \in \{1, -1\}^n$, SVC resolve o seguinte problema primal:

$$\begin{aligned} \min_{w,b,\zeta} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{l}\zeta_i \\ & y_i(w^T\phi(x_i)+b) \geq 1-\zeta_i, \\ & \zeta_i \geq 0, i = 1,\ldots,l. \end{aligned} \tag{2-7}$$

Sendo o dual:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{sujeito a} \quad & y^T\alpha = 0, \\ & 0 \leq \alpha_i \leq C, \qquad i = 1,\ldots,l. \end{aligned} \tag{2-8}$$

Onde e é um vetor unitário, C é a cota superior, Q é matriz nxn positiva semidefinida l by l matriz positiva semidefinida, $Q_{ij} \equiv y_i y_j K(x_i,x_j)$, onde $K(x_i,x_j) \equiv \phi(x_i)^T\phi(x_j)$ é o núcleo.

Onde a função de decisão é:

$$f(x) = sign(\sum_{i=1}^{l} y_i\alpha_i K(x_i,x)+b). \tag{2-9}$$

2.2.6 SGDClassifier - Gradiente Descendente Estocástico

Dado um conjunto de exemplos de treino $x_1,y_1),\ldots,(x_n,y_n)$, onde $x_i \in \mathbf{R}^n e y_i \in \{-1,1\}$, o nosso objetivo é aprender a função de pontuação linear $f(x) = w^T x + b$ com parâmetros $w \in \mathbf{R}^m$ e intercepta $b \in \mathbf{R}$. A fim de fazer previsões, nós simplesmente olhamos para o sinal de $f(x)$. A escolha comum para encontrar os parâmetros do modelo é, minimizando o erro de treinamento regularizada dada pela [9]:

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w) \tag{2-10}$$

Onde L é uma função perda que mede a aderência do modelo e R é um termo de regularização (penalidade aka) , que penaliza a complexidade do modelo; $\alpha > 0$ é um hiperparâmetro não negativo.

Diferentes alternativas para L implicam em diferentes classificadores:

Hinge : (soft-margin) Support Vector Machines.
Log : Regressão Logística.

Least-Squares : Regressão Ridge.
Epsilon-Insensitive : (soft-margin) Support Vector Regression.

Todas as funções de perda acima podem ser consideradas como uma cota superior para o erro de classificação.

Alternativas populares para a regularização do termo R:

Norma L2: $R(w) := \frac{1}{2}\sum_{i=1}^{n} w_i^2$,
Norma L1: $R(w) := \sum_{i=1}^{n} |w_i|$, que lida com soluções esparsas.
Elastic Net : $R(w) := \frac{\rho}{2}\sum_{i=1}^{n} w_i^2 + (1-\rho)\sum_{i=1}^{n} |w_i|$, uma combinação convexa das normas L2 e L1, onde ρ é dado por $1 - L1_ratio$.

O gradiente descendente estocástico é um método de otimização de problemas de otimização sem restrições . Em contraste com o (lote) gradiente descendente, SGD aproxima o verdadeiro gradiente de $E(w,b)$, considerando-se um único exemplo de formação de cada vez.

O método itera sobre os exemplos de treinamento e para cada um exemplo atualiza os parâmetros do modelo de acordo com a regra de atualização dada pela

$$w \leftarrow w - \eta(\alpha\frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i)}{\partial w}) \tag{2-11}$$

Onde η é a taxa de aprendizagem , que controla o passo de tamanho no espaço de parâmetros . O intercepto b é atualizado de forma semelhante , mas sem regularização.

A taxa de aprendizagem taxa η pode ser constante ou diminuindo gradualmente. Para a classificação, o cronograma da taxa de aprendizagem (learning rate =' ideal ') é dada por:

$$\eta^{(t)} = \frac{1}{\alpha(t_0 + t)} \tag{2-12}$$

Onde t é o intervalo de tempo , t_0 é determinado com base numa heurística proposta por Leon Bottou de tal modo que as alterações iniciais esperadas são comparáveis com o tamanho esperado dos pesos (isto assumindo que o norma das amostras de treinamento é de aprox. 1).

2.2.7 Cross-validation e GridSearchCV

Aprender os parâmetros de uma função de previsão e testá-lo nos mesmos dados é um erro metodológico[9]: um modelo que seria apenas repetir os rótulos das amostras que acaba de ver teria uma pontuação perfeita, mas não seria suficiente para prever qualquer coisa útil sobre dados ainda desconhecidos. Esta situação é denominada overfitting. Para evitarmos isso, é prática comum quando se realiza um experimento supervisionado de aprendizado de máquina assegura que os dados de treino e teste sejam separados.

Ao avaliar diferentes configurações ("parâmetros") para estimadores há um risco de overfitting no conjunto de teste porque os parâmetros podem ser ajustados ao estimador com objetivo de otimizá-los. Desta forma, o conhecimento sobre o conjunto de teste pode "contaminar"o modelo de avaliação e métricas perdendo o desempenho da generalização.

Para resolver este problema , utilizamos parte do conjunto de treino para validação, separando em k-partições, onde k-1 partições são treinadas e esta outra partição será o conjunto de validação do algoritmo de aprendizado de máquina, que será executado k vezes, e quando o experimento estiver bem sucedido, a avaliação final pode ser feito no conjunto de teste [9]. Este procedimento tem o nome de Cross-Validation (CV) ou validação cruzada. Abaixo está descrito o pseudo-algoritmo da validação cruzada

- Um modelo é treinado usando k-1 das partições como dados de treinamento
- O modelo resultante é validado na parte restante dos dados (isto é, é usado como um conjunto de teste para calcular uma medida de desempenho, tais como a precisão).

O método GridSearchCV é um processo de busca por melhores hiperparâmetros, além de realizar validação cruzada (CV) avalia os parâmetros que melhor estimam o modelo.

CAPÍTULO 3

Metodologia

3.1 Captura dos dados

Os dados foram coletados do sítio ReclameAqui utilizando técnicas de *web scraping* em três etapas e armazenados em dois bancos de dados, e que de forma resumida podemos descrevê-la desta forma:

1. A primeira etapa consistiu na obtenção da página inicial da empresa reclamada, selecionando todas as reclamações;
2. Após esta seleção a fase seguinte é coletar o endereço individual de cada reclamação, juntamente com a data de abertura do processo e o local da reclamação;
3. Na etapa final, iremos visitar cada reclamação e coletaremos os dados da reclamação.

O conceito de *web scraping* consite em extrair conteúdo de páginas públicas na internet e utilizá-las para outros fins.

Como iremos utilizar as informações contidas no site ReclameAqui para realizar nosso estudo. Como o site utiliza-se de proteções para evitar o uso destas técnicas para extração do conteúdo por robôs, com a verificação de interação humana através de CAPTCHA, acronônimo de "Completely Automated Public Turing test to tell Computers and Humans Apart".

Uma das ferramentas utilizadas para a interação com a aplicação web e extração de seu conteúdo é o Selenium. Esta ferramenta foi desenvolvida para testar e homologar aplicativos web de forma programática, dando ao usuário o poder de interagir com o navegador.

Como este processo de captura exige interação do usuário com o website, pois aleatoriamente o site verifica a presença de usuário humano através do preenchimento do CAPTCHA para validação. A utilização do selenium faz se necessária, pois permite

de forma programática simular um usuário humano, interagindo com a página quando necessário.

3.1.1 Coleta dos dados: Primeira Etapa

A primeira etapa do processo é identificar a primeira página das empresas reclamadas e armazená-las no banco de dados *sqlite* para posterior consulta. Para realizar o processo de captura foi utilizando o Python utilizando os pacotes Selenium e Sqlite3 com o objetivo de obter os endereços iniciais de cada empresa e armazená-las.

Os dados serão inicialmente armazenados no Sqlite3 pela facilidade de armazenar e realizar consultas, além de permitir evitar duplicidade através da utilização de chave primária, sendo um banco de dados relacional.

O algoritmo de execução deste procedimento será detalhado abaixo.

Código 3.1 Algoritmo de Captura - Nível 1: Importação dos pacotes

```
1 from selenium import webdriver
2 import sqlite3
```

Os pacotes Sqlite3 e o Selenium serão carregados para sua posterior utilização no processo de captura.

Código 3.2 Algoritmo de Captura - Nível 1: Conexão com o Sqlite

```
3 conn = sqlite3.connect("base.db")
4 cursor = conn.cursor()
```

Criamos uma conexão com o arquivo "base.db", caso ele não exista ele será criado. E atribuímos sua conexão a variável cursor.

Código 3.3 Algoritmo de Captura - Nível 1: Criação da Tabela link_l1

```
5 cursor.execute("""CREATE TABLE link_L1
6                   (empresa text, link text)
7                   """)
```

Após a criação da comunicação com o banco de dados através da variável cursor, criamos uma tabela link_l1 com os campos empresa e link, onde serão armazenados o nome da empresa e a página inicial da empresa respectivamente.

Código 3.4 Algoritmo de Captura - Nível 1: Instância do Navegador

```
8  driver = webdriver.Firefox()
9  pagina='http://www.reclameaqui.com.br/ranking/'
10 driver.get(pagina)
```

Para instanciarmos uma conexão programática com browser utilizaremos o Mozilla Firefox, o driver já está disponível no pacote selenium, gravaremos na variável driver a instância do driver Firefox.

Em seguida abriremos a primeira página através do comando driver.get(pagina), onde a variável página é a página do site www.reclameaqui.com.br/ranking onde estão listados todos os rankings disponíveis pelo site.

No nosso caso, iremos capturar as empresas com mais reclamações no últimos 12 meses.

Código 3.5 Algoritmo de Captura - Nível 1: Identificação e captura

```
11 def scrapRanking():
12     for i in range(20):
13         empresa = driver.find_element_by_xpath('//*[@id="
               tabela-ranking"]/tbody/tr[14]/td/div/table['+
               str(i+1)+']/tbody/tr/td/a').get_attribute("
               title")
14         link = driver.find_element_by_xpath('//*[@id="
               tabela-ranking"]/tbody/tr[14]/td/div/table['+
               str(i+1)+']/tbody/tr/td/a').get_attribute("href
               ")
15                 lista = [empresa,link]
16         cursor.execute("INSERT INTO link_L1 VALUES (?,?)",
               lista)
17         conn.commit()
18
19 scrapRanking() # executa o método
```

O método scrapRanking() localiza na página aberta pelo o navegador (www.reclameaqui.com.br/ranking) os elementos xpath's para armazenar na tabela link_l1. Através do elemento xpath utilizamos o texto, ou seja, o nome da empresa além do seu link, obtidos através da função get_attribute("title") e get_attribute("href"), respectivamente das 20 empresas listadas.

