UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

INSTITUTO DE MATEMÁTICA

CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

THAIS LUCA MARQUES DE ALMEIDA

ESTUDO SOBRE APLICAÇÃO DE APRENDIZADO DE MÁQUINA PARA IDENTIFICAÇÃO DE ASSALTOS ATRAVÉS DE INFORMAÇÕES DO TWITTER

RIO DE JANEIRO

2019

THAIS LUCA MARQUES DE ALMEIDA

ESTUDO SOBRE APLICAÇÃO DE APRENDIZADO DE MÁQUINA PARA IDENTIFICAÇÃO DE ASSALTOS ATRAVÉS DE INFORMAÇÕES DO TWITTER

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. João Carlos Pereira da Silva, D.Sc.

RIO DE JANEIRO

2019

CIP - Catalogação na Publicação

A447e

Almeida, Thais Luca Marques de
Estudo sobre Aplicação de Aprendizado de Máquina para Identificação de Assaltos através de Informações do Twitter / Thais Luca Marques de Almeida. -- Rio de Janeiro, 2018.
92 f.

Orientador: João Carlos Pereira da Silva.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Classificação. 2. Twitter. 3. Processamento de Linguagem Natural. 4. Aprendizado de Máquina. I. Silva, João Carlos Pereira da , orient. II. Titulo.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

THAIS LUCA MARQUES DE ALMEIDA

ESTUDO SOBRE APLICAÇÃO DE APRENDIZADO DE MÁQUINA PARA IDENTIFICAÇÃO DE ASSALTOS ATRAVÉS DE INFORMAÇÕES DO TWITTER

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 17 de dezembro de 2018 .

BANCA EXAMINADORA:

João Carlos P. da Silva

Prof. João Carlos Pereira da Silva, D.Sc.

Giseli Rabello

Profa. Giseli Rabello Lopes, D.Sc.

Fabrício Firmino de Faria

Prof. Fabrício Firmino de Faria, M.Sc.

AGRADECIMENTOS

Dedico este trabalho às mulheres da minha vida: minha mãe Angela Maria, minha irmã Thatiane Luca e minha avó Celestina. A última com muito carinho porque sei que com certeza estaria na minha colação para me aplaudir. Por toda a força e auto estima que me passaram em todos esses anos. Agradeço também ao meu pai, Carlos Alberto, por todo apoio durante minha trajetória. Também dedico este trabalho à todas as mulheres que, assim como eu, vencem batalhas todos os dias nesta área com pouca representatividade.

Não posso deixar de agradecer a Deus. Por ter me proporcionado todas as coisas maravilhosas que vivi durante esses cinco anos de graduação. Por estar sempre comigo desde a minha entrada na UFRJ até a conclusão do curso. Por me mostrar o caminho nos momentos mais difíceis, por toda auto estima e força diante dos obstáculos e por me mostrar que eu sou capaz de fazer qualquer coisa.

Agradeço aos meus amigos de graduação, aos que acreditaram e que não acreditaram em mim. Agradeço especialmente a turma de 2013.2 por ter me acolhido e por terem se tornado pessoas tão especiais na minha vida. Obrigada por terem me ensinado tanto e pela oportunidade de aprender com vocês.

Um agradecimento especial ao Hugo Siqueira Gomes, por toda ajuda, carinho e paciência durante o desenvolvimento do projeto. Por ter me emprestado um pouco do seu tempo para me ajudar com dúvidas ou debatendo problemas que apareceram ao longo do desenvolvimento. Por ter se mostrado um grande amigo mesmo estando longe. Também ao João Vitor de Oliveira, minha melhor dupla, monitor e amigo para todas as horas. Agradeço por sempre estar presente nesses anos de graduação.

Agradeço ao meu orientador de projeto final Prof. João Carlos por toda paciência e dedicação durante todos esses meses de trabalho. Não só por ter sido um ótimo orientador, mas também por ter me ajudado sempre que eu estava desanimada ou nervosa com a entrega do trabalho. Agradeço por todo conhecimento passado e pela ideia que me deu a oportunidade de ter contato com uma área incrível da computação que é IA.

Ao Fabrício Firmino e ao Raphael Fonseca, por terem me ajudado a dar os primeiros passos, me auxiliando com as requisições da API do Twitter e por estarem dispostos a me ajudar quando eu precisei.

A todos que estiveram dispostos a doar um tempo das suas vidas corridas classificando *tweets*: Alessandra Lima, Amanda Lima, Alice Grobberio, Igor Carpanese, João Vitor de Oliveira, Kátia Fugazza, Lucas Rodrigues, Maíra Passos, Matheus Santiago, Nathany Lisboa, Raffael Siqueira, Taynara Ferreira, Vanessa Leite e Yasmin Assumpção. Obrigada por terem tornado este trabalho possível.

RESUMO

O Twitter é uma plataforma de serviço de *microblogging* que tem chamado a atenção de diversos pesquisadores graças ao grande volume de dados que são gerados diariamente. Nesta plataforma, usuários enviam e recebem mensagens, chamadas *tweets*, de até 280 caracteres em tempo real. Por causa da sua popularização e do aumento da violência no estado do Rio de Janeiro, muitos usuários utilizam o serviço para relatar assaltos, em uma forma de tentar proteger uns aos outros.

Neste trabalho, o objetivo é criar um sistema que fornece um mapeamento dos bairros do Rio de Janeiro com o maior número de denúncias de assalto segundo informações coletadas do Twitter. Para selecionar os *tweets* que contêm informações de assaltos, são utilizados filtros de busca por palavras-chave e outros que garantem que os textos foram postados na cidade ou em locais próximos. Após a coleta dos dados, foram aplicadas técnicas de processamento de linguagem natural para melhorar a qualidade dos mesmos. Além disso, foram investigados métodos de aprendizados para treinar classificadores que identifiquem se um *tweet* relata ou não um assalto. Como todos os métodos a serem testados utilizam regressão, foi necessário representar esses dados numericamente e a forma de representação escolhida foi a Ponderação TF-IDF. Durante o experimento, foi investigado o desempenho de três métodos, a partir de uma base anotada utilizada para treinamento e validação, são eles: SVM, Naive Bayes e Redes Neurais Artificiais.

Palavras-chave: Classificação. Twitter. Processamento de Linguagem Natural. Aprendizado de Máquina.

ABSTRACT

Twitter is a popular microblogging service that has received the attention of many researchers due to its large user base. Twitter users can send and receive real-time messages, called tweets, which must be fewer than 280 characters. Due to the popularity of this social network and the increasing violence in the state of Rio de Janeiro, many people use Twitter to alert family and friends about robberies.

In this paper, we build a system that can monitor robberies in Rio de Janeiro's neighborhoods which will use information provided by Twitter. To select tweets about robbery, we apply keyword and location filters to guarantee that all tweets were post in the city or nearby it. After collect tweets, we used natural language processing techniques to further shape the data. Furthermore, we investigate machine learning methods to train classifiers to determine if a tweet is alerting of a crime. As all of the machine learning methods used for the tests apply regression, the data must be represented numerically. We chose TF-IDF vector representation. We investigate the performance of three different methods against labeled data used for training and validation: SVM, Naive Bayes and Artificial Neural Networks.

Keywords: Classifiers. Twitter data. Natural Language Processing. Machine Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
API	*Application Programming Interface*
ER	Expressão Regular
HTTP	Protocolo de Transferência de Hipertexto
IA	Inteligência Artificial
IDF	*Inverse Document Frequency*
ISP	Instituto de Segurança Pública
JSON	*JavaScript Object Notation*
L-BFGS	*Limited memory Broyden-Fletcher-Goldfarb-Shanno*
NBC	Naive Bayes *Classifier*
PLN	Processamento de Linguagem Natural
ReLu	*Rectified Linear Units*
RI	Recuperação da Informação
RSS	*Residual Sum of Squares*
SGD	*Stochastic Gradient Descent*
SVD	*Singular Value Decomposition*
SVM	*Support Vector Machine*
TF	*Term Frequency*
TF-IDF	*Term Frequency–Inverse Document Frequency*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Os grandes eventos prometiam fazer do Rio uma potência de turismo, mas o turista que se interessa pelo maior destino turístico do Brasil enfrenta uma realidade aterrorizante. Logo no ano seguinte aos Jogos Olímpicos de 2016, o estado registrou uma alta de 91,5% para os principais tipos de roubo em relação a 2010 [13]. Em abril de 2017, o número de roubos já havia batido recorde histórico no Rio: foram 12.089 casos registrados pelo Instituto de Segurança Pública (ISP).

Não é de hoje que o carioca convive com uma cidade violenta. Desde 2012 que o número de roubos vem aumentando na cidade e tem crescido mais nas áreas turísticas nos últimos anos. Em Copacabana, houve um crescimento de 128% nos registros de roubos, que aumentaram de 246 ocorrências registradas entre janeiro e abril de 2015 para 563 ocorrências no mesmo período em 2017. Em Ipanema, os roubos aumentaram 54%: passando de 165 entre janeiro e abril de 2015 para 254 nos primeiros quatro meses de 2017. No Leblon, o número de ocorrências aumentou de 507 para 648 no período, contabilizando um aumento de 28%. Os crimes com maior número de registro são: crimes domésticos; crimes de rua, como roubos a pedestre, roubos de celulares e de veículos; e também assaltos em coletivos. Estes são considerados crimes de oportunidade, por serem rápidos e pontuais, e são os mais complicados de serem combatidos [9].

Durante o Carnaval de 2018, houve um aumento de 700% de ocorrências durante os dias de blocos no Centro do Rio, região que possui a terceira maior concentração de blocos da cidade. A maior parte dos registros eram relativos a roubos e furtos de celulares, documentos e dinheiro. De todas as ocorrências registradas, 90% envolviam falsos ambulantes, uso de aplicativos de transporte compartilhado e fotos no meio da multidão [10]. Segundo o Estadão, quem mais sofreu com a violência no Carnaval foram turistas, pois o número de roubos a turistas teve aumento de 613,63% em relação ao ano anterior [6].

Em meio ao descaso e a falta de dinheiro do estado, o turista que deseja visitar o Rio pode escolher outro destino para passar as férias, enquanto a população se

vê obrigada a conviver com a falta de segurança diariamente. E, enquanto a cidade maravilhosa vive o caos, o medo e a insegurança, o carioca tenta se proteger como pode, seja pelo uso de aplicativos de registro de roubos e furtos [7] ou pelo alerta de vizinhos e pessoas próximas. Para ajudar uns aos outros, muitos costumam utilizar as redes sociais para contar relatos, já que muitos dos casos, principalmente furtos, não costumam ser levados à delegacia. Se são levados, geralmente não é feito um boletim de ocorrências. Uma rede social muito utilizada para divulgar informações sobre furtos e assaltos é o Twitter [1].

O Twitter é uma plataforma de serviço de *microblogging* que permite aos usuários enviar e receber pequenas mensagens, chamadas *tweets*, de até 280 caracteres em tempo real. Por conta do formato livre das mensagens e fácil acessibilidade, os internautas tendem a migrar de meios tradicionais de comunicação, como blogs tradicionais ou listas de e-mails, para este tipo de serviço [21]. Também por ser uma plataforma de disseminação rápida, o Twitter já foi utilizado em aplicações dos mais diversos domínios e é constantemente escolhido para coletar dados de pesquisa, por ter a maioria dos seus dados disponíveis abertamente para pesquisadores e desenvolvedores através das APIs públicas do site.

1.1 MOTIVAÇÃO

Não existe uma solução a médio ou longo prazo para acabar com a violência da cidade. Um grande agravante para esta situação está no fato de que alguma das vítimas não procuram as delegacias para efetuar o registro da ocorrência. Por isso, uma das motivações para este trabalho é verificar se realmente podemos obter mais registros de assaltos ou furtos em redes sociais do que registros formais.

Para ajudar a população a se proteger, também existem aplicativos e sites para fazer denúncias, porém como são baseados em um ambiente colaborativo, ainda exigem um esforço das vítimas para registrar uma ocorrência. Sites como Onde Fui Roubado [2], por exemplo, requerem um preenchimento de um formulário contendo

[1] https://twitter.com/

[2] http://www.ondefuiroubado.com.br/

todos os detalhes do assalto, e aplicativos de celular, como o B.O. Coletivo [3], ainda precisam ser baixados pelas vítimas para fazer um registro formal. Usar as redes sociais é uma forma muito mais rápida e natural, o usuário informa amigos e familiares através de um pequeno relato informal e em tempo real. Como as pessoas estão cada vez mais habituadas a compartilhar informações na internet, é intuitivo pensar que podemos encontrar mais denúncias de roubos e furtos em redes sociais do que em mapas colaborativos ou registros formais de ocorrência.

1.2 OBJETIVO

O objetivo deste trabalho é identificar quais são os bairros que mais sofrem com a violência no Rio de Janeiro e fazer um mapeamento de assaltos segundo informações obtidas através dos usuários do Twitter. Também tem como objetivo implementar e aplicar conceitos de processamento de linguagem natural e aprendizado de máquina nos textos coletados. O primeiro para limpeza e melhoria desses dados e o segundo para determinar qual é o melhor método possível para identificar *tweets* que relatam assaltos. Além disso, disponibilizar os bairros mais perigosos de acordo com os *tweets* coletados durante o desenvolvimento do projeto. Utilizando os casos registrados e divulgados pelo ISP, verificar se realmente podemos encontrar mais denúncias de assaltos em redes sociais do que em registros formais.