Os elementos xpath utilizam expressões de caminho para selecionar nós num documento xpath, é uma outra forma de representação da linguagem HTML através de árvore.

Para obter os elementos xpath de um objeto de um website, utilizando o navegador Google Chrome, clique com o botão direito sobre o objeto e selecione "Inspecionar Elemento", após selecione o elemento e clique novamente com o botão direito, selecione "Copiar"e depois "Copiar XPATH".

Armazenamos numa lista e depois passamos para o banco de dados sqlite3 através do comando cursor.execute e posteriormente conn.commit() concretizando a operação.

3.1.2 Coleta dos dados: Segunda Etapa

A segunda etapa do processo de captura é visitar a página de reclamações de cada empresa e por conseguinte coletar os links das reclamações, aproveitando também para coletar a localidade e o horário da reclamação. O código de captura será detalhado abaixo.

Os módulos do Selenium serão bastante úteis caso aconteçam erros esperados na execução do processo de captura, como por exemplo aguardar a página ser carregada corretamente através do WebDriverWait, utilizando o método implicity_wait(tempo) que será utilizado no decorrer do código, ou tratar alguma exceção ou até mesmo ter timeout da página.

Código 3.6 Algoritmo de Captura - Nível 2: Carregamento dos Pacotes

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
import selenium.webdriver.support.expected_conditions as
    EC
from selenium.common.exceptions import TimeoutException
import sqlite3
import os
import re
import time
import requests
import random
import string
import Image
import pytesseract
from PIL import Image
from datetime import datetime
```

O primeiro passo do processo de coleta dos links das reclamações das empresas é carregar os pacotes utilizados.

Código 3.7 Algoritmo de Captura - Nível 2: Conexão com o Sqlite

```
conn = sqlite3.connect("base.db")
cursor = conn.cursor()
cursor.execute("CREATE TABLE reclamacoes(business text,
    link text UNIQUE, place text, data text)")
```

Nesta etapa conectamos ao banco de dados e criamos a tabela para receber os links das reclamações que serão capturados.

Código 3.8 Algoritmo de Captura - Nível 2: Parse Datas e Localidade

```
20 meses = [u'Janeiro', u'Fevereiro', u'Março', u'Abril', u'
     Maio', u'Junho', u'Julho', u'Agosto', u'Setembro', u'
     Outubro', u'Novembro', u'Dezembro']
21
22 def setDate(txt):
23     try:
24         for i in range(12):
25             txt = txt.replace(meses[i],u'0'+str(i+1))
26             txt = txt.replace(u'de', '-')
27             match = re.search(r'\d{2} - \d{2} - \d{4} - \d
                 {2}:\d{2}', txt)
28         return match
29     except:
30         print 'Is not a String object'
31
32 def getDate(arg):
33     match = setDate(arg.split(',')[1].strip())
34     date = time.strptime(match.group().replace(' ',''), '%
         d-%m-%Y-%H:%M')
35     return date
36
37 def getPlace(arg):
38     arg = arg.split(',')[2].strip()
39
40     return arg
```

A próxima etapa é criar funções que traduzam o texto encontrado e converter em formato data e a localidade da reclamação para facilitar o interpretação do código no processo de captura. Transformando a string encontrada em data e local da reclamação, em funções específicas.

Código 3.9 Algoritmo de Captura - Nível 2: Fila

```
41 from collections import deque
42 lista_url = deque(5*[''],5)
43
44 def grava_url(url):
45     lista_url.appendleft(url)
46
47 def show_url():
48     for i in lista_url:
49         print i
```

Em geral, no processo de captura sempre ocorrem erros inesperados como falha de conexão da internet local, e até mesmo o próprio servidor do site fica fora do ar por alguns instantes, ou por um longo período de manutenção. Nestes casos, utilizaremos uma fila para armazenar os últimos endereços visitados, sendo possível retornar aos últimos endereços visitados mesmo após uma falha que interrompa a execução do código.

A variável lista_url armazena os últimos cinco endereços visitados, descartando sempre o primeiro que entra.

Código 3.10 Algoritmo de Captura - Nível 2: Captura Parte 1/2

```
50 def breakCaptha(wd):
51     try:
52         wd.save_screenshot('./captcha/screenshot.png')
53         im1 = Image.open('./captcha/screenshot.png')
54         im1.crop((50, 310, 165, 350)).save('./captcha/
               screenshot2.png')
55         im = Image.open('./captcha/screenshot2.png')
56         captcha = pytesseract.image_to_string(im, config="
               -psm 6")
57         wd.find_element_by_name('captcha').send_keys(
               captcha)
58         wd.find_element_by_xpath('//*[@id="reclamacoes-
               empresa"]/div/div[1]/div[3]/ul/div/div/form/
               input[2]').click()
59         time.sleep(1)
60     except:
61         print "Didn't broke the captcha"
62
63 def Start_Scrap_L2(business, driver):
64     try:
65         for j in range(20):
66             link = driver.find_element_by_xpath('//*[@id="
                   reclamacoes-empresa"]/div/div[1]/div[3]/ul/
                   li[' + str(j+1) + ']/h3/a').get_attribute("
                   href")
67             text = driver.find_element_by_xpath('//*[@id="
                   reclamacoes-empresa"]/div/div[1]/div[3]/ul/
                   li[' + str(j+1) + ']/span[2]').text
68             lugar = getPlace(text)
69             data = str(datetime(*getDate(text)[:6]))
70             sql2 = 'INSERT INTO reclamacoes VALUES ("' +
                   business + '" , "' + link + '" , "' + lugar
                    + '" , "' + data + '")'
71             try:
72                 cursor.execute(sql2)
73             except sqlite3.IntegrityError as err:
74                 pass
75             conn.commit()
76     except:
77         breakCaptha(driver)
```

Código 3.11 Algoritmo de Captura - Nível 2: Captura Parte 2/2

```
79 def Start_Scrap(i, n_pag):
80     # n_pag  número de páginas a serem visitadas
81     sql = "SELECT * FROM link_L1 WHERE rowid ="  + str(i)
82     cursor.execute(sql)
83     lista = cursor.fetchone()
84     grava_url(lista[1])
85     continua = lista_url[0]
86     driver = webdriver.Firefox()
87     driver.get(continua)     # Continua  o link salvo na
           lista_url[0]
88     driver.implicitly_wait(15)
89     for i in range(n_pag):
90         # Apaga os cookies a cada 100 páginas visitadas
91         if (i % 100) == 0:
92             driver.delete_all_cookies()
93         try:
94             Start_Scrap_L2(lista[0],driver)
95             elem = driver.find_element_by_link_text("
                   próximo »")
96             grava_url(elem.get_attribute("href"))
97             elem.click()
98             time.sleep(5+random.randint(0,5))
99         except:
100            try:
101                some_object = WebDriverWait(driver, 250).
                       until(EC.element_to_be_located((By.
                       XPATH, '//*[@id="reclamacoes-empresa"]/
                       div/div[1]/div[3]/ul/li[1]/h3/a')))
102            except TimeoutException:
103                print lista_url[0]
104
105
106 Start_Scrap(1, 6500)
```

Estas três funções são as principais do código. Quando o código é executado, a função Start_Scrap() inicia o processo de captura, buscando na tabela link_L1 a empresa que será coletada. Em seguida o navegador Firefox é iniciado e recebe a ordem de abrir a primeira página da empresa, programaticamente o navegador aguarda implicitamente 15

segundos para a página ser carregada completamente.

O número de páginas a serem visitadas é um argumento da função Start_Scrap(empresa, número de páginas), a ser definido pelo usuário, assim como a empresa a ser buscada. Por opção, a cada 100 páginas visitadas o navegador apaga os cookies armazenados.

Quando o carregamento da página se conclui, o algoritmo chama o método Start_Scrap_L2() que tem o objetivo inicial de identificar na página carregada os elementos buscados, ou seja, o link da reclamação, o texto que contêm a data e local da reclamação e gravá-las na tabela reclamações. Caso ele não encontre os elementos buscados, ele entra na exceção no método isto porque podemos estar sendo solicitados pela página inserir o código de verificação, Captcha, para validar a presença humana no processo de interação com a página ou é um robô.

Para resolvermos este problema, utilizaremos os pacotes pytesseract e o PIL. Este último com auxílio do selenium tirará uma foto da página visitada e cortará a imagem especificamente no local aonde aparecerá o código em forma de imagem a ser digitado. Para o reconhecimento destes dígitos o primeiro pacote identificará através de OCR (Optical Character Recognition) e retornará os dígitos em uma variável que será enviada para o navegador preencher o campo destinado. E assim, desta forma, podemos continuar coletando as reclamações, contornando este problema.

Após o armazenamento dos dados no banco de dados, o algoritmo avança a página sequencialmente até coletar todos os links das reclamações.

Importante ressaltar a importância entre o tempo de permanência em cada página, se colocarmos o robô para coletar muito rápido o servidor da página identifica a presença de robô e bloqueia o seu acesso. Para evitar isto, é prudente utilizar tempos randômicos superiores a 5 segundos.

3.1.3 Coleta dos dados: Terceira Etapa

Na última etapa do processo de captura do conteúdo da reclamação utilizaremos uma outra abordagem, pois o site não tem nenhum mecanismo de verificação e validação humana. Com os links de cada reclamação iremos baixar o código da página html através do pacote urllib3, identificando as informações necessárias, tratá-las e armazená-las no

banco de dados.

Com esta abordagem aceleramos muito o processo de captura, realizando testes com a conexão de 10 mega utilizando o pacote urllib3 foram baixados 5 mil páginas por hora, enquanto utilizando o selenium 400 páginas por hora. Este fato deve a vários fatores, um deles é que o selenium necessita processar todo código html e apresentá-la ao usuário para realizar a operação necessária. Outro fato crucial é que quando o selenium não encontra o elemento buscado no código ele demora a tratar aquele erro, atrasando o processamento do código.

Com a utlização do urllib3 todo conteúdo da página é carregado na máquina e pré-processado com a utilização lxml.etree, e transformada em uma árvore permitindo buscas pelos elementos XPATH, facilitando e acelerando o processo de busca dos elementos da página.

As informações serão armazenadas no MongoDB, um banco de dados voltado ao armazenamento de documentos, permitindo que cada documento possua campos diferentes e específicos sem comprometer o desempenho do banco de dados.

O código desta terceira etapa do processo está descrita abaixo:

Código 3.12 Algoritmo de Captura - Nível 3: Carregamento dos Pacotes

```
1 import os
2 import time
3 from datetime import datetime
4 from time import mktime
5 import random
6 import re
7 import collections
8 import sqlite3
9 import pymongo
10 import urllib3
11 from lxml import etree
```

A primeiro procedimento do algoritmo é carregar os pacotes que serão utilizados no decorrer do código.

Código 3.13 Algoritmo de Captura - Nível 3: Parse Datas

```
12 inverted_index = collections.defaultdict(set)
13 inverted_index[u'Janeiro'].add('01')
14 inverted_index[u'Fevereiro'].add('02')
15 inverted_index[u'Março'].add('03')
16 inverted_index[u'Abril'].add('04')
17 inverted_index[u'Maio'].add('05')
18 inverted_index[u'Junho'].add('06')
19 inverted_index[u'Julho'].add('07')
20 inverted_index[u'Agosto'].add('08')
21 inverted_index[u'Setembro'].add('09')
22 inverted_index[u'Outubro'].add('10')
23 inverted_index[u'Novembro'].add('11')
24 inverted_index[u'Dezembro'].add('12')
25
26 def getDate(txt):
27     txt = txt.split(',')[1].strip()
28     txt = txt.replace(u'de', '-')
29     txt = txt.replace(' ', '')
30     txt2 = ''.join(txt.split('-')[1])
31     txt = txt.replace(txt2,''.join(inverted_index[txt2]))
32     if len(txt) > 10:
33         date = time.strptime(txt, '%d-%m-%Y-%H:%M')
34     elif len(txt) == 10:
35         date = time.strptime(txt, '%d-%m-%Y')
36     return date
```

A próxima fase do algoritmo é definir funções para tratar e colocar no padrão necessário para armazenarmos no banco de dados.

Código 3.14 Algoritmo de Captura - Nível 3: Conexão ao Sqlite

```
37 connsql = sqlite3.connect("base.db")
38 cursor = connsql.cursor()
```

Abro uma conexão com o bando de dados sqlite que contém os links das reclamações.

Código 3.15 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 1/3

```
39 def set_titulo(txt):
40     global titulo_varg
41     titulo_varg = txt
42
43 def get_titulo():
44     return titulo_varg
45
46 def set_reclamacao_texto(txt):
47     global reclamacao_texto_varg
48     reclamacao_texto_varg = txt
49
50 def get_reclamacao_texto():
51     return reclamacao_texto_varg
52
53 def set_reclamacao_desativada(txt):
54     global reclamacao_desativada_varg
55     reclamacao_desativada_varg = txt
56
57 def get_reclamacao_desativada():
58     return reclamacao_desativada_varg
59
60
61 def set_resposta_empresa_data(txt):
62     global resposta_empresa_data_varg
63     resposta_empresa_data_varg = datetime.fromtimestamp(
           mktime(getDate(txt)))
64
65 def get_resposta_empresa_data():
66     return resposta_empresa_data_varg
67
68 def set_resposta_empresa_texto(txt):
69     global resposta_empresa_texto_varg
70     resposta_empresa_texto_varg = txt
71
72 def get_resposta_empresa_texto():
73     return resposta_empresa_texto_varg
74
75 def set_replica_data(txt):
76     global replica_data_varg
77     replica_data_varg = datetime.fromtimestamp(mktime(
           getDate(txt)))
```

Código 3.16 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 2/3

```
78 def get_replica_data():
79     return replica_data_varg
80
81 def set_replica_texto(txt):
82     global replica_texto_varg
83     replica_texto_varg = txt
84
85 def get_replica_texto():
86     return replica_texto_varg
87
88 def set_consideracao_final_data(txt):
89     global consideracao_final_data_varg
90     consideracao_final_data_varg = datetime.fromtimestamp(
           mktime(getDate(txt)))
91
92 def get_consideracao_final_data():
93     return consideracao_final_data_varg
94
95 def set_consideracao_final_texto(txt):
96     global consideracao_final_texto_varg
97     consideracao_final_texto_varg = txt
98
99 def get_consideracao_final_texto():
100     return consideracao_final_texto_varg
101
102 def set_consideracao_final_imagem(txt):
103     global consideracao_final_imagem_varg
104     consideracao_final_imagem_varg = txt[0]
105
106 def get_consideracao_final_imagem():
107     return consideracao_final_imagem_varg
108
109 def set_consideracao_final_voltaria_fazer_negocio(txt):
110     global consideracao_final_voltaria_fazer_negocio_varg
111     consideracao_final_voltaria_fazer_negocio_varg = txt
112
113 def get_consideracao_final_voltaria_fazer_negocio():
114     return consideracao_final_voltaria_fazer_negocio_varg
```

Código 3.17 Algoritmo de Captura - Nível 3: Definição de variáveis globais - Parte 3/3

```
116     global consideracao_final_nota_varg
117     consideracao_final_nota_varg = txt
118
119 def get_consideracao_final_nota():
120     return consideracao_final_nota_varg
```

Para realizar o tratamento de cada informação proveniente da página html, utilizaremos funções específicas e armazenaremos em variáveis globais permitindo assim o uso destas em outros métodos e funções do código. Deste modo facilitaremos o processo de tratamento das informações para salvá-las no banco de dados.

Código 3.18 Algoritmo de Captura - Nível 3: Fila

```
121 from collections import deque
122 lista_rowid = deque(5*[''],5)
123
124 def grava_rowid(rowid):
125     lista_rowid.appendleft(rowid)
126
127 def show_rowid():
128     for i in lista_rowid:
129         print i
```

Novamente utilizaremos um método de fila para armazenar os últimos links visitados, caso aconteça algum imprevisto ou erro poderemos retornar.

Código 3.19 Algoritmo de Captura - Nível 3: Parse do html

```
130 def set_etreehtml(htmldados):
131     global arvorehtml_varg
132     arvorehtml_varg = etree.HTML(htmldados)
133
134 def get_etreehtml():
135     return arvorehtml_varg
```

Nesta etapa, como mencionada acima ocorre o tratamento da conteúdo da página html, transformando e organizando os seus elementos em formato de árvore, facilitando a busca dos elementos XPATH.

Código 3.20 Algoritmo de Captura - Nível 3: Função de Busca dos elementos XPATH

```
136 def find_information(xpath2):
137     html = get_etreehtml()
138     filtered_html = html.xpath(xpath2)
139     return ''.join(t.strip() for t in filtered_html)
```

Esta função recebe o caminho XPATH a ser localizada na árvore html e retorna a informação.

Código 3.21 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 1/3

```
140 def get_information(conteudo):
141     # criar uma lista para passar o status
142     global status
143     status = [0, 0, 0, 0]
144
145     txt_t0 = find_information('//*[@id="div00"]/div[4]/div
            /div/div/div/div/text()')
146     txt_t1 = find_information('//*[@id="div00"]/div[4]/div
            /div/div/div/div/p/text()'')
147
148     if (txt_t0 == '' and txt_t1 == ''):
149         txt2 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[3]/div[1]/div
                [1]/blockquote/h1/b/text()') #titulo
150         txt3 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[3]/div[1]/div
                [2]/text()') # reclamacao
151         set_titulo(txt2)
152         set_reclamacao_texto(txt3)
153         status[0] = 1
154     else:
155         set_reclamacao_desativada(u'Reclamação desativada
                ou inexistente ou congelada.')
156         status[0] = 0
```

A função get_information ao receber a página html que contêm os dados, inicializa uma lista status com o objetivo de marcar naquele conteúdo quais os objetos estão

presentes na página, ou seja, se a empresa respondeu aquela reclamação, se houve réplica do consumidor e se a reclamação foi avaliada. Com esta marcação facilita o processo de gravação no mongodb, salvando somente o conteúdo disponível na reclamação evitando assim campos vazios.

O primeiro passo desta função é coletar a reclamação realizada, ou seja, o título da reclamação e a reclamação propriamente dita. Entre o processo de captura do links descritos na etapa 2 e este processo, algumas reclamações foram desativadas e neste caso elas foram gravadas com a descrição deste status.

Após o tratamento da reclamação, o algoritmo verifica a existência de resposta da empresa, capturando a data de resposta e o conteúdo da reposta.

Código 3.22 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 2/3

```
157     if status[0] == 1:
158
159         # Resposta da empresa
160         txt = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[4]/div[1]/span/
                text()') # data
161         txt2 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[4]/div[2]/text()
                ') # texto
162
163         if  txt != '':
164             set_resposta_empresa_data(txt)
165             set_resposta_empresa_texto(txt2)
166             status[1] = 1
167         else:
168             status[1] = 0
169
170         #Replica do Consumidor
171
172
173         txt_t0 = find_information('//*[@id="replicas"]/div
                /div[1]/h4/text()')
174
175         if txt_t0 == u'Réplica do Consumidor':
176
177             txt = find_information('//*[@id="replicas"]/
                    div/div[1]/span/text()')
178             txt2 = find_information('//*[@id="replicas"]/
                    div/div[2]/text()')
179             set_replica_data(txt)
180             set_replica_texto(txt2)
181
182             status[2] = 1
183
184         else:
185             status[2] = 0
```

Caso o consumidor não esteja satisfeito com as providências tomadas pela

empresa, ele pode atribuir uma réplica a reclamação dando continuidade a reclamação. Estas informações também foram capturadas, tratadas e armazenadas no banco de dados.

Código 3.23 Algoritmo de Captura - Nível 3: Busca Informação - Parte: 3/3

```
186         w = 4+sum(status[1:3])
187         pattern = 'http://www.reclameaqui.com.br/images/
                img.([\w]+).png'
188
189         txt1 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[' + str(w) +']/
                div[1]/span/text()')
190         txt2 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[' + str(w) +']/
                div[2]/text()')
191         txt3 = re.findall(pattern,conteudo)
192         txt4 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[' + str(w) +']/
                div[3]/div[2]/div[1]/b/span/text()')
193         txt5 = find_information('//*[@id="div00"]/div[4]/
                div/div/div/div/div[1]/div/div[' + str(w) +']/
                div[3]/div[2]/div[2]/p/b/text()')
194
195         if (txt1 != ''):
196
197             set_consideracao_final_data(txt1)
198             set_consideracao_final_texto(txt2)
199             set_consideracao_final_imagem(txt3)
200             set_consideracao_final_voltaria_fazer_negocio(
                    txt4)
201             set_consideracao_final_nota(txt5)
202
203             status[3] = 1
204         else:
205             status[3] = 0
206
207     return status
```

Após todo o processo de reclamação o consumidor avalia a reclamação, atribuindo uma nota a empresa, informando se seu problema foi resolvido e se voltaria a

fazer negócio com a empresa.

Todas essas informações após serem localizadas na árvore html gerada pela função set_etreehtml, são armazenadas nas variáveis globais descritas acima. A função get_information retorna somente o status do conteúdo da reclamação. No processo seguinte iremos abordar o processo de gravação do banco de dados MongoDB.

Código 3.24 Algoritmo de Captura - Nível 3: Instanciamento do MongoDB