1.3 ESTRUTURA

Este trabalho está organizado em 7 capítulos. No capítulo 2, serão explicados conceitos usados para estudo e desenvolvimento deste trabalho, como processamento de linguagem natural, aprendizado de máquina, etc. No capítulo 3, serão apresentados trabalhos que serviram como referência e o quanto estão relacionados com este trabalho. No capítulo 4, será apresentada a arquitetura para montagem da base de dados, representação dos dados e seu armazenamento. O capítulo 5 mostra uma análise sobre o desempenho de quatro métodos de aprendizado treinados com uma

[3] https://www.techtudo.com.br/tudo-sobre/bo-coletivo.html

uma parte da base de dados. O capítulo 6 mostra o uso desses métodos sobre novos dados, os bairros mais perigosos segundo os usuários do Twitter e a comparação entre as denúncias feitas via Twitter com os registros de ocorrência do ISP. O sétimo e último capítulo apresenta as conclusões feitas a respeito deste trabalho e propostas sobre trabalhos futuros.

2 TRABALHOS RELACIONADOS

A recente popularização das redes sociais vem sendo alvo de interesse para extração de diversas informações, afinal muitas pessoas conectadas gera um grande volume de dados diariamente. Só no Twitter são 330 milhões de usuários ativos por mês, 100 milhões logados diariamente, que produzem aproximadamente 500 milhões de *tweets* por dia [20]. Tanta informação disponibilizada em larga escala chamou a atenção dos pesquisadores, já que no Twitter, podemos encontrar usuários falando dos mais diversos temas, o que permite montar bases de dados sobre diferentes assuntos.

2.1 DETECTAÇÃO DE EVENTOS

Enquanto usuários de blogs tradicionais apenas atualizam suas páginas em dias esporádicos, os usuários do Twitter costumam produzir novos *tweets* em diferentes momentos durante o dia e, mesmo que não postem nada no dia, tendem a checar o site constantemente para saber o que há de novo. Essa característica de produção de conteúdo em tempo real é muito importante para detectar eventos. Principalmente para eventos que possuam características em comum, que são: eventos de larga escala nos quais muitos usuários podem ter a mesma experiência, eventos que particularmente influenciam a vida cotidiana das pessoas, por isso elas falam sobre, e que sejam esparsos e temporariamente regionais, assim é possível obter uma localização estimada em tempo real. Esses eventos podem ser grandes eventos sociais como grandes festas, eventos esportivos, exposições, acidentes e campanhas políticas. Também estão inclusos desastres naturais como tempestades, chuva forte, tornados, tufões/furacões/ciclones e terremotos [26].

Por causa dos inúmeros terremotos que acontecem no Japão, pesquisadores da Universidade de Tóquio utilizaram a produção de dados em tempo real do Twitter para propor um sistema de notificações de terremotos através do monitoramento de *tweets* [26]. É possível detectar terremotos monitorando as postagens graças ao grande número de usuários japoneses na rede e como estes estão dispersos geogra-

ficamente pelo arquipélago. Cada usuário funciona como um sensor e cada *tweet* é uma informação enviada por este sensor. A busca por textos que informam sobre terremotos é feita através de palavras-chave como "*earthquake*" ["terremoto"] e "*shaking*" ["tremendo"]. Como ambos os termos podem ser aplicados em outras situações como "*I am attending an Earthquake Conference*" ["Estou participando de um conferência sobre terremotos"] e "*I am shaking hands with his boss*" ["Estou apertando a mão do chefe dele"], foi preparado um conjunto de treinamento para que pudesse ser feito um classificador utilizando *Support Vector Machine* (SVM). Este foi baseado nos seguintes atributos: palavras-chave do *tweet*, número de palavras e contexto das palavras que identificam o evento. Também são aplicados filtros de Kalman [8] e de partículas [11], que costumam ser usados para estimar localização. O modelo criado pode ser aplicado a vários idiomas e pode detectar terremotos com alta probabilidade: 96% dos tremores de escala 3 ou mais registrados pela Agência Meteorológica do Japão (*Japan Meteorological Agency* - JMA) são detectados. O sistema envia um e-mail assim que um tremor é detectado. Este talvez chegue antes que a onda se propague para uma certa região.

Também aproveitando as informações disponibilizadas em tempo real, Aramaki et al. [1] também escolheram o Twitter como fonte de dados para criar um modelo que permite prever epidemias de gripe. Em sua pesquisa, foram utilizados *tweets* em inglês e dois critérios para persistência no conjunto de dados: o usuário que postou ou alguém próximo, no máximo na mesma cidade, deve conter a doença e o tempo verbal da postagem deve estar no presente ou em um passado próximo, com prioridade para as últimas 24h (como no uso da palavra "ontem"). Também devem ser frases afirmativas. Por exemplo, "*A bad influenza is going around in our lab*" ["Uma gripe ruim está circulando por nosso laboratório"], "*I think I'm coming down with the flu*" ["Acho que estou ficando gripado"] e "*My flu is worse than it was yesterday*" ["Minha gripe está pior do que ontem"] são exemplos positivos, enquanto "*Influenza is now raging throughout Japan*" ["A gripe agora se espalha pelo Japão"], "*His wife also contracted the bird flu, but has recovered*" ["A esposa dele também contraiu a gripe aviária, mas já se recuperou"] e "*You might have the flu. Has anyone around you had it?*" ["Você deve estar gripado. Alguém próximo a você

esteve com a doença?"] são exemplos negativos. O artigo também utiliza SVM para separar *tweets* positivos e *tweets* negativos.

Trânsito e acidentes no tráfego também são assuntos constantemente falados no Twitter. Inclusive já existem perfis apenas para informar sobre os assuntos, o que garante que a plataforma também pode ser usada para monitorar o trânsito. Foi assim que Bisio et al. [4] decidiram investigar a produção de textos em tempo real. A aplicação utilizou *tweets* em italiano que foram pré-processados e depois classificados em *tweets* que falavam sobre tráfego ou não. Também classificando *tweets* através do SVM, o sistema foi utilizado para monitorar a rede de rodovias italiana e determinar se um congestionamento é ou não é causado por eventos externos.

2.2 MINERAÇÃO DE OPINIÃO E ANÁLISE DE SENTIMENTO

Além do uso citado acima, os milhões de *tweets* produzidos diariamente também podem conter opiniões e descrever os sentimentos dos usuários. Atraídas por esse grande volume de opiniões de diversos usuários ao redor do mundo, várias companhias se interessam pelo Twitter para ganhar um *feedback* de seus produtos, realizar pesquisas de mercado e analisar quais são as novas demandas dos seus consumidores.

O interesse de empresas no Twitter é tão grande que o site já possui um serviço de "Recurso das marcas" [1], que oferece serviços para tratamento de *tweets* e logotipos/ícones da marca Twitter para serem utilizados em sites de empresas que possuam perfil na rede social. Além de empresas, partidos políticos também procuram saber como as pessoas reagem aos seus planos de governo através da plataforma, que também acaba servindo para campanhas políticas. Por conta destes e outros motivos similares, os dados coletados são principalmente usados para minerar opinião e para tarefas de análise de sentimento.

Motivados por este tipo de análise, Pak et al. [21] decidiram criar um classificador para sentimentos que não precisa de um operador humano para classificar um conjunto de treinamento. Os *tweets* são divididos em três conjuntos através de

[1] https://about.twitter.com/pt/company/brand-resources.html

uma análise linguística: textos contendo emoções positivas (felicidade, diversão ou alegria), textos contendo emoções negativas (tristeza, raiva ou desapontamento) e textos que não expressam emoções.

Inspirados pelo trabalho de Pak et al. [21], Narr et al. [19] criaram um classificador para análise de sentimentos independente de um idioma. A motivação para sua criação foi o Twitter ser uma plataforma de uso internacional e possuir uma grande variedade de *tweets* em diversos idiomas, o que implica em grande parte da informação perdida quando se cria um classificador a partir de um idioma específico.

Ambos os trabalhos utilizaram o algoritmo de classificação Naive Bayes para treinar seus classificadores, o segundo contou com *tweets* em 4 idiomas diferentes: inglês, alemão, francês e português; que foram anotados utilizando o *Amazon's Mechanical Turk* [2]. Ambos assumem que cada *tweet* tem apenas uma única opinião, para que não seja possível separar múltiplos sentimentos em um mesmo *tweet*. No segundo trabalho, também assume-se que todas as palavras possam ser identificadas através de espaços, o que faz com que o classificador seja apenas aplicado a linguagens em que todas as palavras são separadas por espaços. Ambos os estudos também trabalharam em cima de *emojis*, mas só o segundo considerou que "carinhas" com múltiplos caracteres como ":(((" intensificam mais o sentimento do que quando se utiliza apenas um caracter como em ":(".

As opiniões vão muito além de *feedbacks* para empresas. Estudantes da Universidade Sathyabama, em Chennai, Índia, decidiram recolher informações sobre medicamentos utilizados pelos usuários no Twitter [2]. Através de um classificador, o objetivo é identificar a opinião do usuário sobre algum tipo de remédio para que outras pessoas possam escolher o melhor medicamento baseado em opiniões de pessoas que já o utilizaram. O trabalho consistiu em criar um conjunto de dados que define se uma palavra é "boa" ou "ruim" por uma classificação já feita previamente. Depois que os *tweets* são pré-processados, são classificados em *tweets* relacionados a drogas e *tweets* que falam sobre doenças. Essa classificação é baseada no plano de decisão do algoritmo de classificação SVM. O SVM é também utilizado para identi-

[2] https://www.mturk.com/

ficar a opinião do usuário que postou a mensagem sobre o medicamento. Isto feito através do uso do conjunto pré-classificado, baseando-se em palavras que descrevem sentimentos e opiniões como “bom”, “mau”, “não”, etc. Os textos podem ser classificados em positivos, negativos, ambos ou neutros. Este é um classificador que pode ser perigoso, pois tomar medicação sem consultar um médico pode ser prejudicial à saúde. Remédios tem reações adversas e possuem contraindicações que devem ser respeitadas.

2.3 CONCLUSÃO

Neste capítulo, foram vistos os trabalhos que serviram como base para este estudo. Tê-los como referência foi muito importante para tomar a decisão de quais métodos de aprendizado utilizar na classificação dos *tweets*.

Há algumas semelhanças entre o trabalho proposto e os trabalhos apresentados. Além dos métodos de aprendizado utilizados, também temos a importância do tempo verbal durante a classificação. Apesar de não exigir que os verbos estejam conjulgados em um tempo específico, a própria escolha de palavras-chave neste trabalho já determina o uso de *tweets* com verbos conjulgados no passado. Outra semelhança com dois dos trabalhos, é a preocupação em identificar o local onde tudo ocorreu. Como no caso das epidemias de gripe e dos terremotos no Japão, também temos interesse em uma área específica, que é a cidade do Rio de Janeiro.

A principal diferença que pode ser observada em relação aos outros trabalhos é o estímulo por *tweets* em tempo real. Neste trabalho, não estamos tratando de grandes eventos ou com informações relatadas em tempo real. Este trabalho explora os *tweets* de um tipo de usuário diferente dos usuários dos outros trabalhos. Dificilmente uma pessoa relata nas redes sociais que foi assaltada imediatamente, o que é diferente do usuário que está no engarrafamento e usa o Twitter para reclamar, por exemplo. Por isso, a principal diferença entre este e os outros trabalhos está em não explorar diretamente as informações em tempo real.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma breve descrição dos conceitos que serviram como base para este trabalho. Descreveremos conceitos básicos de Processamento de Linguagem Natural (PLN) e métodos de Aprendizado de Máquina (AM) usados na resolução de problemas que envolvem grandes volumes de dados.

3.1 PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento de Linguagem Natural (PLN) consiste no desenvolvimento de modelos computacionais para a realização de tarefas que dependem de informações expressadas em alguma linguagem natural [23]. É muito utilizada em mineração de textos, atividade que visa descobrir conhecimento em dados não estruturados, em sistemas de tradução e interpretação de textos, busca de informações em documentos, etc. Tarefas de PLN necessitam que seus dados passem por um pré-processamento antes de serem utilizados para extrair alguma informação. Portanto, a seguir são apresentadas as técnicas de pré-processamento utilizadas neste trabalho e algumas formas de representar estes dados para desenvolvimento do mesmo.

3.1.1 Pré-Processamento

Antes de qualquer tipo de processamento sobre dados em linguagem natural, é preciso que estes sejam pré-processados. Esta etapa é executada imediatamente após a coleta dos dados e é a parte que consome mais tempo no processo de extração de informações. Podem ser utilizados diversos métodos para aumentar a qualidade dos dados, algumas técnicas podem ser combinadas e outras podem ser utilizadas conforme a necessidade do conjunto de dados.