```
208 try:
209     con=pymongo.MongoClient()
210     print "Connected successfully!!!"
211 except pymongo.errors.ConnectionFailure, e:
212    print "Could not connect to MongoDB: %s" % e
213 con
214
215 db = con.reclameaquidb
216 collection = db.ReclameAquiBase
```

O MongoDB é instanciado no algoritmo, abrindo uma conexão com o banco de dados, criando uma coleção denominada ReclameAquiBase onde serão armazenados todos os documentos (reclamações).

Código 3.25 Algoritmo de Captura - Nível 3: Salvando no MongoDB

```
217 def Mongo_save(status, parse_lista, rowid):
218     # Empty dictionary for storing related data
219     data ={}
220     data['empresa'] = parse_lista[0]
221     data['cod_sql'] = rowid
222     data['url'] = parse_lista[1]
223     data['localidade'] = parse_lista[2]
224     data['data_captura'] = datetime.now()
225     data['reclamacao_data'] = datetime.fromtimestamp(
            mktime(time.strptime(parse_lista[3], '%Y-%m-%d %H:%
            M:%S')))
226
227     if status[0] == 1:
228         data['reclamacao_titulo'] = get_titulo()
229         data['reclamacao_texto'] = get_reclamacao_texto()
230     elif status[0] == 0:
231         data['reclamacao_bloqueada'] =
                get_reclamacao_desativada()
232
233     if status[1] == 1:
234         data['resposta_empresa_data'] =
                get_resposta_empresa_data()
235         data['resposta_empresa_texto'] =
                get_resposta_empresa_texto()
236
237     if status[2] == 1:
238         data['replica_data'] = get_replica_data()
239         data['replica_texto'] = get_replica_texto()
240
241     if status[3] == 1:
242         data['consideracao_final_data']=
                get_consideracao_final_data()
243         data['consideracao_final_texto']=
                get_consideracao_final_texto()
244         data['consideracao_final_imagem']=
                get_consideracao_final_imagem()
245         data['consideracao_final_voltaria_fazer_negocio']=
                get_consideracao_final_voltaria_fazer_negocio()
246         data['consideracao_final_nota']=
                get_consideracao_final_nota()
247
248     collection.insert(data)
249
```

Após tratar todo o conteúdo da página html, extraindo todas as informações disponíveis a método Mongo_Save recebe o status da reclamação descrito acima, o nome da empresa reclamada e o código do link da reclamação do Sqlite.

Gravando num dicionário todas as informações encontradas de acordo com seus status, puxando os valores das variáveis globais, finalizando o processo de gravação através do comando collection.insert(data).

Código 3.26 Algoritmo de Captura - Nível 3: Scraping nível 3

```
251 def Start_Scrap(n1,n2):
252     for i in range(n1,n2):
253         if (i % 5000) == 0:
254             time.sleep(60*10)
255
256         sql = "SELECT * FROM reclamacoes WHERE rowid = " +
                str(i)
257         cursor.execute(sql)
258         lista = cursor.fetchone()
259         grava_rowid(i)
260
261         #time.sleep(2)
262         r = http.request('GET', lista[1])
263         set_etreehtml(r.data)
264
265         status = get_information(r.data)
266         Mongo_save(status, lista, i)
267
268
269 http = urllib3.PoolManager()
270
271 Start_Scrap(n1,n2)    # n1: primeiro id a ser capturado,
```

O método Start_Scrap inicializa todo o processo de captura, extração, tratamento e gravação no MongoDB, sendo alimentada pelos links armazenadas no Sqlite.

3.2 Tratamento dos Dados

3.2.1 Correção Ortográfica

Com o objetivo de homogeneizar e corrigir a ortografia das palavras capturadas nas reclamações no site do ReclameAqui, iremos implementar um algoritmo de correção utilizando principalmente o pacote enchant com a utilização do dicionário português brasileiro.

O algoritmo de correção ortográfica, será demonstrado abaixo.

Código 3.27 Correção Ortográfica: Carregamento dos Pacotes

```
import os
import time
from datetime import datetime
from time import mktime
import random
import re
import collections
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
import pymongo
from nltk.tokenize import WordPunctTokenizer
from nltk.corpus import stopwords
import enchant
```

Novamente iremos carregar os pacotes que serão utilizados no algoritmo. Podemos ressaltar os pacotes nltk que será utilizado no processamento do texto e o pacote enchant que será utilizado para realizar a correção das palavras que apresentam erros ortográficos.

Código 3.28 Correção Ortográfica: Instanciamento do MongoDB

```
15 try:
16     conn=pymongo.MongoClient()
17     print "Connected successfully!!!"
18 except pymongo.errors.ConnectionFailure, e:
19    print "Could not connect to MongoDB: %s" % e
20 conn
21
22 db = conn.reclameaquidb
23 collection = db.ReclameAquiBase
24 collection
```

Instanciaremos o banco de dados MongoDB que contêm as reclamações capturadas.

Código 3.29 Correção Ortográfica: Consulta ao MongoDB

```
25 reclamacao = pd.DataFrame(list(collection.find({u'
       consideracao_final_imagem':{'$exists': True }},{ u'_id'
       :1, u'cod_sql':1, u'reclamacao_texto':1 })))
26
27 listareclamacao = reclamacao.reclamacao_texto.tolist()
```

Nesta etapa iremos realizar uma consulta ao MongoDB utilizando o pacote pandas para transformar o resultado da consulta num dataframe (tabela).

Após finalizar a consulta, gravaremos os textos das reclamações numa lista para realizarmos as operações necessárias.

Código 3.30 Correção Ortográfica: Tokenização das Reclamações

```
28 sw = stopwords.words('portuguese')
29
30 textos_limpos = []
31 for texto in listareclamacao:
32     tlimpo = [token.lower() for token in
           WordPunctTokenizer().tokenize(texto) if token not
           in sw]
33     textos_limpos.append(tlimpo)
34
35 lista_tokens = []
36 for listas in textos_limpos:
37     lista_tokens += listas
38
39 len(lista_tokens) --> 59.063.409
40
41 vocabulario = set(lista_tokens)
42 len(vocabulario) --> 636.343
43
44 from collections import Counter
45 counts = Counter(lista_tokens)
46 counts
47
48 frequenciatokens = pd.DataFrame.from_dict(counts, orient='
       index').reset_index()
49 frequenciatokens = frequenciatokens.rename(columns={'index
       ': 'token',0:'frequencia'})
50 frequenciatokens = frequenciatokens.sort(['frequencia'],
       ascending=[0])
```

Primeiramente iremos carregar a lista de *stopwords* para a língua portuguesa, disponível no pacote nltk. Iremos separar cada reclamação individualmente todas as palavras (tokens) com exceção das stopwords.

Após essa separação iremos criar uma lista de todas as palavras utilizadas em todas reclamações e calcular suas frequências. As reclamações contêm 59.063.409 palavras e 636.343 palavras diferentes (vocabulário).

Ao calcular a frequência de cada token, voltaremos com estas informações para o pandas e ordenaremos decrescentemente.

Código 3.31 Correção Ortográfica: Verificação Ortográfica - Parte 1/3

```
enchant.list_languages()
d = enchant.Dict("pt_BR")

# a comparação com o dicionário para verificar se a palavra está certa ou errada do pyenchant
frequenciatokens['correto'] = frequenciatokens['token'].apply(lambda x: d.check(x))

frequenciatokens.sort(['frequencia'],ascending=[0])
basecorrigida = frequenciatokens[frequenciatokens['correto'] == False]

basecorrigida = basecorrigida.reset_index()

# O objetivo foi exportar para o excel, e analisar quais tokens representam 80% dos erros
basecorrigida.to_csv('basecorrigida_pareto.csv', sep='\t', encoding='utf-8')
```

Para realizarmos a verificação ortográfica, criaremos uma coluna dentro do dataframe criado no pandas contendo o resultado da verificação da comparação do dicionário do enchant, e filtraremos somente aqueles tokens que não estão corretos ortograficamente de acordo com o enchant.

Foram encontrados 3.921.580 tokens ortograficamente incorretos, realizando uma análise de pareto se considerássemos corrigir todos os tokens com frequência superior a 50, estaríamos corrigindo 81,78% dos erros encontrados. Teríamos que corrigir aproximadamente 4 mil tokens.

Código 3.32 Correção Ortográfica: Verificação Ortográfica - Parte 2/3

```
palavra = []
palavracorrigida = []


for i in range(0,3000): # total de 326912

    x = basecorrigida.iloc[i].token
    try:
        palavra.append(int(x))
        palavracorrigida.append(int(x))
    except:
        sugestao = d.suggest(x)
        print '-->  ' + x
        print 'Escolha as opcoes?'

        for i,s in enumerate(sugestao):
            print str(i) +' - '+s,

        print ''
        try:
            y = str(input())
        except:
            y = x

        try:
            y = int(y)
            p = sugestao[y]
        except:
            p = y

        palavra.append(x)
        palavracorrigida.append(p)


tabela = pd.DataFrame(palavracorrigida, index=palavra)
tabela.to_pickle('dicionario_de_correcao.pickle')

# vou eliminar oq não foi corrigido
tabela.rename(columns={'index':'antes',0:'depois'},
    inplace=True)
```

Para corrigir estes 4 mil tokens, implementamos um algoritmo novamente com o auxílio do enchant que dá sugestões de correção, que poderíamos aceitar fornecendo a opção correta, ou caso nenhuma das sugestões seja escolhida poderíamos manualmente incluir a correção. Gerando ao final 2 listas contendo as informações antes e depois da correção que serão transformadas em um dataframe do pandas.

Código 3.33 Correção Ortográfica: Verificação Ortográfica - Parte 2/3

```
tabela['igual'] = tabela.apply(lambda x : True if (x[u'antes'] == x[u'depois']) else False, axis=1)

tabela = tabela[tabela['igual'] == False]
tabela = tabela[['antes','depois']]
tabela = tabela.set_index(['antes'])
substituicao = tabela.to_dict()

reclamacao.reclamacao_texto = reclamacao['reclamacao_texto'].apply(lambda x: x.lower())