Limpeza de dados consiste em retirar do texto aquilo que é desnecessário. Nesta etapa, são identificadas anomalias para redução de inconsistências e são tomadas as medidas necessárias para tornar o documento o mais adequado possível. Elementos do texto que podem ser considerados dados inconsistentes são links. Es-

tes normalmente são elementos do texto que não carregam informações importantes e por isso podem atrapalhar o processamento dos dados. A identificação deste tipo de anomalia pode ser feita através do uso de **Expressões Regulares** [3]. Muito usadas em todas as linguagens de programação e em processadores de palavras e de textos, em sua definição formal, uma expressão regular (ER) é uma expressão que caracteriza um conjunto de *strings*. ERs são particularmente usadas em textos quando se deseja buscar por um padrão específico em um conjunto de palavras. Sua função é varrer o *corpus*, que pode ser um documento ou uma coleção, procurando e retornando todas as palavras ou textos que satisfazem esse padrão. Neste trabalho, ERs são muito importantes para identificar e remover links porque estes costumam ser mais difíceis de serem completamente removidos através de uma busca mais simples.

Tokenização, ou atomização, é o segundo passo da fase de pré-processamento e tem como objetivo quebrar o texto em segmentos menores, podendo ser frases ou palavras. Quando é feita uma segmentação por palavras, cada segmento é chamado **token** e, neste caso, também estão incluídos caracteres e pontuação. Espaços são sempre descartados e, em caso de palavras combinadas, ou separadas por caracteres como "&" e "-", estas devem ser unidas para formar um único *token*. Em algumas tarefas de PLN, todos os *tokens* devem ser processados para serem representados em letras minúsculas. Este procedimento não é feito em tarefas de análise de sentimento, extração de informações e traduções, pois palavras em letras maiúsculas e minúsculas podem alterar o sentido do texto. No exemplo, a frase: "O restaurante de São Francisco não cobra $10." pode ser separada em 10 *tokens*, como é mostrado a seguir:

O restaurante de São Francisco não cobra $10.
[O] [restaurante] [de] [São] [Francisco] [não] [cobra] [$] [10][.]

Lista de abreviações ajuda a identificar abreviações para que elas possam ser substituídas pelas palavras completas correspondentes. Abreviações são um grande complicador e a substituição é feita com a ajuda de dicionários pré-estabelecidos.

Dicionários de abreviações são muito úteis quando os dados são coletados da Internet, diminuindo o número de palavras distintas encontradas, como no caso de "vc" e "você", que são consideradas pelo sistema como palavras diferentes, mas se referem ao mesmo pronome de tratamento.

Remoção de *Stopwords* consiste em remover termos que não possuem nenhum valor semântico, úteis apenas para compreensão do texto. São termos que aparecem com mais frequência em documentos, por isso seus valores discriminatórios são baixos [3]. Artigos, preposições, pontuação, conjunções e pronomes são fortes candidatos a entrarem na lista de *stopwords*, que deve ser estendida conforme o idioma. Alguns verbos, advérbios e adjetivos também podem fazer parte da lista. A remoção de *stopwords* possui um benefício adicional importante: diminuir o tamanho da estrutura de indexação consideravelmente. Podendo haver uma redução de 40% ou mais das palavras do *corpus* apenas com sua remoção.

3.1.2 Representação de Documentos

O processo de representação de um documento busca descrever ou identificar um documento pelo seu conteúdo.

Modelo *Bag Of Words*, ou saco de palavras em português, é a representação mais simples utilizada em processamento de linguagem natural. Também é frequentemente utilizado em métodos de classificação de documentos. Neste modelo, o documento é representado apenas pelo conjunto de palavras que o compõe, desconsiderando estruturas gramaticais e também a ordem como as palavras aparecem no texto. Este é o modelo utilizado logo após o pré-processamento (Seção 3.1.1) e é útil para montagem da matriz de termos e documentos.

Matriz de Termos e Documentos é uma matriz do tipo

$$\begin{array}{c} \\ k_1 \\ k_2 \\ k_3 \end{array} \begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{bmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \\ f_{3,1} & f_{3,2} \end{bmatrix} \end{array}$$

onde cada elemento $f_{i,j}$ representa a frequência do termo k_i no documento d_j. Este tipo de representação fornece mais informação do que simplesmente anotar se um termo ocorre ou não no documento e ainda assim é considerada uma abordagem simplista. Para melhorar a representação, pode-se utilizar uma forma de representação mais robusta como a Ponderação TF-IDF [3]. Descrita na seção a seguir.

3.1.3 Ponderação de Termos: a Ponderação TF-IDF

Nem todos os termos são igualmente úteis para descrever o conteúdo de um documento. Alguns termos podem ser mais vagos que outros e decidir a importância de um termo no conteúdo de algum documento não é uma tarefa trivial. Palavras que aparecem em muitos documentos não são úteis e podem atrapalhar o processamento. Já palavras que aparecem pouco na coleção são bastante úteis, já que reduzem consideravelmente o conjunto de documentos que podem ser de interesse do usuário. Assim cada termo deve receber um peso dependendo do número de documentos em que aparece. Este peso é o que quantifica sua importância para descrever o conteúdo do documento.

A fim de caracterizar a importância dos termos, um peso $w_{i,j}$, em que $w_{i,j} > 0$, é associado a cada termo k_i de um documento d_j na coleção. Para um termo k_i que não aparece no documento, $w_{i,j} = 0$. Esses pesos podem ser computados usando as **frequências de ocorrência** dos termos nos documentos.

Seja $f_{i,j}$ a frequência de ocorrência do termo de indexação k_i no documento d_j, ou seja, o número de vezes que o termo k_i aparece no texto do documento d_j. A frequência total F_i do termo k_i na coleção é a soma das frequências de ocorrência do termo em todos os documentos:

$$F_i = \sum_{j=1}^{N} f_{i,j} \tag{3.1}$$

onde N é o número de documentos na coleção. A **frequência de documentos** de um termo k_i é o número de documentos nos quais ele ocorre e é indicado simplesmente como n_i, onde $n_i \leq F_i$.

Um esquema de ponderação muito popular que faz uso desses conceitos é o esquema de ponderação **TF-IDF**. Este é composto pela frequência do termo (TF, *Term Frequency*) e a frequência inversa do documento (IDF, *Inverse Document Frequency*). A forma de ponderação da frequência dos termos (TF) foi proposta inicialmente por Luhn através da hipótese que ficou conhecida como **Hipótese de Luhn** [3], que diz que o valor ou peso de um termo k_i que ocorre em um documento d_j é simplesmente proporcional à frequência $f_{i,j}$ e quanto mais frequente um termo k_i ocorrer no texto de um documento d_j, maior será a sua frequência de termo $TF_{i,j}$.

Essa hipótese é baseada em que termos com alta frequência são importantes para descrever tópicos-chave de um documento e leva a formulação da ponderação TF como $tf_{i,j} = f_{i,j}$. Há uma outra variante da ponderação TF, mostrada na Equação 3.2. Nesta variante, que é a forma preferível para a ponderação TF, os pesos são tornados diretamente comparáveis aos pesos IDF (discutido a seguir).

$$tf_{i,j} = \begin{cases} 1 + \log f_{i,j} & \text{se } f_{i.j} > 0 \\ 0 & \text{caso contrário} \end{cases} \tag{3.2}$$

Já a ponderação pela frequência inversa de documentos (IDF) é uma interpretação estatística da especificidade dos termos - o quão bem um termo descreve o tópico de um documento. Esta ponderação se tornou fundamental para ponderação dos termos e também é baseada na exaustividade dos termos da linguagem - a descrição de um documento é interpretada como a abrangência que ela provê para os tópicos principais de um documento. Se novos termos forem adicionados a um documento, a sua exaustividade de descrição aumenta. Com descrições mais longas, a especificidade dos termos tende a ficar mais baixa. Pela especificidade, os pesos

dos termos podem ser representados como uma função das frequências relativas dos termos e o IDF de um termo pode ser computado como

$$IDF_i = \log \frac{N}{n_i} \tag{3.3}$$

onde IDF_i é a frequência inversa de documentos do termo k_i e $\frac{N}{n_i}$ é simplesmente uma frequência relativa. Em coleções de grandes proporções, os termos seletivos são geralmente substantivos e grupos de substantivos. Os menos seletivos são normalmente artigos, conjunções e preposições, *stopwords* que geralmente são removidos do texto. Quanto mais alto o peso do termo, mais raro ele é.

Juntando as ponderações 3.2 e 3.3, chegamos ao esquema de ponderação TF-IDF mais popular:

$$tf - idf_{i,j} = \begin{cases} (1 + \log f_{i,j}) \times \log \frac{N}{n_i} & \text{se} f_{i,j} > 0 \\ 0 & \text{caso contrário} \end{cases} \tag{3.4}$$

3.1.4 Normalização pelo Tamanho dos Documentos

O tamanho dos documentos de uma coleção pode variar bastante e isto pode ser um problema. No caso de *tweets*, temos textos pequenos, com poucas palavras, mas ainda podemos ter *tweets* de diversos tamanhos. Para evitar que problemas apareçam, pode-se dividir o número de ordem de cada documento pelo seu tamanho. Essa normalização pelo tamanho dos documentos é amplamente adotada por modelos de Recuperação de Informação (RI) e pode também ser usada em outros tipos de aplicação.

O tamanho dos documentos pode ser computado de diferentes maneiras. Pode ser utilizado o tamanho em bytes (tamanho em bytes do documento), número de palavras (número de palavras nele contidas) ou pela norma. Neste último, documentos são representados como vetores ponderados (representados pelos seus pesos) e a representação de um documento é um vetor composto pelos valores de todos os seus

termos. O tamanho do documento é dado pela norma desse vetor, como ilustrado na Equação 3.5.

$$|\vec{d_j}| = \sqrt{\sum_{i}^{t} w_{i,j}^2} \tag{3.5}$$

A normalização pelo tamanho dos documentos é constantemente usada em RI porque pode garantir uma melhora na qualidade dos resultados. No caso de pequenos textos como *tweets*, não há garantia de que a normalização irá contribuir para melhores resultados, o que será testado durante experimento apresentado nos próximos capítulos.

3.2 DECOMPOSIÇÃO DE MATRIZES: *SINGULAR VALUE DECOMPOSITION*

Nesta seção, será apresentado o algoritmo utilizado neste trabalho para reduzir a dimensão da matriz TF-IDF. Importante citar que é um algoritmo para matrizes não quadradas, já que estamos interessados em reduzir a matriz de termos por documento $M \times N$, onde $M \neq N$ e esta é muito improvável de ser simétrica.

O algoritmo descreve uma extensão da decomposição diagonal simétrica, conhecida como *Singular Value Decomposition* (SVD). Neste tipo de decomposição, dada uma matriz C de dimensões $M \times N$, define-se uma matriz U de dimensões $M \times M$ cujas colunas são os autovetores ortogonais de CC^T. Também é definida V como uma matriz $N \times N$ cujas as colunas são os autovetores ortogonais de C^TC.

Assim, podemos ter uma decomposição SVD da matriz C dada por $C = U \sum V^T$. Em que U, Σ e V^T são matrizes únicas e Σ é uma matriz diagonal onde os valores singulares são positivos e estão ordenados de forma decrescente. Multiplicando C pela sua versão transposta temos:

$$CC^T = U\Sigma V^T V \Sigma U^T = U\Sigma^2 U^T \tag{3.6}$$

O lado esquerdo (CC^T) da equação representa uma matriz quadrada com linhas

e colunas correspondentes a cada um dos M termos. A entrada (i,j) da matriz é uma medida da sobreposição entre os termos i e x, com base em sua co-ocorrência nos documentos. Quando são escritos os valores numéricos do SVD, é conveniente representar a matriz como uma matriz $r \times r$ com os valores singulares nas diagonais, já que todas as suas entradas fora desta sub-matriz são zeros. Portanto, é também conveniente omitir as $M - r$ colunas mais à direita de U correspondentes a estas linhas omitidas de Σ. Da mesma forma são omitidas as $N - r$ colunas mais à direita de V, já que estas correspondem em V^T como as linhas que serão multiplicadas pelas $N - r$ colunas com valor zero de Σ. Esta forma de SVD é normalmente conhecida como *reduced SVD* ou *truncated SVD*.

3.3 APRENDIZADO DE MÁQUINA

Aprendizado de Máquina (AM) é um subcampo de estudo em Inteligência Artificial (IA) que vem se tornando cada vez mais popular e é cada vez mais usado em diversas áreas além da computação. Seu sucesso surge principalmente da necessidade de se trabalhar com grandes volumes de dados [27]. Estes são processados através do uso de métodos estatísticos, técnicas de reconhecimento de padrões e conceitos de teoria de aprendizado computacional para resolver um problema da melhor forma possível. Algoritmos de aprendizado constroem modelos a partir dos dados que lhe são fornecidos para que possam ser utilizados para fazer previsões em dados que ainda não foram vistos pelo algoritmo.

Seu aspecto iterativo é importante pois os modelos podem se adaptar de forma independente quando expostos a novos dados. Através de análises estatísticas aprimoradas sobre os dados fornecidos, um melhor desempenho é alcançado conforme os modelos aprendem de seus erros. Este processo de aprendizado é feito através de experiências/exemplos, dando aos computadores habilidade de aprender sem que sejam explicitamente programados para executar tarefas específicas.

Essa tecnologia vem sendo muito aplicada aos mais diversos tipos de indústria, garantindo que as organizações possam trabalhar com mais eficácia ou ganhem uma

vantagem competitiva sobre seus concorrentes. Técnicas de AM podem ser aplicadas em serviços financeiros para bancos que desejam ganhar intuições importantes ou para prevenção de fraudes, por exemplo. Também é muito utilizada na medicina para ajudar médicos a analisar dados de pacientes e identificar tendências ou alertas, promovendo o aperfeiçoamento de diagnósticos e tratamentos.

A seguir são apresentados conceitos de aprendizado de máquina e alguns de seus métodos, utilizados neste trabalho, como K-Means [15], *Support Vector Machine* [15], Naive Bayes [15] e Redes Neurais [25].

3.3.1 Principais Conceitos em Aprendizado de Máquina

Antes de construir um modelo, é preciso **coletar e preparar os dados** que serão utilizados para executar determinada tarefa. Estes devem ser preparados e carregados para que possam ser utilizados pelo modelo a ser construído.

Dados são representados como vetores onde cada dimensão desse vetor é considerado um **atributo** para o método de aprendizado. Um **atributo** deve representar uma característica importante para a construção do modelo. A forma como os atributos são escolhidos é muito importante, pois os dados são a parte mais importante do aprendizado e sua qualidade determinará o bom funcionamento do modelo proposto.

Um conjunto de valores para esses atributos é uma **instância**. Cada combinação desses valores representa uma entrada a ser mapeada para uma saída do modelo treinado. A Tabela 3.1 representa um pequeno conjunto de exemplos para identificar se uma bebida é vinho ou cerveja, onde cada linha dessa tabela é uma instância e os atributos escolhidos para representar as características das duas bebidas são a cor e o volume de álcool. Em AM, podem ser usados dados rotulados ou não. Na Tabela 3.1, as instâncias são mapeadas para uma saída específica, um rótulo, que pode ser vinho ou cerveja [1].

A Tabela 3.1 é um exemplo de dados rotulados vindos de um processo manual

[1] https://edu.google.com/higher-ed-solutions/google-cloud-platform/

Cor (nm)	Álcool (%)	Rótulo
610	5	Cerveja
599	13	Vinho
693	14	Vinho

Tabela 3.1: Exemplo de conjunto de dados para treinar um modelo que identifique se uma bebida é cerveja ou vinho.

para indicar qual a saída esperada para uma determinada instância. Dados não rotulados são geralmente utilizados em algoritmos de agrupamento, onde são agrupados em n grupos conforme sua proximidade. Uma aplicação que utiliza este tipo de dados pode ser uma empresa que deseja traçar os perfis dos seus clientes. Os m clientes são agrupados em n perfis de acordo com um critério de similaridade entre os valores de seus atributos.

No caso dos dados rotulados do exemplo, o conjunto será dividido para treinamento e validação. O primeiro é chamado **conjunto de treinamento** e irá ser usado pelo algoritmo para construir um modelo que classifique bebidas em vinho ou cerveja. Para medir o desempenho do algoritmo, é utilizado o outro subconjunto dos dados, o **conjunto de testes** ou **validação**.

Após os dados serem tratados e carregados, deve-se escolher o modelo que melhor se adapta ao tipo dos dados, que devem ser embaralhados para que a ordem em que aparecem não interfira no êxito do modelo. Para escolher o modelo, deve-se primeiro observar bem o problema a ser resolvido e o tipo dos dados coletados. Com o problema categorizado pelo tipo de saída, é possível identificar o melhor algoritmo a ser aplicado. Os tipos de aprendizado serão descritos na próxima seção.

Depois de escolhido o algoritmo de aprendizado, é feito o **treinamento**, onde os dados selecionados como conjunto de treinamento são utilizados para gradualmente melhorar a habilidade do modelo em prever ou agrupar novos dados. Nesta fase, chamada de **fase de treinamento**, o modelo é otimizado para ser usado para prever classificações futuras. É durante a fase de treinamento que os parâmetros do modelo são melhorados conforme uma medida de erro, sempre buscando o menor

erro possível. Após o treinamento, é preciso verificar se o modelo construído é bom. Para isso, é utilizado o conjunto de validação com os dados nunca vistos pelo modelo.

O desempenho dos modelos pode ser medido utilizando métricas de avaliação como acurácia, precisão, sensibilidade e especificidade. A primeira é encontrada dividindo o número de classificações feitas de forma correta pelo modelo pelo total de instâncias do conjunto de validação. A precisão diz respeito a proporção de classificações positivas que estava correta. Já sensibilidade e especificidade se referem a porcentagem de acertos positivos e negativos. Para calcular a sensibilidade, o número de instâncias classificadas de forma correta como positivas é dividida pelo total de instâncias rotuladas como positivas. Do mesmo modo é calculada a especificidade: o número de classificações corretas negativas é divido pelo total de instâncias rotuladas como negativas.

É muito importante avaliar o modelo e sua capacidade de generalização. Para avaliar como o modelo se comporta para diferentes conjuntos de dados, aplica-se a técnica de **validação cruzada** [17]. A ideia é treinar o máximo de modelos possíveis, a partir de diferentes conjuntos de treinamento e validação, para obter modelos melhores. Ter um pequeno conjunto de validação significa que esse conjunto pode se adaptar bem ou não ao modelo apenas por sorte. Por isso, existem vários modelos que podem ser usados para reutilizar exemplos para treinamento e validação, como o método de validação cruzada K-Fold.

O **K-Fold** [17] é utilizado para determinar o melhor modelo a partir de diferentes conjuntos de treinamento e validação. O método consiste em particionar o conjunto de dados rotulados em k partes. Para cada chamada do algoritmo, são treinados k modelos utilizando a cada iteração um dos k conjuntos como conjunto de validação. Os dados restantes são usados como conjunto de treinamento. No final das k execuções, pode ser selecionado o modelo que possui o menor erro médio para um conjunto de validação e calcular o desempenho do método para cada validação (média das k execuções).

3.3.2 Tipos de Aprendizado

Um modelo de aprendizado é classificado pelo seu tipo de aprendizado e pelo tipo dos dados que lhe são fornecidos. Modelos podem ser classificados em quatro grupos diferentes: supervisionados, não supervisionados, semi-supervisionados e de aprendizado por reforço. Como este trabalho apenas utilizou algoritmos supervisionados e não supervisionados, apenas estes dois tipos serão descritos a seguir.

No **aprendizado supervisionado** são utilizados dados de exemplos de entrada rotulados. Existe a ideia de um "supervisor" que instrui o modelo de aprendizado a associar os rótulos aos exemplos passados como treinamento. O objetivo é fazer um mapeamento entre um conjunto de variáveis de entrada x e uma saída y.

Dado um conjunto x_i, y_i, onde x_i é uma instância e y_i é sua classificação, a tarefa do modelo supervisionado é encontrar uma função h, tal que, $h(x_i) = y_i$. O modelo construído a partir do conjunto de treinamento será aplicado para prever saídas de dados desconhecidos pelo algoritmo.

Em problemas em que a variável de saída é numérica, a tarefa é chamada de **regressão**. Se a variável de saída é simbólica, a tarefa é chamada de **classificação** e seu valor é chamado **classe**. A classificação ou a regressão pode ser feita calculando-se a distância entre um exemplo de teste e todos os exemplos de treinamento. O cálculo da distância entre os valores de um atributo deve estar de acordo com o seu tipo (numérico ou simbólico). Para atributos numéricos, utiliza-se a distância normalizada (Equação 3.7) e, para atributos simbólicos, pode-se inferir regras como na Equação 3.8. Onde V_1 e V_2 são atributos de uma instância e V_{max} e V_{min} os maiores e menores valores que este atributo pode ter.

$$d(V_1, V_2) = \left| \frac{V_1 - V_2}{V_{max} - V_{min}} \right| \tag{3.7}$$

$$d(V_1, V_2) = \begin{cases} 1, & \text{se } V_1 \neq V_2. \\ 0, & \text{se } V_1 = V_2. \end{cases} \tag{3.8}$$

As principais vantagens em algoritmos de aprendizado supervisionado são: o conjunto de dados poder ser atualizado sem necessariamente alterar o modelo e ser naturalmente adequado a domínios numéricos onde a noção de distância tem significado. São muito utilizados em sistemas de recomendação. Tem como desvantagens o espaço necessário para armazenar todo o banco de dados, o tempo de treinamento, que pode ser alto, e são sensíveis a atributos irrelevantes e ruídos. Os exemplos mais conhecidos são Redes Neurais [25], *Support Vector Machine* [15] e Classificador Naive Bayes [15].

Já no **aprendizado não supervisionado**, não são usadas instâncias pré-rotuladas, não há um conceito alvo associado [30]. Este tipo de aprendizado é utilizado para ganhar alguma percepção da natureza (ou estrutura) dos dados entre as instâncias. O interesse principal é desvendar a organização dos padrões em *clusters* que permitirão descobrir similaridades e diferenças entre padrões.

Seja um conjunto de dados tal que $X = \{x_1, x_2, ..., x_n\}$, define-se como m-agrupamentos de X a partição de X em m conjuntos (*clusters* ou grupos) $C_1, C_2, ..., C_m$, satisfazendo as seguintes condições: cada *cluster* deve conter pelo menos um elemento, a união de todos os elementos de todos os *clusters* deve ser o conjunto X e a interseção entre quaisquer dois ou mais *clusters* deve ser vazia. Elementos em torno de um *cluster* C_i são mais similares entre si e menos similares aos vetores dos outros *clusters*. A definição de similaridade e a escolha de uma medida apropriada para os tipos de dados é essencial no aprendizado não supervisionado.

O algoritmo mais conhecido para agrupar instâncias é o K-Means [15] e este é utilizado para particionar n instâncias em k conjuntos. Tem como objetivo separar diferentes tipos de dados através do cálculo da distância média entre os atributos de uma instância em relação aos atributos das outras instâncias. Tentando sempre minimizar as distâncias de uma instância ao *cluster* que ela pertence. Porém ainda existem vários outros tipos de algoritmos de clusterização [30]: sequenciais, hierárquicos, baseados na otimização de funções de custo, etc.

3.3.3 K-Means

O K-Means é considerado o algoritmo de *clustering* mais importante [15]. Seu objetivo é minimizar a distância euclidiana quadrática média dos documentos aos centros dos seus *clusters*. O centro de um *cluster* é a média ou o centróide $\vec{\mu}$ dos documentos em um cluster ω:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{X} \in \omega} \vec{X} \tag{3.9}$$

Um *cluster* ideal no K-Means é um esfera cujo o centróide é seu centro de gravidade. Certamente um *cluster* não deve sobrepor um ou mais *clusters*. A medida utilizada para calcular quão bem os centróides representam os membros dos seus *clusters* é chamada de soma residual de quadrados (*Residual Sum of Squares* - RSS), que é a distância ao quadrado de cada vetor de seu centróide mais esse resultado para todos os outros vetores:

$$RSS_k = \sum_{\vec{X} \in \omega_k} \left| \vec{X} - \vec{\mu}(\omega_k) \right|^2 \tag{3.10}$$

$$RSS = \sum_{k=1}^{K} RSS_k \tag{3.11}$$

É essa função RSS (3.11) a função objetivo do algoritmo e deve-se minimizá-la. Minimizar essa função é equivalente a minimizar a distância quadrada média.

O primeiro passo do K-Means é escolher aleatoriamente k documentos como centros iniciais de k *clusters*, que são chamados de **sementes**. Escolhidas as sementes, o algoritmo move os centros pelo espaço para tentar minimizar a função RSS. É um algoritmo iterativo que sempre repete os mesmos dois passos: atribuir documentos ao *cluster* com o centróide mais próximo e recalcular cada centróide com base nos membros atuais de seu *cluster*. A Figura 3.1 mostra como o algoritmo funciona para nove iterações.