# processo de substituição

i = 0
t = datetime.now()
for p1, p2 in substituicao.iteritems():

    p1 = u'\\b'+p1+'\\b'
    reclamacao['reclamacao_texto'].replace(p1 , p2, inplace=True ,regex=True)

    i +=1
    if i%100==0:
        t2 = datetime.now()
        print i, t2 - t
        t = t2


reclamacao.rename(columns={'reclamacao_texto':'reclamacao_corrigida'}, inplace=True)

reclamacao2 = pd.read_pickle('basereclamacaocorrigida.pickle')
```

Para evitar correções desnecessárias iremos excluir da nossa lista de substituições os tokens que consideramos corretos e não precisariam ser corrigidos.

Utilizando expressões regulares iremos realizar a substituição de todos os tokens que consideramos incorretos pela sua correção no primeiro dataframe carregado do banco de dados. E iremos salvá-lo num formato encapsulado denominado pickle para posterior utilização.

3.2.2 Construção da Matriz LSI

Para a construção da Matriz LSI, utilizaremos a base corrigida anteriormente e o pacote gensim para eliminar a palavras que aparecem somente uma vez e construir a matriz.
A construção desta matriz tem dois objetivos, reduzir a dimensionalidade da variáveis e capturar semelhança dos documentos.

Código 3.34 `LSI: Carregamento dos Pacotes`

```
import pandas as pd
import numpy as np
from time import time

from nltk.tokenize import WordPunctTokenizer
from nltk.corpus import stopwords
import string
import nltk

from gensim import corpora, models, similarities
```

Primeiramente iremos capturar os pacotes que serão utilizados para a construção da matriz.

Código 3.35 LSI: Carregamento das Stopwords

```
11 sw = stopwords.words('portuguese') + list(string.
     punctuation)
12 sw.remove(u'nÃ£o')
13 lista_remover = [u'00', u'000', u'01', u'02', u'03',
     u'04', u'05', u'06', u'07', u'08', u'09', u'10',
      u'100', u'11', u'12', u'13', u'14', u'15', u'16'
     , u'17', u'18', u'19', u'1Âº', u'20', u'200', u'
     2012', u'2013', u'2014', u'2015', u'21', u'22', u
     '23', u'24', u'25', u'26', u'27', u'28', u'29',
     u'30', u'31', u'32', u'33', u'34', u'35', u'36',
      u'37', u'38', u'39', u'3g', u'40', u'41', u'42',
       u'43', u'44', u'45', u'46', u'47', u'48', u'49'
     , u'50', u'51', u'52', u'53', u'54', u'55', u'56
     ', u'57', u'58', u'59', u'60', u'69', u'70', u'
     72', u'79', u'80', u'89', u'90', u'99', u'120', u
     '2011', u'300', u'4004']
14 sw += lista_remover
```

Iremos carregar a lista de stopwords e pontuações e incluiremos uma lista de dígitos que serão eliminados para o processamento da matriz LSI.

Código 3.36 LSI: Criação do Dicionário

```
15 t0 = time()
16 dictionary = corpora.Dictionary(textos_limpos)
17 once_ids = [tokenid for tokenid, docfreq in dictionary.dfs
     .iteritems() if docfreq == 1]
18 dictionary.filter_tokens(once_ids) # remove stop words and
      words that appear only once
19 dictionary.compactify() # remove gaps in id sequence after
      words that were removed
20 print("Completo em %fm" % ((time() - t0)/60.))
21 print(dictionary)
22
23 dictionary.save('./matrizes_full/corpora.dict') # store
     the dictionary, for future reference
```

Para gerar a matriz LSI construiremos um dicionário com todos os tokens presentes nos documentos e eliminaremos as stopwords e o tokens com frequência igual a um. E após

salvaremos o dicionário, caso precisemos posteriormente.

Código 3.37 LSI: Construção do Bag-of-Words e Tf-idf

```
corpus = [dictionary.doc2bow(text) for text in textos_limpos]
# Salvo o corpus no formato Matrix Market Format
corpora.MmCorpus.serialize('./matrizes_full/corpus.mm', corpus) # store to disk, for later use

# Rodo a matrix tf-idf
tfidf = models.TfidfModel(corpus) # step 1 -- initialize a model
# Salvo o modelo da matrix tf-idf
tfidf.save('./matrizes_full/tfidf_model')

corpus_tfidf = tfidf[corpus]
# Salvo a matrix tf-idf
corpora.MmCorpus.serialize('./matrizes_full/_tfidf.mm', tfidf[corpus])#, progress_cnt=10000)
```

Nesta etapa iremos construir a matriz bag-of-words e tf-idf com base no dicionário criado, ou seja, só serão utilizadas as palavras que estiverem listadas neste dicionário.

Código 3.38 LSI: Construção da Matriz LSI

```
lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=500) # initialize an LSI transformation
corpus_lsi = lsi[corpus_tfidf] # create a double wrapper over the original corpus: bow->tfidf->fold-in-lsi

lsi.save('./matrizes_full/lsi_model')
corpora.MmCorpus.serialize('./matrizes_full/_lsi_500.mm', lsi[corpus_tfidf])#, progress_cnt=10000)
```

Por fim, iremos gerar a matriz LSI com 500 vetores (ou tópicos), com base na matriz tf-idf gerada na etapa anterior e gravaremos para utilizarmos nos algoritmos de aprendizagem de máquinas.

CAPÍTULO 4

Pesquisa

4.1 Pesquisa

O site ReclameAqui é um site gratuito onde os consumidores podem realizar reclamações e elogios as empresas prestadoras de bens e serviços. O serviço do site é oferecido através da internet e pode ser consultado por qualquer interessado, que disponibiliza o conteúdo das reclamações, localização, data de registro e término, a resolução do problema, e a avaliação do consumidor, com exceção de dados com informações pessoais.

Para a escolha das empresas que iremos pesquisar, optou-se pelas empresas com mais reclamações nos últimos 12 meses do ranking de setembro de 2015. Todas as reclamações disponíveis destas empresas foram obtidas, e somente foram aproveitadas as reclamações que foram finalizadas, ou seja, o consumidor atribuiu uma nota a reclamação. O processo de captura coletou 1.092.163 reclamações, onde somente 424.318 reclamações foram finalizadas devidamente.

4.2 Análise Exploratória dos Dados

Na tabela 4.1, consta a lista de todas as reclamações por empresas que foram consideradas:

A amplitude do número de reclamações obtidas por empresa é bem ampla, a empresa mais reclamada possui aproximadamente 81 mil reclamações enquanto a empresa com menos reclamação finalizada possui 4 reclamações, ocasionando um desbalanceamento no número de observações por empresa a ser analisado.

Ao analisarmos descritivamente as notas dadas as reclamações, na Tabela 4.2, às empresas reclamadas possuem notas médias em torno de 4 e 6 pontos, com desvio-padrão

de 3 pontos, portanto estas avaliações são bem dispersas.

Podemos verificar na Figura 4.1 o violinplot das notas por empresa, e observamos que as notas estão dispersas, e existe pequenas áreas de concentração nas extremidades, entre notas compreendidas entre 0 e 1 e 8 a 10, e com pouca concentração na nota 5.

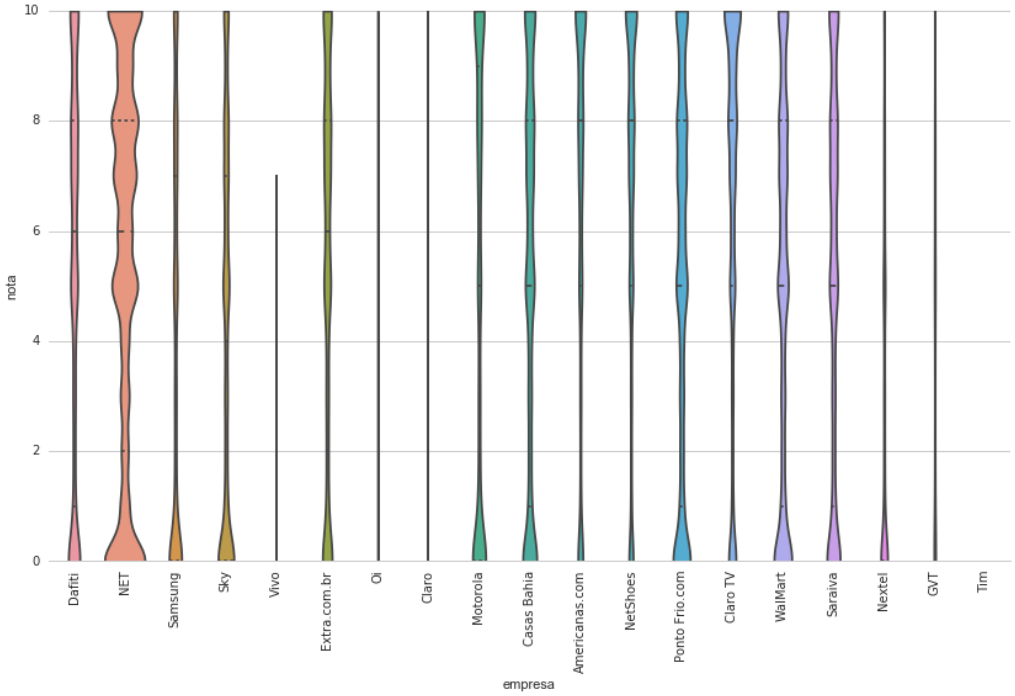

Figura 4.1: *ViolinPlot: Distribuição das Notas por empresa*

Se analisarmos as notas atribuídas pelo consumidor em conjunto com a variável se o problema foi resolvido ou não, Gráfico 4.2, podemos analisar que na maioria dos casos quando o problema é resolvido o consumidor atribui notas mais altas, porém existe casos onde o consumidor atribui notas baixas mesmo tendo seu problema solucionado. Entretanto quando o problema não é resolvido o consumidor sempre atribui nota baixa.

Analisando o tempo total da reclamação no Gráfico 4.3, ou seja, a diferença entre a abertura da reclamação e a finalização da mesma pelo consumidor. Para concentramos nossa análise iremos transformar as horas em dias e truncá-las em 365 dias (1 ano).
Desta forma:

$$duracao = \begin{cases} x = 365, & \text{se} \quad x \geq 365, \\ x = x, & \text{se} \quad x \leq 365. \end{cases}$$

Depois concentraremos o tempo:

Tabela 4.1: *Quantidade de Reclamações por empresas*

Empresa	Reclamações
NET	81.298
WalMart	35.737
Ponto Frio.com	34.453
Claro TV	32.817
Sky	31.752
Casas Bahia	27.950
Saraiva	26.439
Motorola	26.378
Samsung	25.151
Dafiti	22.995
NetShoes	22.556
Americanas.com	20.867
Extra.com.br	19.214
Nextel	13.390
GVT	3.249
Oi	36
Claro	21
Vivo	11
Tim	4
Total	424.218

Tabela 4.2: *Média e Desvio-padrão das Notas por empresas*

	Notas	
Empresas	Média	Desvio-padrão
NetShoes	6,785290	3,276277
Claro TV	6,640613	3,430419
Americanas.com	6,525902	3,547252
NET	5,397365	3,555880
Extra.com.br	5,374623	3,528822
Dafiti	5,183605	3,586242
Casas Bahia	5,158891	3,579579
Ponto Frio.com	5,071750	3,528987
Saraiva	5,069594	3,560676
Motorola	4,914285	4,100156
Claro	4,857143	3,539572
WalMart	4,772589	3,536115
Sky	3,971781	3,566414
Samsung	3,849231	3,684573
GVT	3,783318	3,339358
Oi	3,583333	3,341300
Nextel	2,735848	3,147585
Vivo	1,727273	3,003029
Tim	0,000000	0,000000

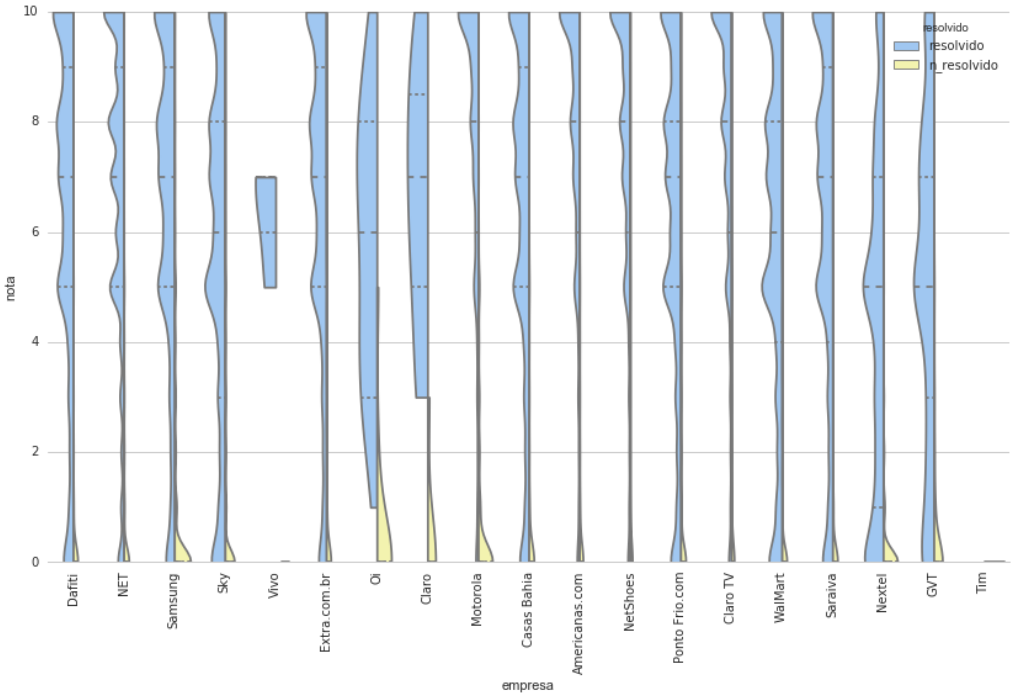

Figura 4.2: *ViolinPlot: Distribuição das Notas e Resolução da reclamação por empresa*

$$Tempo_ajustado = \frac{maximo - Tempo}{maximo - minimo}$$

Onde o máximo será 365 dias e o mínimo o menor tempo registrado.

Assim teremos que os tempos próximos a 1 serão as reclamações que foram finalizadas com menor tempo, e com duração a 0 (zero) as reclamações que demoraram mais tempo para serem finalizadas.

Podemos verificar na Figura 4.3, que o tempo de finalização das reclamações estão concentradas entre 0.8 e 1.0, para ambos os casos de resolução da reclamação independente da nota atribuída pelo consumidor.

Comparando o comportamento das notas com o estado gerador da reclamação , Figura 4.4, podemos concluir graficamente que não existe diferença nas avaliações de diferentes regiões do Brasil. Cabe ressaltar que os estados: RJ, SP, DF e PR os consumidores avaliam com maior frequência as reclamações com notas 5, em relação aos demais estados.

Utilizando técnicas de processamento de linguagem natural, podemos verificar

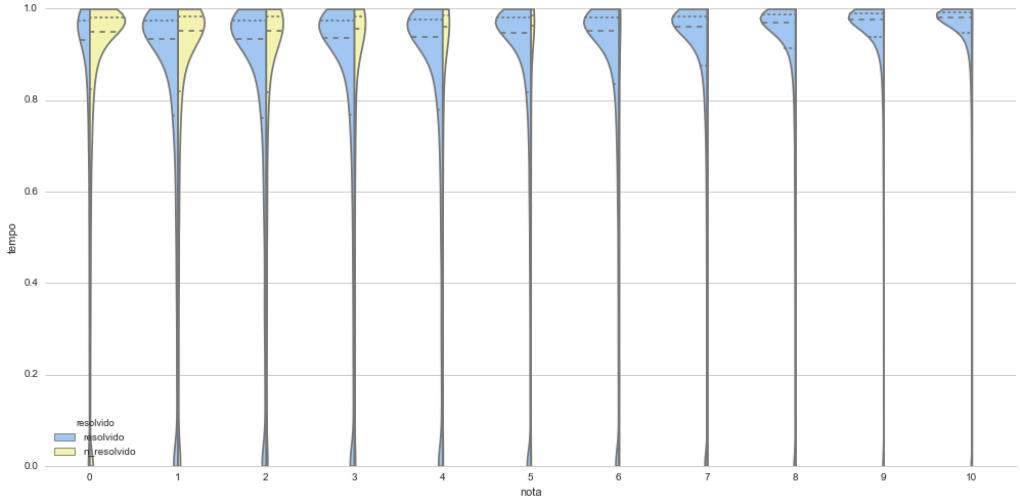

Figura 4.3: *ViolinPlot: Distribuição do Tempo e Resolução da reclamação por nota*

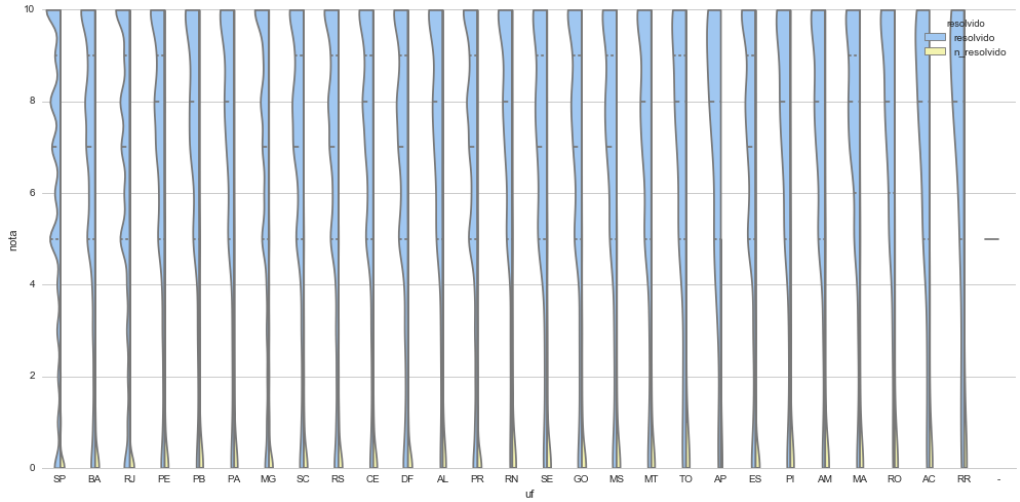

Figura 4.4: *ViolinPlot: Distribuição da Nota e Resolução da reclamação por Estado*

que muitas das reclamações contém exclamações, que poderia indicar um sentimento de irritação ou insatisfação do consumidor registrada em sua escrita na reclamação. Ao analisarmos o Gráfico 4.5, podemos constatar que grande parte das reclamações com notas entre 0 e 4 que possuem exclamações.

Para realizar a análise, algumas reclamações possuíam mais de 100 pontos de exclamações, para concentrar os valores sem perder o propósito da análise o número de exclamações foi truncado em 10 unidades.

Outra técnica proveniente é a contagem de tokens (palavras) contidas nas reclamações na Figura 4.6. O comportamento de insatisfação poderia ser acompanhado por um excesso de palavras expressando a indignação pelo meio textual. Como observado na

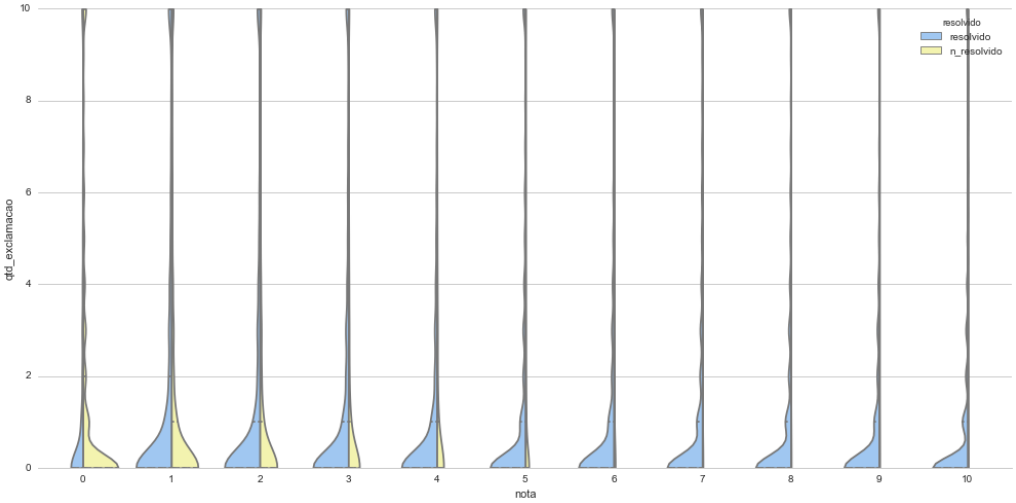

Figura 4.5: *ViolinPlot: Distribuição Exclamações e Resolução da reclamação por nota*

Figura4.6 não existe um padrão muito bem definido o porquê o consumidor atribui notas altas ou baixas devido ao número de palavras escritas na reclamação.

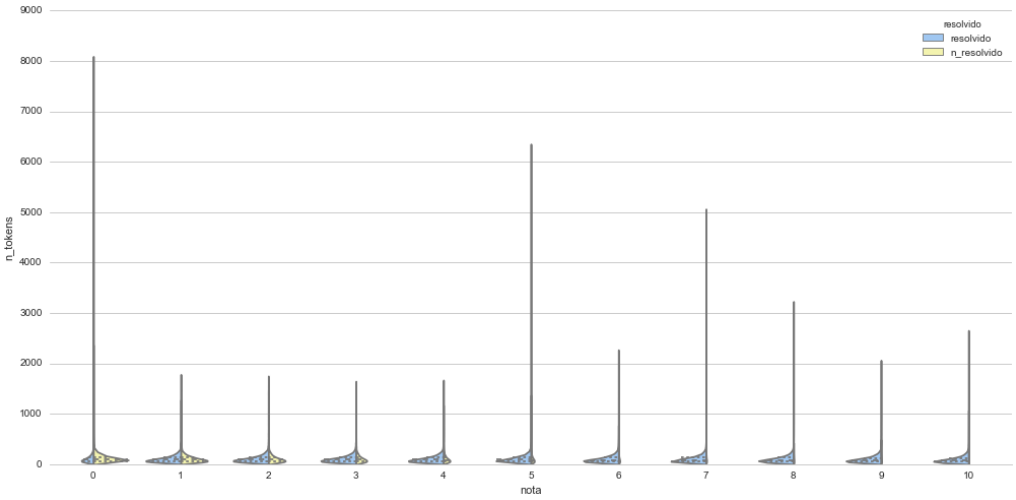

Figura 4.6: *ViolinPlot: Distribuição de Tokens e Resolução da reclamação por nota*

Verificando a capacidade de escrita do consumidor poderíamos considerar que estes consumidores poderiam atribuir notas medianas ou baixas a reclamação, fato que novamente não é comprovado graficamente na Figura 4.7.

Dada uma suposição que o consumidor quando insatisfeito pode ameaçar a empresa reclamada a procurar meios legais para a resolução de seus problemas para ter seu problema resolvido, ao buscarmos pelas palavras justiça e suas similaridades semânticas, com a utilização do algoritmo Word2vec, encontramos os seguintes tokens

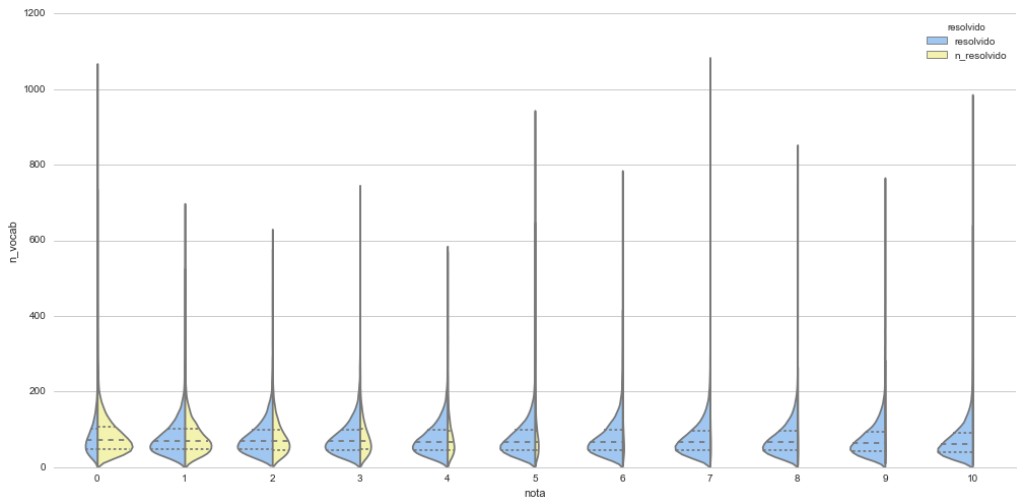