Como todo algoritmo iterativo, o K-Means também possui um critério de pa-

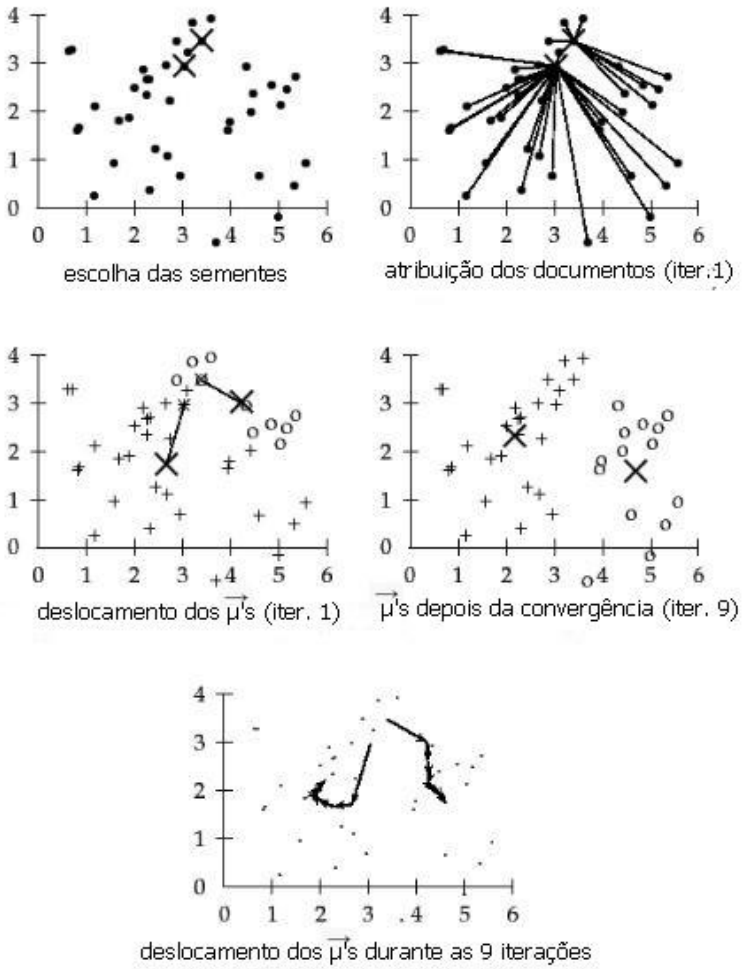

Figura 3.1: Exemplo do uso do K-Means para K=2 em $\mathbb{R}^2$.

rada. Este critério pode ser dado por um número máximo de iterações completas, limitando o tempo de execução do algoritmo, que pode atrapalhar o algoritmo e levar a um agrupamento ruim devido ao número insuficiente de iterações. O K-Means também para se a atribuição dos documentos aos *clusters* (sua função de particionamento) ou os centróides não mudam mais entre iterações. No caso da atribuição, se existe um mínimo local ruim, o algoritmo produz um bom agrupamento, mas os tempos de execução podem ser extremamente longos. O algoritmo também para caso a RSS fique abaixo de um certo limite, critério que garante que o agrupamento tenha a qualidade desejada após o término. Caso a função RSS fique abaixo de um limite muito pequeno, está próximo de convergir. O algoritmo fica mais próximo de convergir quando a função RSS decresce a cada iteração.

3.3.4 *Support Vector Machines*

Aplicadas com sucesso em problemas de RI e classificação de textos, *Support Vector Machines* (SVMs) ou Máquinas de Vetores de Suporte, são algoritmos baseados em uma noção de "margem". Utilizando um espaço vetorial, seu objetivo é procurar uma superfície de decisão que esteja o mais distante possível de qualquer um dos dados (pontos) do conjunto de treinamento, desconsiderando alguns como *outliers* (pontos isolados) ou ruídos. Aqui trataremos do SVM apenas no caso linear, usado para separar os dados em apenas duas classes como mostra a Figura 3.2.

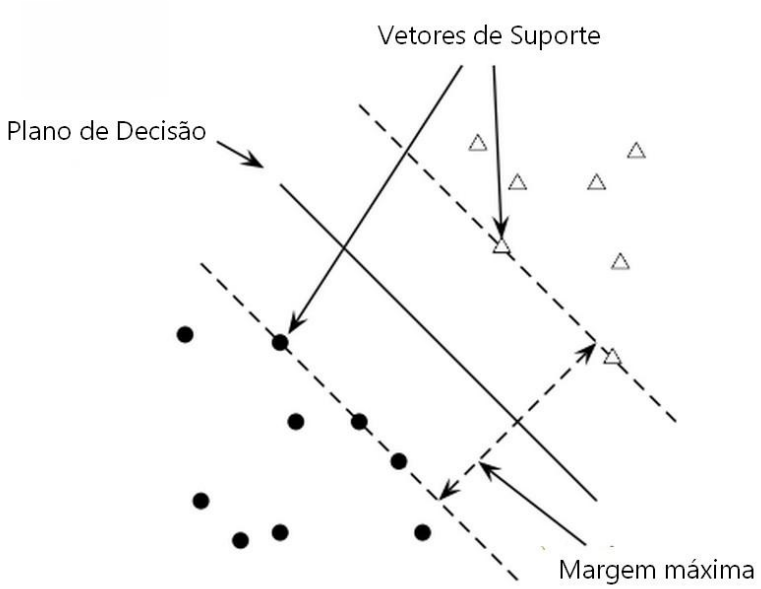

Figura 3.2: Na figura, temos o particionamento dos dados em duas classes e os vetores de suporte são os cinco pontos que se encontram à esquerda da margem do classificador.

É essa distância entre o plano e o ponto mais próximo que determina a **margem** do classificador. Os dados do conjunto de treinamento são utilizados para alimentar funções de otimização sobre o plano de decisão. A função de decisão do SVM é definida por um subconjunto (geralmente pequeno) dos pontos que definem a posição do separador. Esses pontos são chamados **vetores de suporte**, onde cada vetor é formado por um ponto e a origem. Como também mostra a Figura 3.2.

Pontos mais próximos ao plano de decisão representam decisões de classificação

incertas, por isso o algoritmo tem como objetivo tentar maximizar essa margem. Para pontos mais próximos ao plano, existe 50% de chance do classificador decidir de qualquer maneira, escolhendo de forma aleatória qualquer uma das classificações. Já classificadores que possuem uma margem maior, que mantém o plano de decisão mais longe dos pontos, não tomam decisões com pouca certeza. Isto garante à classificação uma margem de segurança, pois pequenos erros de medição ou pequenas variações na representação do documento não levarão a uma classificação incorreta. Se o ponto estiver dentro da margem, o classificador pode retornar "não sei" como resposta, ao invés de escolher de forma aleatória uma das duas classes.

3.3.5 Classificador Naive Bayes

Redes Bayesianas são modelos baseados no Teorema de Bayes, portanto fornecem predições associadas a probabilidades. O algoritmo de classificação Naive Bayes (*Naive Bayes Classifier* - NBC) é uma simplificação de redes bayesianas que na prática alcança um bom desempenho.

É um classificador muito utilizado em AM e é denominado *naive* (ingênuo) por assumir que os atributos são condicionalmente independentes. O algoritmo simplesmente ignora a relação entre os atributos, ou seja, a informação de um evento não é informativa sobre nenhum outro. Para problemas de aprendizado de máquina, é fácil perceber que esta premissa é falsa, já que os atributos são geralmente dependentes. No caso de *tweets*, a presença de um atributo com um outro específico pode determinar se um texto é positivo ou negativo, por exemplo. Mesmo considerando os atributos como independentes, o modelo resultante é fácil de ajustar e o classificador demonstra ter bom desempenho em várias tarefas de classificação [16].

A probabilidade de um documento pertencer a uma classe c é calculada como em 3.12 [18]. Onde x é um atributo escalar no espaço e y é a classificação para uma classe c. D é um vetor de números reais ou *bits* binários. Assume-se que as classes não estão ordenadas e são mutualmente exclusivas.

No caso de dados de valores contínuos, que são distribuídos de forma normal

(gaussiana), temos a probabilidade calculada como em 3.13, onde a média (μ) e a variância (θ) dependem da classe c.

$$p(x|y=c) = \prod_{i=1}^{D} p(x_i|y=c) \tag{3.12}$$

$$p(x|y=c,\theta_c) = \prod_{i=1}^{D} N(x_i|\mu_{ic},\theta_{ic}) \tag{3.13}$$

Para dados binários, o parâmetro θ_{ic} da função Bernoulli deve ser estimado e a probabilidade é calculada como em 3.14.

$$p(x|y=c,\theta_c) = \prod_{i=1}^{D} Be(x_i|\theta_{ic}) \tag{3.14}$$

Para classificação de textos, o algoritmo mais indicado é o Naive Bayes Multinomial. Neste caso, cada documento é visto como uma coleção de palavras e a ordem dessas palavras nos documentos é considerada irrelevante. A probabilidade de um documento ser da classe c é dada por:

$$P(c|d) = \frac{P(c)\prod_{\omega\in d} P(w|c)^{n_{wd}}}{P(d)} \tag{3.15}$$

onde n_{wd} é o número de vezes em que a palavra w ocorre em um documento d, $P(w|c)$ é a probabilidade condicional do termo w ocorrer em um documento da classe c. Neste caso, esta probabilidade é interpretada como uma medida para evidenciar o quanto w contribui para que c seja a classe correta do documento. $P(c)$ é a probabilidade a priori de um documento pertencer a classe c e $P(d)$ é uma constante que faz com que a soma das probabilidades para as diferentes classes resulte em um. $P(c)$ é estimada pela frequência relativa do número de documentos de uma classe e todos os documentos do conjunto: $\frac{D_c}{D}$, onde D_c é o número de documentos classificados como da classe c e D é o número total de documentos do conjunto. Caso um termo de um documento não forneça uma evidência clara de uma classe contra a outra, é escolhida a classe que tiver frequência relativa mais alta. $P(w|c)$ é estimada como:

$$P(w|c) = \frac{1 + \sum_{d \in D_c} n_{wd}}{k + \sum_{w'} \sum_{d \in D_c} n_{w'd}} \tag{3.16}$$

onde D_c é a coleção de todos os documentos do conjunto de treinamento que pertencem à classe c, e k é o tamanho do vocabulário, i.e., número de palavras distintas em todos os documentos do conjunto. O adicional no numerador é chamada de correção de Laplace e corresponde a inicialização do contador de cada palavra como 1 ao invés de 0. É necessário também adicionar k no denominador para obter uma distribuição de probabilidades que some um. Esse tipo de correção é necessário por conta do problema das frequências zero: uma única palavra que pertence a um documento de teste d, que não ocorre em nenhum dos documentos de treinamento de uma determinada categoria c, irá render uma probabilidade $P(c|d)$ igual a zero.

3.3.6 Redes Neurais Artificiais

Inspiradas nas redes biológicas, as redes neurais são formadas por conjuntos de neurônios - células responsáveis por processar informação no cérebro humano. A rede corresponde a uma coleção de neurônios interconectados. Uma rede neural artificial é composta por um número de **nós** conectados por **links**, onde cada link possui um **peso** associado a ele.

Os pesos são o principal meio de armazenamento das redes e o processo de aprendizado geralmente ocorre pela atualização desses pesos. Alguns nós são conectados ao ambiente externo e podem ser designados como nós de entrada ou de saída. Treinar uma rede neural consiste em determinar os pesos de forma que o erro na saída seja o menor possível.

Cada nó possui um conjunto de links de entrada e um conjunto de links de saída para outros nós. Também possui uma **função de ativação**, um meio de calcular o nível de ativação no próximo passo a partir dos dados de entrada e os pesos atuais. A ideia é que cada nó faça cálculos locais baseados nas entradas fornecidas pelos seus vizinhos, sem a necessidade de qualquer controle global sobre todo o conjunto de nós.

Um nó recebe sinais (entradas) fornecidos por seus nós vizinhos e calcula a nova saída utilizando os pesos de cada link. Podemos ter dois tipos de cálculos de saída. No caso linear, não temos uma função de ativação, a saída é dada apenas pela soma ponderada dos valores de entrada e os pesos dos links. No caso não linear, existe uma função de ativação que recebe como entrada a soma ponderada das entradas e seus pesos mais uma entrada zero, chamada de bias b, e irá retornar uma saída y. As Equações 3.17 e 3.18 mostram como a saída é calculada e a Figura 3.3 a estrutura de um neurônio.

$$z = b + \sum_{i=1}^{N} w_i x_i \tag{3.17}$$

$$y = f(z) \tag{3.18}$$

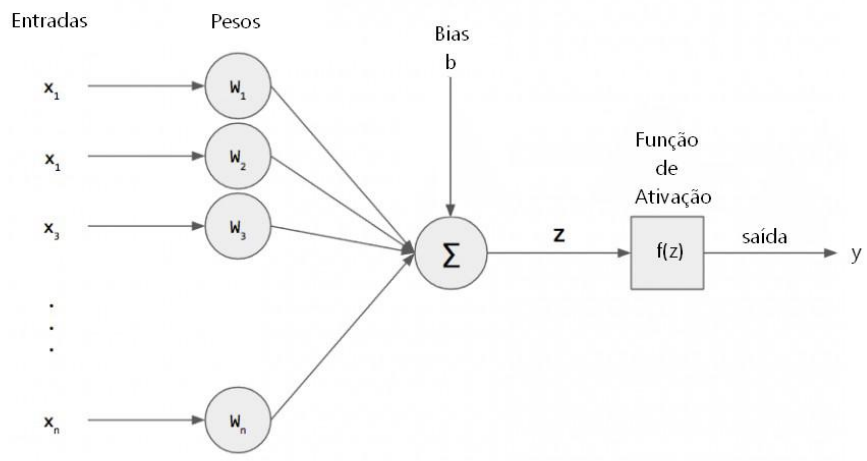

Figura 3.3: Estrutura de um neurônio.

Nas Equações 3.17 e 3.18, N é o número total de entradas e f é a função de ativação. Normalmente todos os nós possuem a mesma função de ativação e podem ser obtidos diferentes modelos variando essa função. Os três tipos de funções mais utilizados são função degrau, função sinal e função sigmóide [25].