Figura 4.7: *ViolinPlot: Distribuição do Vocabulário e Resolução da reclamação por nota*

com similaridades dos cosenos superior a 0.70: jec, procon, procom, advogado. E ao comparamos a frequência em conjunto destes tokens das reclamações, podemos verificar que em grande parte das reclamações este termos não são escritos, e não possuem qualquer relação com as notas atribuídas no final da reclamação 4.8.

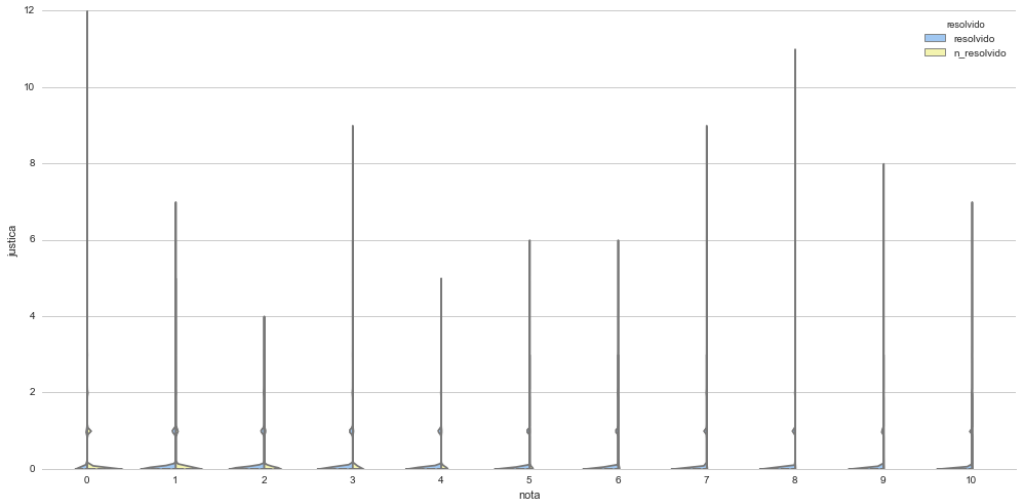

Figura 4.8: *ViolinPlot: Distribuição do Termo Justiça e Resolução da reclamação por nota*

4.3 Definição das Classes

Para agruparmos as notas em duas classes, denominaremos insatisfeito notas entre 0 a 3, e satisfeito as notas compreendidas entre 7 a 10. As notas intermediárias, neutras, compreendidas entre 4 a 6 não serão utilizadas no modelo de classificação. Iremos criar um modelo de classificação com o objetivo de identificar consumidores que

estão satisfeitos e insatisfeitos.

A Tabela 4.3 demonstra a quantidade de reclamações separadas pela definição das classes citadas acima.

Podemos verificar que um modelo dummie é capaz de prever com 56,16% de precisão, isto porque as classes encontram-se um pouco desbalanceadas.

Tabela 4.3: *Agrupamento das Notas por empresa*

Empresa	Nível de Satisfação			Total
	Insatisfeito	Neutro	Satisfeito	
NET	25.077	18.429	37.792	81.298
WalMart	13.366	8.799	13.572	35.737
Ponto Frio.com	11.721	8.393	14.339	34.453
Claro TV	6.586	5.557	20.674	32.817
Sky	15.175	7.011	9.566	31.752
Casas Bahia	9.427	6.471	12.052	27.950
Saraiva	9.145	6.209	11.085	26.439
Motorola	11.173	3.440	11.765	26.378
Samsung	12.748	4.725	7.678	25.151
Dafiti	7.603	5.307	10.085	22.995
NetShoes	3.962	4.060	14.534	22.556
Americanas.com	4.611	3.363	12.893	20.867
Extra.com.br	5.964	4.406	8.844	19.214
Nextel	8.424	2.867	2.099	13.390
GVT	1.572	853	824	3.249
Oi	20	6	10	36
Claro	9	4	8	21
Vivo	8	1	2	11
Tim	4	-	-	4
Total	146.595	89.901	187.822	424.318

4.4 Modelos de Classificação

Para a realização dos modelos de classificação utilizaremos o pacote scikit-learn [9] do Python, especializado em algoritmos de aprendizagem de máquinas.

Para criar os modelos de classificação utilizaremos cross-validation com 10-partições, e para garantir a proporcionalidade de cada classe em cada partição faremos uso da função *Stratifiedkfold*. Para estimar o melhor modelo para o mesmo algoritmo de classificação e as mesmas variáveis, será usado o GridSearchCV, que tem como objetivo

encontrar os melhores parâmetros para realizar a classificação. Vale ressaltar que os modelos gerados utilizarão sempre a mesma semente aleatória, permitindo assim reproduzir e comparar os modelos. Os parâmetros utilizados em cada algoritmo de aprendizagem de máquinas estão descritos na Tabela 4.4.

Em todos os modelos as variáveis de treinos e testes são separadas. As variáveis de treino representam 60% da população analisada.

Tabela 4.4: *Parâmetros utilizados no GridSearchCV por algoritmo de aprendizagem de máquinas*

Parâmetro	SVM	Regressão Logística	RidgeClassifier	SGDClassifier
Random_State	[50]	[50]	[100]	[50]
Regularização	[0.01, 0.05, 0.1]	[1, 2, 5, 10, 50, 100, 200, 500 ,1000]	[0.0001, 0.001, 0.01, 0.1, 1, 5, 10]	[0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001]
Solver	-	['sag']	['auto']	-
Kernel	['linear']	-	-	-
Intercepto	-	[True, False]	[True, False]	[True, False]
Penalty	- -	['l2']	-	['l2', 'l1', 'elasticnet']
Loss	-	-	-	['hinge', 'log', 'squared_hinge', 'perceptron', 'huber' 'squared_loss']
Learning_rate	-	-	-	['optimal']
Average	-	-	-	[True, False]
n_iter	-	-	-	[10]
Shrinking	[True]	-	-	-

4.4.1 Primeiro Modelo

O primeiro modelo de classificação utilizará a matriz LSI com os 500 vetores mais o tempo de duração da reclamação, o algoritmo utilizado será o RidgeClassifier.

Analisando o relatório de classificação, concluímos que este modelo apresenta uma precisão de 0,6543, conforme Tabela 4.5, ou seja, modelo se mostra mais preciso que o algoritmo dummy. Porém podemos perceber que o algoritmo não consegue ajustar o modelo somente com estas variáveis, apesar de empregar 500 vetores classificados semânticamente semelhantes.

4.4.2 Segundo Modelo

Alguns autores afirmam que o número ideal para aplicação da matriz LSI em um modelo de classificação seria algo em torno de 200 a 500 vetores, iremos aplicar o mesmo algoritmo de classificação, neste caso o SVC (SVM para classificação), para uma lista ,[50, 100, 150, 200, 250, 300, 400, 500], vetores LSI juntamente com a variável que representa se a reclamação foi resolvido e o tempo de duração da reclamação, conforme demonstrado na Tabela 4.6.

Podemos verificar empiricamente para este conjunto de dados que o incremento na precisão do modelo é bem pequeno em relação ao número de variáveis incluídas no modelo. Ou seja, o aumento de variáveis no modelo não gera o benefício de aumentar significativamente a precisão do modelo. Por este motivo, iremos assumir e utilizar 100 vetores LSI para os próximos modelos.

Analisando os resultados obtidos para 100 vetores LSI. o relatório de classificação, o modelo de classificação consegue atingir uma boa precisão, conforme Tabela 4.7, atingindo aproximadamente 0,82%. Ao analisarmos o desempenho pelos resultados obtidos, verificamos que o algoritmo não consegue determinar com exatidão quais os consumidores estão insatisfeitos, uma vez que ele retorna poucas reclamações com insatisfação, entretanto obtém uma precisão de 0.98 dentre as observações retornadas.

4.4.3 Terceiro Modelo

Agora empregando o algoritmo de Regressão Logística, Ridgeclassifier e SGDClassifier, com 100 vetores LSI , tempo de duração da reclamação e se o problema foi resolvido ou não. Os resultados dos modelos de classificação tem melhorias atingindo uma precisão de 0,8227 para a Regressão Logística, conforme Tabela 4.8.

Conforme tabela 4.8 que apresenta os resultados para os algoritmos de aprendizagem de máquinas, iremos analisar os relatório de classificação do melhor modelo de classificação, ou seja, a Regressão Logística na Tabela 4.9.

Podemos verificar que o algoritmo continua não encontrando um padrão para separar os consumidores insatisfeitos dos satisfeitos, não conseguindo retornar as observações que apresentam insatisfação, com 0.65 de revocação.

Tabela 4.5: *Ridge com 500 vetores + duração*

Classe	Precisão	Revocação	fl-score	Suporte
Insatisfeito	0,64	0,48	0.55	58.438
Satisfeito	0.66	0.79	0.72	75.329
Média / Total	0,65	0,65	0,64	133.767

Tabela 4.6: *SVM 50 a 500 vetores LSI + resolução + duração*

Vetores LSI	Precisão
50	0,819738
100	0,819738
200	0,819761
250	0.820209
300	0,820336
400	0,820493
500	0,820523

Tabela 4.7: *SVM com 100 vetores + resolução + duração*

Classe	Precisão	Revocação	fl-score	Suporte
Insatisfeito	0,98	0,60	0.74	58.438
Satisfeito	0,76	0,99	0,86	75.329
Média / Total	0,86	0,82	0,81	133.767

Tabela 4.8: *Algoritmos de classificação com 100 vetores + resolução + duração*

Algortimo de Classificação	Precisão
Regressão Logística	0,8227
RidgeClassifier	0,8217
SGDClassifier	0,8225

Tabela 4.9: *Regressão Logística com 100 vetores + resolução + duração*

Classe	Precisão	Revocação	fl-score	Suporte