Para problemas mais complexos, são necessários mais neurônios. Os neurônios de uma rede neural são organizados em **camadas**. O número de neurônios por

camada e o número de camadas é importante para um melhor desempenho da aplicação. Camadas intermediárias são chamadas **camadas ocultas** e tem o propósito de capturar/representar as diversas nuances que os dados de treinamento possam ter. Quanto mais camadas ocultas, mais ajustada pode ficar a rede. O que é especialmente útil quando o problema não é linearmente separável. Neurônios de camadas ocultas tipicamente utilizam a função sigmóide, mas também pode ser utilizada uma função tangente hiperbólica.

O treinamento de uma rede neural é feito por retropropagação, ou *backpropagation*, que consiste em passar os exemplos pela rede, calcular o erro entre a saída e o valor objetivo do exemplo e depois voltar pela rede atualizando os pesos de forma a minimizar o erro. Para m saídas, o erro é calculado da seguinte forma:

$$E = \frac{1}{2}\sum_{j=1}^{m}(y_j - V_j)^2 \tag{3.19}$$

onde V_j é o valor objetivo do exemplo j e y_j é o valor da saída da rede para o mesmo exemplo. Para atualizar os pesos, é utilizada a regra delta:

$$\omega_{j,i} = \omega_{j,i} - \eta\delta_j y_i \tag{3.20}$$

onde $\omega_{j,i}$ é o delta anterior, η é a taxa de aprendizado - normalmente contida no intervalo [0.01, 0.7] -, δ_j é o erro e y_i é a saída. Quanto menor a taxa de aprendizado, mais lenta é a convergência. Se a taxa de aprendizado é alta, o algoritmo pode não convergir. O erro pode ser calculado de duas formas:

$$\delta_{saida} = -(y_{saida} - V)f'(z_{saida}) \tag{3.21}$$

$$\delta_j = g'(z_j)\sum_{i=1}^{N}\delta_i\omega_{i,j} \tag{3.22}$$

O erro pode ser retropropagado após a rede passar por todos os exemplos (*batch*) ou após a passagem por cada exemplo. Cada vez que os exemplos passam pela

rede corresponde a uma **época**. Exemplos podem ser passados para rede mais de uma vez, gerando várias épocas. A quantidade de épocas pode levar a problemas como **overfitting** (superaquedação) - quando a rede "treina demais", se viciando nos exemplos e perdendo a generalização - e **underfitting** (sub adequação) - quando a rede não treina o suficiente para ajustar adequadamente seus pesos. Uma solução para estes problemas é o **early stopping**: parar assim que o erro no conjunto de validação começar a crescer.

4 COLEÇÃO DE DADOS

Este capítulo descreve como foi realizada a coleta de dados e os critérios para integrar um *tweet* à base de dados. Todo o fluxo (Figura 4.1) foi desenvolvido a partir de amostras coletadas no início do trabalho, que são apresentadas na Tabela 4.1, e é baseada em tentar selecionar apenas textos que possam realmente relatar um assalto, um furto ou uma tentativa de assalto ou furto.

Para extração dos *tweets*, foi preciso integrar o trabalho à API do Twitter. Essa extração é feita através do uso de uma lista de palavras-chave como "assalto", "assaltado", "assaltada", "roubado", "roubada", "furtado" e "furtada". Após essa coleta, são aplicados filtros para garantir que os textos coletados estão de acordo com a proposta do trabalho. *Tweets* que não estão de acordo com os critérios são marcados como negativos. Ao fim da filtragem, os dados são armazenados no banco de dados. Para este trabalho, o serviço de banco de dados escolhido para armazenar os dados obtidos foi o MongoDB [1].

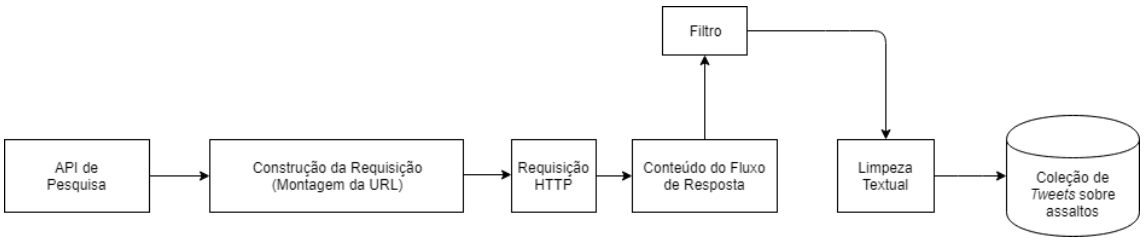

Figura 4.1: Fluxo utilizado para criação da base de dados.

4.1 EXTRAÇÃO DE DADOS DO TWITTER

Foi implementado um *crawler* que faz uso da metodologia de busca baseada em uma API para montar o conjunto de dados utilizado durante o desenvolvimento da aplicação. A API é disponibilizada pelo próprio Twitter e permite a coleta de *tweets* por meio de requisições ao servidor. No caso deste trabalho, o serviço foi utilizado

[1] https://www.mongodb.com/

Usuário 1: Fui roubado na sexta feira em Vila Isabel, levaram meu veículo pálio preto de duas portas e placa XXX000. Favor ligar para XXXXX-XXXX.
Usuário 2: 2 vagabundos atacando passageiros pelas janelas abertas dos coletivos na pista do meio em frente a Prefeitura do Rio, na Cidade Nova.
Usuário 3: Eu estava aqui pensando: Já fui assaltado nas zonas Norte (Irajá e Thomaz Coelho), Sul (Copacabana) e Central (Largo da Carioca)... Só falta a zona Oeste para completar a cidade.
Usuário 4: Fui assaltado 3 vezes em menos de 1 mês, pelo mesmo bandido.
Usuário 5: Todo mundo foi quase assaltado hoje na frente da escola, na rua. É tão triste saber que onde a gente pisa não é mais a mesma coisa, não temos segurança nem na escola.

Tabela 4.1: Exemplos de como os usuários da plataforma informam que foram assaltados.

apenas para a coleta de dados, mas ele também pode ser utilizado para publicar e gerenciar informações de um perfil na plataforma.

Antes de fazer uso do envio de requisições, é necessário criar uma aplicação para o projeto no Twitter Apps [2]. Para isso, basta entrar com *login* e senha de perfil na plataforma e clicar em "*Create New App*" [Criar Nova Aplicação]. Haverá um direcionamento para uma tela de cadastro onde é necessário registrar nome, descrição e *website*, uma *home page* ou outro serviço, para outros usuários baixarem e/ou fazerem uso da aplicação que utilizará a API. Esta primeira etapa é essencial porque irá garantir as credenciais de acesso, que estão disponíveis na aba "*Keys and Access Tokens*" [Chaves e Tokens de Acesso] encontrados ao clicar no nome da aplicação. Ao visualizar a aplicação, também é possível modificar o tipo de permissão de acesso, que, no caso deste trabalho, foi utilizado o nível de acesso a leitura, mas também estão disponíveis outros níveis de acesso: nível para leitura e escrita, escrita e mensagens diretas, quando um usuário deseja enviar uma mensagem privada para outro usuário. Também é possível testar se o acesso à API foi corretamente estabelecido na mesma página.

Neste trabalho, não foi utilizada nenhuma biblioteca de acesso à API, como

[2]https://apps.twitter.com/

Tweepy [3] ou Twitter4J [4], por exemplo. As requisições são feitas através de OAuth [5], um serviço projetado para trabalhar com o Protocolo de Transferência de Hipertexto (HTTP), que fornece um acesso seguro aos recursos do servidor, permitindo o acesso de terceiros aos recursos sem compartilhar suas credenciais. As URLs (*Uniform Resource Locator* - Localizador Uniforme de Recursos) são montadas de acordo com parâmetros definidos pelo pesquisador, levando em conta o tipo de *tweet* desejado na resposta. A resposta para cada requisição é um vetor de até 180 textos em formato JSON (*JavaScript Object Notation* - Notação de Objetos JavaScript). Os limites de taxa são divididos em janelas de 15 minutos e possuem intervalos iniciais disponíveis para solicitações GET de 15 requisições a cada 15 minutos e de 180 requisições a cada 15 minutos.

4.2 CRIAÇÃO DA BASE DE DADOS

Antes de falar da base de dados propriamente dita, vejamos uma breve descrição do que é enviado como resposta do servidor. A API retorna uma lista de objetos em formato JSON e, para cada elemento desta lista, são encontradas informações como a data de postagem do *tweet*, sua identificação (ID), informações de quem realizou a postagem (nome, nome de perfil, localização e descrição informados pelo próprio usuário), de qual lugar essa mensagem foi postada (informado através de um serviço de geolocalização), o texto da mensagem e algumas outras informações.

Dado que já são conhecidas as propriedades de um *tweet*, é possível fazer uma primeira filtragem nos dados de resposta no próprio envio de requisições à API. A busca de *tweets* é realizada através de uma pesquisa por palavras, o que faz do parâmetro "q", de *query* [Consulta], o único parâmetro obrigatório a ser preenchido na URL de requisição. É por ele que são enviadas as palavras-chave utilizadas para a extração. Neste trabalho, só serão utilizados textos em português, então um dos parâmetros também passados na URL é o de idioma, atribuído como "pt". Também é enviado o parâmetro de geolocalização e este recebe um conjunto de três elementos:

[3] http://www.tweepy.org/
[4] http://twitter4j.org/en/index.html
[5] https://oauth.net/

latitude, longitude e raio; que torna possível extrair postagens feitas apenas ao redor de um ponto específico limitado por um raio de cobertura. As coordenadas da cidade do Rio de Janeiro foram fornecidas pelo Google Maps [6] e o raio de cobertura utilizado é de 70 km. Portanto, pode-se dizer que o primeiro filtro é feito através da própria requisição à API.

Para cada objeto que chega no vetor de respostas, é preciso verificar se este é um *retweet*. *Retweets* são clones de um *tweet* que talvez já possa ser encontrado na resposta e devem ser descartados. Para "retweetar" uma mensagem, pode-se utilizar o botão de "retweet" no *tweet* que se deseja replicar, ou usar a construção como no exemplo da Figura 4.2. Neste exemplo, a postagem original foi feita pelo perfil do G1, perfil do portal de notícias da Globo que possui "g1" como seu nome de usuário. Junto às propriedades do objeto JSON da resposta, há também uma *flag* (um indicador) para identificar *retweets*. Caso o usuário que deseja replicar a mensagem utilize o botão de *retweet*, a *flag* chamada "retweeted" possui valor "verdadeiro". Se o usuário repassa uma mensagem para seus seguidores como na Figura 4.2, o campo possui valor "falso" e deve-se aplicar uma outra regra para descartá-lo. Graças ao recente aumento de 140 para até 280 caracteres por *tweet* na última atualização do Twitter, os *tweets* que são replicações aparecem truncados, mesmo quando outro parâmetro de busca é utilizado na requisição ($tweet\_mode = extend$) para garantir que textos de até 280 caracteres sejam retornados. Para evitar duplicação de denúncias de assaltos e *tweets* pela metade, *retweets* são eliminados de acordo com a *flag* ou ao ser identificada a presença da construção "RT @" em algum lugar do texto.

A próxima etapa para eliminar textos que não são interessantes para pesquisa é verificar se o texto com certeza não relata um assalto. Isto é feito por uma busca por palavras e sentenças que configuram que alguém não foi vítima de um crime. É verificado se frases como "nunca fui assalto", "nunca fui roubado", e outras variações observadas em amostras coletadas no início do trabalho, aparecem nos textos retornados. Esses *tweets* não são utilizados e são marcados como negativos (recebem valor um) em um novo atributo, criado pela aplicação, intitulado "*classification*"

[6] https://www.google.com.br/maps

Figura 4.2: No exemplo de *retweet*, a mensagem aparece logo após a menção ao perfil.

["classificação"]. Também é possível encontrar sentenças como "quase fui assaltado", mas, neste caso, o texto é mantido, já que relata uma situação de perigo e pode ser usado para analisar quantas vezes usuários se sentiram ameaçados.

Para a persistência na base de dados, os *tweets* precisam citar pelo menos um bairro do município. Uma forma de identificar onde usuários relataram assaltos ou sensação de perigo seria utilizar o campo de geolocalização do *tweet*, porém apenas 1.24% das postagens possuem este serviço de localização ativo [22]. Uma enquete rápida feita com um grupo de usuários na própria rede social determinou que a maioria não permite que o Twitter mostre suas coordenadas de localização por motivos de segurança. Outros não conheciam o serviço, que também não parece funcionar corretamente. Durante alguns testes com o serviço habilitado, o campo de coordenadas ainda continuava vazio, mesmo dando acesso à localização ao aplicativo e passado o tempo de 30 minutos que a API precisa para coletar os dados. Como não há garantia de que o serviço realmente funciona e também por ser desprezado pelos usuários, foi utilizado um dicionário de frequências contendo todos os bairros da cidade. Como os bairros "Barra da Tijuca" e "Vasco da Gama" são mais conhecidos pelas reduções "Barra" e "Vasco", outros apelidos também foram levados em consideração e adicionados ao dicionário que contém 174 lugares. O formato de dicionário é útil para contar quantas vezes um bairro é mencionado. Cada vez que um nome de bairro (chave) é encontrado em um *tweet*, sua frequência (valor) é incrementada.

A maioria dos *tweets* não menciona bairros, porém muitas vezes os usuários contam onde foram assaltados em resposta a outro usuário, como mostra o exemplo da Figura 4.3. Por isso, foi realizada a experiência de coletar conversas entre usuários para verificar se há menção a algum bairro do Rio de Janeiro. A API do Twitter disponibiliza um serviço para coletar conversas, mas como este não é gratuito, a única forma possível de coletar conversas seria pesquisando pelos *usernames* [Nomes de usuário] dos indivíduos envolvidos na conversa. Por exemplo, caso um usuário "Fulano" disse ter sido assaltado, é feita uma busca por "@Fulano" (pelas últimas menções àquele usuário) para então verificar se alguém o mencionou e utilizar o nome desse outro usuário para buscar as respostas do usuário "Fulano" e assim verificar se nestas respostas são mencionados os bairros. O grande problema dessa abordagem, além do grande número de requisições, é que não existe garantia de que serão retornadas rapidamente as menções que procuramos. Seriam necessárias muitas requisições à API, que tem um limite de requisições, para encontrar informações dentre usuários que postam muito na plataforma ou que são muito mencionados. Isto fez com que a ideia fosse descartada e, como foram encontrados poucos *tweets* contendo nomes de bairro, uma outra coleção com os demais *tweets* também foi criada para que os dados não fossem totalmente descartados.

Após o filtro dos bairros, os dados passam por uma limpeza textual. Nesta etapa, são removidas menções [Exemplo 4.3] - quando dois ou mais usuários trocam mensagens entre si através de um "@" atrelado ao nome de usuário que desejam responder -, todo e qualquer tipo de link, diferentes tipos de risadas, caracteres especiais (parênteses, colchetes, chaves, etc), pontuação, *emojis*, e *hashtags* [Exemplo 4.4] - utilizadas quando usuários desejam comentar ou dar apoio a um assunto. Menções, caracteres especiais, links e *hashtags* são removidos com auxílio de expressões regulares. Pontuação é removida através do uso de métodos da biblioteca NLTK (Natural Language Toolkit) [7]. Já os *emojis*, pelo formato *unicode*, são removidos com ajuda da biblioteca emoji [8]. Ambas as bibliotecas citadas foram utilizadas em códigos Python. Também foi utilizado um dicionário pré-estabelecido de 129

[7] https://www.nltk.org/

[8] https://pypi.org/project/emoji/

Figura 4.3: Usuários trocam informações sobre o assalto.

abreviações para que as abreviações encontradas fossem substituídas pela palavra completa.

Figura 4.4: Exemplos de uso de *hashtags* para agrupar *tweets* sobre a Copa na Rússia e *tweets* que citam a empresa McDonalds.

4.3 ARMAZENAMENTO

Tweets são dados complicados de serem armazenados em banco de dados relacionais. A melhor forma de armazená-los seria utilizar um banco de dados NoSQL.

Por sua garantia de melhor performance, este tipo de armazenamento também foi adotado pelo Twitter devido ao aumento de volume de dados. No caso da rede social, são utilizados diversos serviços de armazenamento de dados, como Cassandra [9], MySQL [10] e FlockDB [11].

Como os dados fornecidos pela API são no formato chave e valor, armazená-los em um banco de dados não-relacional seria uma tarefa muito mais simples. Além do fato de que banco de dados NoSQL são recomendados para aplicações que realizam updates e consultas excessivas [29]. Como os *tweets* estão disponíveis por até uma semana através da API pública, é muito mais coerente realizar updates pelo ID, utilizando *upsert*, do que correr o risco de inserir novamente na coleção um texto que já faz parte da mesma. Vale citar que não estão sendo acompanhados *tweets* de pessoas específicas, mas sim qualquer *tweet* que contenha uma das palavras-chave.

Para este trabalho, foi escolhido o banco de dados MongoDB, que é um banco de dados de código aberto, gratuito, de alta performance, sem esquemas e orientado à documentos. Como foi escrito em C++, é portável para diferentes sistemas operacionais [28]. Neste tipo de banco, existem coleções de documentos em que cada documento é autossuficiente, contendo todos os dados que possam ser necessários. Diferente do conceito de não repetição e chaves estrangeiras encontrado no modelo relacional, este tipo de banco foi criado para que não sejam necessários *joins*, que prejudicam consideravelmente a performance das consultas. Apenas com uma chave primária é retornado tudo que é necessário.

A principal vantagem em utilizar o MongoDB é que ele permite o armazenamento direto de objetos em formato JSON, justamente o tipo de documento retornado pela API do Twitter, logo todo o *tweet* poder ser armazenado assim que é obtida a resposta da requisição. Além disso, outra vantagem importante é não ser necessário modificar a estrutura de armazenamento caso a API seja modificada (caso haja adição de novos campos ao objeto, por exemplo) e o serviço também garante respostas de consultas muito mais rápidas do que no modelo relacional.

[9] http://cassandra.apache.org/

[10] https://www.mysql.com/

[11] https://github.com/twitter-archive/flockdb

4.4 CRIAÇÃO DA BASE ANOTADA

Com intuito de montar uma base anotada de *tweets* que denunciam ou não um assalto, foram selecionados 500 dos 1000 primeiros *tweets* coletados no início deste trabalho. Como se tratava de um conjunto complexo, com muitos ruídos e as mesmas palavras nos dois tipos de classificação, foi necessário recorrer a classificação por humanos.

O processo de classificação foi realizado por 19 colaboradores que, ao longo de duas semanas, receberam como orientação classificar como positivos apenas *tweets* que acreditavam relatar um assalto, roubo ou furto. Também foi mantido o critério de que *tweets* que informam que alguém foi quase assaltado poderiam ser considerados *tweets* positivos. Um exemplo de *tweet* desse conjunto que poderia ser considerado positivo é: "Fui assaltado. Dois pivetes cada um montado numa BMX, me fecharam na rua." E um negativo é: "Acabei de andar da linha do trem até a minha casa sem ser assaltado. Que decepção". Para este conjunto, a classificação é feita de acordo com a opinião de cada colaborador, por isso foi muito importante que um mesmo *tweet* fosse classificado por diferentes colaboradores, como em uma espécie de votação.

Cada *tweet* foi classificado por sete colaboradores diferentes. A grande maioria das classificações foi feita de forma unânime, foram poucas opiniões divergentes sobre qual deveria ser a classificação de um documento. O número ímpar de classificações poderia ser útil para evitar empates, porém quando aconteceu de um *tweet* receber dois tipos de classificação de diferentes colaboradores, mais da metade dos votos foram para uma mesma classificação. Em alguns casos, foram cinco votos para uma classificação positiva e apenas dois para uma classificação negativa, portanto possíveis empates não atrapalharam.

Ao final do processo, cada voto foi computado e o número que apareceu mais vezes (0 para positivo ou 1 para negativo) foi adotado como a classificação do *tweet*. Foram classificados 292 *tweets* como negativos (58% do conjunto) e 208 como positivos (42% do conjunto). Para total entendimento durante o processo de classificação

pelos colaboradores, foram utilizados dados crus, que ainda não haviam passado pelo pré-processamento.

4.5 REPRESENTAÇÃO DOS *TWEETS*

Partindo do princípio que nem todos os textos coletados durante este trabalho talvez cumprissem o requisito de relatar um assalto, foi necessário buscar uma forma de identificar esses *tweets* que, mesmo contendo alguma das palavras-chave, não se encaixam nas exigências do projeto. Uma forma de identificar esses *tweets* é utilizar a base de dados anotada para treinar um classificador.

Como os métodos mais utilizados em artigos de mineração de opinião e análise de sentimento utilizam regressão, foi necessário representar os 500 *tweets* numericamente para realização do experimento. Durante o experimento, cada *tweet* é representado como um documento que deve ser representado pelo seu conteúdo. Na indexação, os documentos passam pela fase de pré-processamento e depois se inicia o processo de ponderação TF-IDF. Após a ponderação, é feita a normalização pelo tamanho dos documentos.

Após esses dois processos, temos a matriz TF-IDF. Na matriz, cada linha representa um *tweet* e cada coluna um termo no *corpus*. A célula é preenchida com zero se um termo não aparece em um *tweet* e é preenchida com o valor TF-IDF correspondente se este está presente no documento. Foram identificados 2066 termos distintos, gerando uma base de vetores de ordem 500x2066. Nesse conjunto, o menor documento possui um único termo e o maior possui 35 termos. Há uma média de 11 palavras por documento, o que dificulta que um *tweet* seja caracterizado por seus termos, já que temos a média de poucas palavras.

Como não há garantia de que a normalização pelo tamanho de documentos resulta em melhores resultados, foram geradas duas bases distintas: uma normalizada e outra não normalizada. A fim de comparar o desempenho dos métodos diante dos dois tipos de dados.

5 EXPERIMENTOS E RESULTADOS

Quando se deseja trabalhar com um tipo de informação específica, realizar uma busca por palavras-chave para coletar dados da Internet possivelmente não será suficiente para retornar o tipo de resposta desejada. Neste trabalho, o local de onde são retirados esses dados é um agravante. Quando coletamos dados de redes sociais como o Twitter, onde os *tweets* produzidos não tem obrigatoriamente um conteúdo sério, é possível que a maioria dos textos do banco de dados não contenham as informações esperadas. Para identificar textos que realmente são denúncias de assalto, foi realizada a experiência de treinar três métodos de aprendizado diferentes. Um que utiliza geometria, um baseado em probabilidades e um modelo estatístico. Todos foram inicialmente testados com os dois tipos de base (normalizada e não normalizada), sendo escolhida para ser utilizada a base com a qual os métodos tiveram melhor desempenho. Para cada um dos métodos citados, foram aplicados K-Fold e SVD na tentativa de alcançar melhores resultados.

Há uma grande variedade de métodos de aprendizado com diferentes propostas para classificar um documento. Neste trabalho, foram utilizados os três métodos mais citados em artigos usados como referência: SVM [15], Algoritmo de Classificação Naive Bayes [15] e Redes Neurais Artificiais [25]. Como são métodos de aprendizado supervisionado, foram separados 335 *tweets* para treinamento, o que representa 67% do conjunto, e 165 *tweets* para validação dos modelos treinados. Antes de iniciar o experimento propriamente dito, foi preciso entender melhor a natureza dos dados após a ponderação. Para isso, foi utilizado o algoritmo de particionamento K-Means [15], que permite avaliar se *tweets* positivos e negativos são diferentes e claramente separáveis em dois grupos distintos.

O algoritmo de validação K-Fold [17] foi utilizado para avaliar a capacidade de generalização de cada método utilizando dois valores para k. Primeiro, foram gerados 5 subconjuntos de tamanho 100 para validação, o que resultou em 5 subconjuntos de 400 *tweets* para treinamento. O próximo passo foi definir 10 subconjuntos de tamanho 50 para validação e usar 10 subconjuntos de 450 *tweets* para treinamento. Para cada um dos valores de k, foram calculadas as médias de cada métrica de avaliação

e seus desvios-padrão.

O algoritmo de decomposição SVD foi utilizado porque alguns dos algoritmos escolhidos são sensíveis a ruídos e valores atípicos. O algoritmo é aplicado na tentativa de melhorar a qualidade dos dados e assim melhorar o desempenho dos métodos de classificação. Como se trata de um método que trabalha com projeções sobre o hiperplano formado pela dimensão dos exemplos, o máximo que podemos reduzir o conjunto de dados é uma matriz quadrada 500x500. Para testar diferentes quantidades de atributos, o valor da segunda dimensão variou de 50 à 500 durante os testes. A cada iteração, a quantidade de atributos era aumentada em 10 até que o valor final fosse atingido. Os valores de início e o passo a cada iteração foram escolhidos de forma arbitrária.

5.1 PARTICIONAMENTO DOS DADOS USANDO K-MEANS

A importância de segmentar *tweets* pouco semelhantes está em verificar se estamos lidando com uma base de dados heterogênea. Em uma situação ideal, o conjunto de dados deveria ser claramente separável em *tweets* positivos e negativos. Como queremos separar os documentos em duas classes, o fracionamento foi feito em duas categorias (dois *clusters*) para as bases normalizada e não normalizada. O algoritmo escolhe seus *clusters* iniciais de forma aleatória, então foi preciso executá-lo várias vezes para a mesma base para tentar obter resultados melhores. A implementação utilizada do K-Means [1] é da biblioteca Scikit-Learn.

Na Tabela 5.1, estão apresentados os resultados do algoritmo diante de ambas as bases. Quando utilizada a base não normalizada, o algoritmo demonstrou ter alta similaridade entre quase todos os 500 *tweets*, agrupando quase todos em um mesmo grupo, exceto por um único *tweet* que foi agrupado sozinho. O primeiro grupo chamaremos de grupo A e o segundo chamaremos de grupo B. A contagem de positivos e negativos em cada grupo foi feita de acordo com a classificação citada na Seção 4.4.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

	Base Não Normalizada		Base Normalizada	
	Grupo A	Grupo B	Grupo A	Grupo B
Positivos	208	0	197	11
Negativos	291	1	275	17

Tabela 5.1: Resultado do agrupamento das duas bases utilizando K-Means.