Insatisfeito	0,92	0,65	0.76	58.438
Satisfeito	0,78	0,96	0,86	75.329
Média / Total	0,84	0,82	0,82	133.767

4.4.4 Quarto Modelo

Iremos adicionar outra variável ao nosso modelo, empregando o mesmo algoritmos utilizados anteriormente, incluindo a quantidade de pontos de exclamações. Com objetivo de identifcar que o consumidor insatisfeito poderia lançar uso dos pontos de exclamação para expressar seu sentimento. Os resultados dos modelos de classificação podem ser observados na Tabela 4.10, onde novamente a Regressão Logística tem um resultado superior aos demais algoritmos de classificação.

Tabela 4.10: *Algoritmos de classificação com 100 vetores + resolução + duração + exclamação*

Algortimo de Classificação	Precisão
Regressão Logística	0,8229
RidgeClassifier	0,8218
SGDClassifier	0,8223

Podemos analisar o relatório de classificação da Regressão Logística 4.11, utilizando 100 vetores LSI, o tempo de duração da reclamação, se o problema foi resolvido ou não e a quantidade de pontos de exclamação utilizados pelo consumidor, obtivemos os mesmos resultado descritos no modelo de classificação anterior.

Tabela 4.11: *Regressão Logística com 100 vetores + resolução + duração + exclamação*

Classe	Precisão	Revocação	fl-score	Suporte
Insatisfeito	0,92	0,65	0.76	58.438
Satisfeito	0,78	0,96	0,86	75.329
Média / Total	0,84	0,82	0,82	133.767

4.4.5 Quinto Modelo

Partindo da premissa que o consumidor insatisfeito poderia ameaçar a empresa reclamada com a possibilidade de entrar na justiça ou serviços de proteção ao consumidor, e por consequência atribuiria uma nota baixa ao final da reclamação será avaliada no próximo modelo. Iremos remover do nosso modelo a variável exclamação e adicionaremos a variável justiça que será representada pela presença ou não do termo dentro do conteúdo da reclamação.

Os modelos de classificação aprendizagem de máquinas novamente a Regressão Logística obteve desempenho superior aos demais algoritmos, entretanto o incremento ao

modelo foi pouco significante. O modelo de Regressão Linear obteve 0,8228 de precisão, como apresentado na tabela 4.12 e o relatório de classificação na tabela 4.13 que não apresenta nenhuma alteração aos modelos testados anteriormente.

Tabela 4.12: *Algoritmos de classificação com 100 vetores + resolução + duração + exclamação*

Algortimo de Classificação	Precisão
Regressão Logística	0,8228
RidgeClassifier	0,8218
SGDClassifier	0,8225

Tabela 4.13: *Regressão Logística com 100 vetores + resolução + duração + justiça*

Classe	Precisão	Revocação	fl-score	Suporte
Insatisfeito	0,92	0,65	0.76	58.438
Satisfeito	0,78	0,96	0,86	75.329
Média / Total	0,84	0,82	0,82	133.767

Para as variáveis de frequência de palavras e a quantidade de vocabulário utilizado na reclamação não apresentaram resultados expressivos.

CAPÍTULO 5

Conclusão

5.1 Conclusão

A análise de sentimentos em reclamações com base em informações extraídas do maior site de reclamações do Brasil, ReclameAqui, é possível através de aplicação de técnicas de processamento de linguagem de natural e aprendizagem de máquinas, além de outras variáveis disponíveis no portal.

A identificação de padrões no conteúdo escrito, através dos marcadores semânticos das reclamações extraídas pela técnica LSI (latent semantic index), permite reduzir a dimensionalidade do problema e capturar a essência da reclamação. Estes dados juntamente com a variável de resolução do problema e o tempo de duração da reclamação conseguem prever com uma boa precisão a avaliação final do consumidor.

O algoritmo de regressão logística obteve um desempenho superior em todos os modelos gerados, comprovando a importância de sua aplicação.

A melhoria no desempenho do algoritmo de classificação se dá através da escolha das variáveis, não só pelo o algoritmo de aprendizagem de máquinas empregado. Para futuros estudos seria o interessante capturar outras variáveis que estão disponíveis no portal (junho de 2016) do Reclameaqui que na época de elaboração desta dissertação não estavam disponíveis.

Este estudo é válido pois comprova que é possível criar um algoritmo de classificação com base em informações de reclamações de reais, com a aplicação de técnicas de processamento de linguagem natural, mineração de dados e aprendizagem de máquinas.

Referências Bibliográficas

[1] AGRESTI, A. **An introduction to categorical data analysis. new york: J**. *J Wiley and Sons*, 1996.

[2] BEINEKE, P.; HASTIE, T.; MANNING, C.; VAITHYANATHAN, S. **Exploring sentiment summarization**. In: *Proceedings of the AAAI spring symposium on exploring attitude and affect in text: theories and applications*, volume 39, 2004.

[3] DEERWESTER, S.; DUMAIS, S. T.; FURNAS, G. W.; LANDAUER, T. K.; HARSHMAN, R. **Indexing by latent semantic analysis**. *Journal of the American society for information science*, 41(6):391, 1990.

[4] MAAS, A. L.; DALY, R. E.; PHAM, P. T.; HUANG, D.; NG, A. Y.; POTTS, C. **Learning word vectors for sentiment analysis**. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, p. 142–150. Association for Computational Linguistics, 2011.

[5] PALTOGLOU, G.; THELWALL, M. **A study of information retrieval weighting schemes for sentiment analysis**. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, p. 1386–1395. Association for Computational Linguistics, 2010.

[6] PANG, B.; LEE, L. **A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts**. In: *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, p. 271. Association for Computational Linguistics, 2004.

[7] PANG, B.; LEE, L. **Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales**. In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, p. 115–124. Association for Computational Linguistics, 2005.

[8] PANG, B.; LEE, L.; VAITHYANATHAN, S. **Thumbs up?: sentiment classification using machine learning techniques**. In: *Proceedings of the ACL-02 conference on*

Empirical methods in natural language processing-Volume 10, p. 79–86. Association for Computational Linguistics, 2002.

[9] PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. **Scikit-learn: Machine learning in Python**. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] TURNEY, P. D. **Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews**. In: *Proceedings of the 40th annual meeting on association for computational linguistics*, p. 417–424. Association for Computational Linguistics, 2002.

[11] TURNEY, P. D.; PANTEL, P.; OTHERS. **From frequency to meaning: Vector space models of semantics**. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

[12] ULLMAN, J. D.; LESKOVEC, J.; RAJARAMAN, A. **Mining of massive datasets**, 2011.

[13] WANG, Y.; LI, Z.; LIU, J.; HE, Z.; HUANG, Y.; LI, D. **Word vector modeling for sentiment analysis of product reviews**. In: *Natural Language Processing and Chinese Computing*, p. 168–180. Springer, 2014.

[14] YU, H.; HATZIVASSILOGLOU, V. **Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences**. In: *Proceedings of the 2003 conference on Empirical methods in natural language processing*, p. 129–136. Association for Computational Linguistics, 2003.