Os dados também não são claramente separáveis para o algoritmo quando utilizada a base normalizada. Desta vez, foram agrupados 472 *tweets* no grupo A e 28 no grupo B. Analisando os *tweets* do grupo A, 197 são positivos e 275 negativos. A quantidade de *tweets* negativos também é maior no grupo B: foram 17 negativos e 11 positivos. Ao analisar os resultados para a base não normalizada, dos 499 *tweets* agrupados no grupo A, 291 eram negativos, assim como o único *tweet* do grupo B, e 208 eram positivos. Para a base de dados não normalizada, a separação manteve-se sempre a mesma para diferentes execuções do algoritmo.

O fato do algoritmo sempre agrupar grande parte dos dados em um mesmo grupo chamou a atenção para a representação dos dados na base, que parece ser altamente homogênea. Fazendo um estudo mais profundo da base de dados, foi notado que muitos termos aparecem em poucos documentos, sendo que 1444 termos só aparecem uma única vez em todo o *corpus*. Isto resulta em representações muito parecidas dos atributos, o que dificulta a separação quando utilizado um algoritmo que agrupa dados de acordo com a distância euclidiana média entre os valores desses atributos. Por isso, é compreensível que a maioria dos *tweets* seja agrupada em torno de um mesmo *cluster*. A distribuição de frequências por termo está apresentada na Figura 5.1.

Na tentativa de facilitar o trabalho do algoritmo e tentar obter um melhor particionamento, foram removidos da base os termos que só aparecem uma única vez considerados pouco relevantes. Termos pouco relevantes são termos que normalmente não são relacionados a assaltos em nenhum contexto, portanto removê-los não acarreta em perda de informação. Foram removidos termos como “veia”, “dura”, “meta”, “fundo” e “derretendo”. A remoção garantiu uma pequena melhora na sepa-

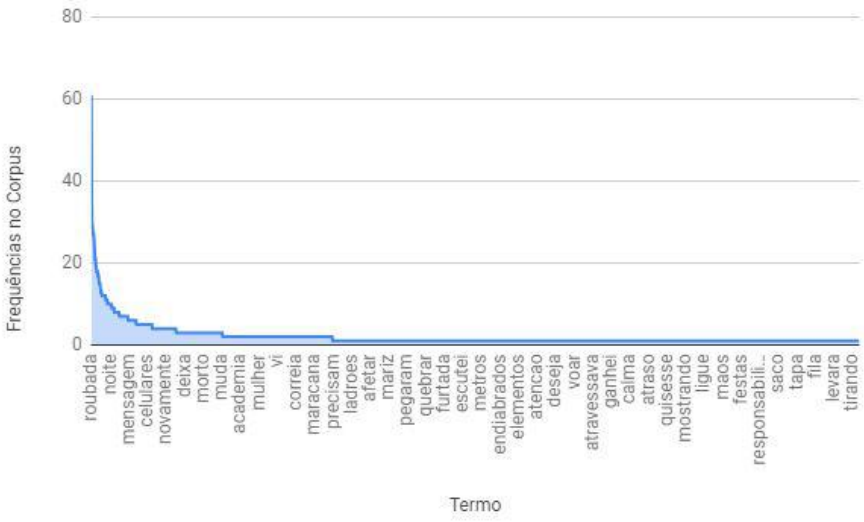

Figura 5.1: Número de ocorrências de um termo em toda base de dados.

ração, mas não tão expressiva quanto o esperado. Remover todos os termos que só aparecem uma única vez no conjunto também não é uma boa solução, pois aproximadamente 70% dos termos só aparecem uma única vez no *corpus*. A Figura 5.2 apresenta as 15 palavras que aparecem em mais documentos. Mesmo estas sendo as palavras mais comuns, costumam aparecer uma única vez por documento, o que implica em representações iguais da palavra "roubada" em 61 documentos, por exemplo. O problema não é resolvido em nenhum dos dois casos.

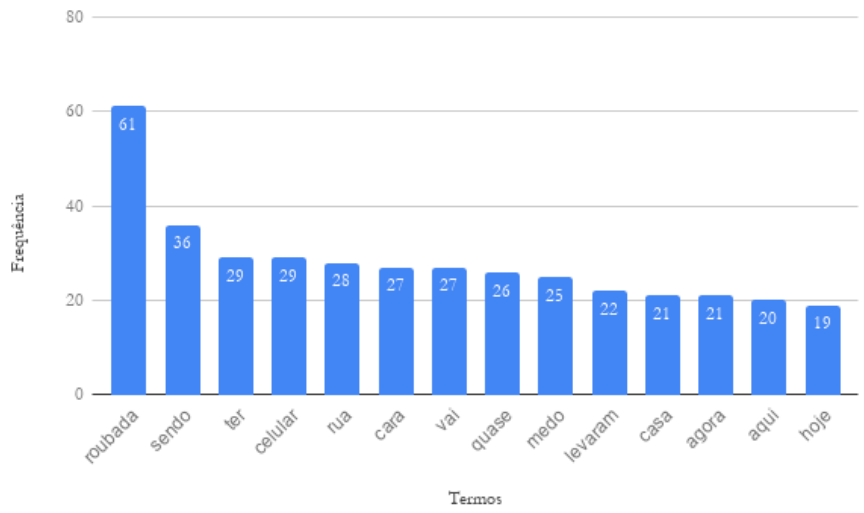

Figura 5.2: 15 termos mais comuns e o número de ocorrências em toda a base de dados.

	Base Não Normalizada		Base Normalizada	
	Grupo A	Grupo B	Grupo A	Grupo B
Positivos	196	12	185	23
Negativos	278	14	251	41

Tabela 5.2: *Tweets* agrupados pelo algoritmo K-Means para as bases não normalizada e normalizada usando SVD.

O K-Means é extremamente sensível a ruídos, por isso utilizar o SVD para representar os dados de uma forma mais eficiente é uma boa tentativa de melhorar o particionamento. Os resultados encontrados são apresentados na Tabela 5.2. Ainda que não seja uma melhora expressiva, o algoritmo consegue agrupar mais *tweets* em um segundo grupo após a redução de atributos. Para a base não normalizada, o melhor resultado foi obtido utilizando uma matriz de 500 exemplos e 450 atributos. Foram agrupados 474 *tweets* no grupo A e 26 no grupo B, com uma divisão entre 196 *tweets* positivos e 278 negativos no grupo A e 14 *tweets* negativos e 12 positivos no grupo B. Também houve uma melhora no particionamento de dados normalizados, alcançada utilizando uma matriz 500x300, 500 exemplos e 300 atributos. Foram 439 *tweets* no grupo A, sendo 251 negativos e 185 positivos. Nos 64 tweets do grupo B, foram agrupados 23 positivos e 41 negativos. Por fim, a Tabela 5.3 compara os resultados de todos os experimentos.

Mesmo melhorando a qualidade dos dados e eliminando termos poucos frequentes e pouco relevantes, o método tende sempre a agrupar a maioria dos dados em torno de um mesmo *cluster*. Analisando os resultados obtidos, pode-se perceber que o algoritmo sempre agrupa mais negativos do que positivos. O que faz parecer que o algoritmo está agrupando *tweets* negativos com diferentes representações. Os resultados ainda podem variar bastante, já que estamos usando um método que escolhe seus centróides iniciais de forma aleatória, o que também acontece com a escolha da semente do SVD. Assim, podem ser obtidos diferentes resultados para um mesmo conjunto de dados e escolha de dimensão. Melhores resultados talvez possam ser obtidos através de testes exaustivos, da escolha de uma boa semente

	Base Não Normalizada				Base Normalizada			
	Sem SVD		Com SVD		Sem SVD		Com SVD	
	Grupo A	Grupo B	Grupo A	Grupo B	Grupo A	Grupo B	Grupo A	Grupo B
Positivos	208	0	196	12	197	11	185	23
Negativos	291	1	278	14	275	17	251	41

Tabela 5.3: Resultados do particionamento das duas bases com e sem uso do SVD.

para o SVD e de centróides iniciais bastante representativos. Entretanto, ter um conjunto de dados bastante homogêneo não indica que teremos um mau desempenho dos algoritmos de classificação.

5.2 CLASSIFICADOR UTILIZANDO SVM

Para os testes com o SVM, foi utilizada a implementação SVC [2] (*Support Vector Classification*) disponível na biblioteca Scikit-Learn. Esta implementação é baseada na LIBSVM [3], uma biblioteca *open source* muito popular implementada em C++, que utiliza validação cruzada para seleção de modelos, possui diferentes tipos de kernel e uma implementação ponderada para dados não balanceados.

Os primeiros testes foram realizados com o kernel "rbf" e coeficiente 1.0. Com esta configuração, que é a configuração padrão dos parâmetros, o método apresentou desempenho ruim para ambas as bases, já que classificou todos os *tweets* do conjunto de validação como negativos. Para melhorar os resultados, foi necessário estimar melhor os parâmetros e, como a implementação do SVC possui vários parâmetros que podem ser ajustados, foi utilizado o algoritmo GridSearchCV [4] para encontrar a melhor configuração possível destes parâmetros. O algoritmo implementa uma busca exaustiva para determinar os parâmetros do melhor modelo possível. É uma busca feita através de validação cruzada e uma "grade" de parâmetros a serem testados. Os parâmetros da grade são definidos pelo usuário e, neste caso, foram testados dois parâmetros: o parâmetro de penalidade e o tipo da função kernel. Para o

[2]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[3]https://www.csie.ntu.edu.tw/ cjlin/libsvm/

[4]https://scikit-learn.org/0.16/modules/generated/sklearn.grid_search.GridSearchCV.html

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.76	0.75	0.59	0.87
2	0.77	0.79	0.63	0.85
3	0.71	0.67	0.54	0.86
4	0.72	0.68	0.50	0.92
5	0.82	0.83	0.62	0.92
Média	**0.75**	**0.74**	**0.57**	**0.88**
Desvio Padrão	**0.04**	**0.06**	**0.05**	**0.03**

Tabela 5.4: Resultados do SVM utilizando K-Fold para K=5.

parâmetro de penalidade, foram testados valores de 1.0 a 10.0 e, para a função kernel, foram testados os kernels linear, sigmóide, polinomial e "rbf". Para os tipos de kernel que não são do tipo linear, foi definido também um intervalo de 10^{-3} a 10^{-5} para o parâmetro *gamma*, que é usado como medida de similaridade entre pontos. Os dados a serem testados foram escolhidos segundo recomendações da própria documentação do GridSearchCV. Ao final dos testes, o modelo de melhor precisão tinha kernel linear e parâmetro de penalidade 1.0.

Apenas a mudança de kernel já demonstrou uma melhora excepcional nos resultados. Foram obtidos 73% de acurácia, 75% de precisão, 59% de sensibilidade e 82% de especificidade com os dados não normalizados. Os resultados são ainda melhores para a base normalizada, já que foram alcançados 79% de acurácia e 81% de precisão e as medidas de sensibilidade e especificidade foram de 69% e 85%, respectivamente. A partir dos resultados para a base normalizada, foi utilizado o algoritmo K-Fold. A Tabela 5.4 mostra o desempenho dos cinco modelos treinados na primeira execução do algoritmo, com o parâmetro $K = 5$. Os cinco modelos treinados alcançaram média de 75% de acurácia, 74% de precisão, 57% de sensibilidade e 88% de especificidade. Estas médias ficaram próximas das métricas do melhor modelo treinado, que alcançou 82% de acurácia, 83% de precisão, 62% de sensibilidade e 92% de especificidade.

Na Tabela 5.5, são apresentados os resultados quando definido $K = 10$. As

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.80	0.84	0.70	0.84
2	0.84	0.88	0.75	0.88
3	0.84	0.82	0.56	0.97
4	0.58	0.50	0.41	0.80
5	0.78	0.86	0.76	0.78
6	0.78	0.75	0.55	0.93
7	0.76	0.79	0.61	0.84
8	0.70	0.62	0.60	0.81
9	0.79	0.74	0.59	0.96
10	0.61	0.60	0.39	0.80
Média	**0.74**	**0.74**	**0.59**	**0.86**
Desvio Padrão	**0.08**	**0.12**	**0.12**	**0.06**

Tabela 5.5: Resultados do SVM utilizando K-Fold para K=10

médias dos modelos treinados não se afastam muito das médias dos cinco modelos anteriores. Neste caso, as médias foram de 74% de acurácia, 74% de precisão, 59% de sensibilidade e 86% de especificidade. Também podemos encontrar o modelo com melhor desempenho durante os testes, que alcançou 84% de acurácia, 88% de precisão, 75% de sensibilidade e 88% de especificidade.

Já os resultados de acurácia e precisão utilizando SVD estão apresentados na Figura 5.3. Ao variar o número de atributos, foi possível perceber que o método parece trabalhar melhor com dimensões maiores. Para acurácia, temos um primeiro pico nos resultados quando utilizados 410 atributos. Com essa quantidade de atributos, temos 78% de acurácia, e este valor se mantém para 420 termos. O desempenho diminui um pouco para 430 e 440 atributos, mas logo depois a maior acurácia é alcançada com uma matriz de 450 atributos. Nesta etapa da iteração, temos o melhor resultado com acurácia de 79%, que se mantém para os demais valores de dimensão.

O comportamento das medidas de precisão não é muito diferente do que acontece com a acurácia. No início, há uma variação maior entre as medidas conforme a

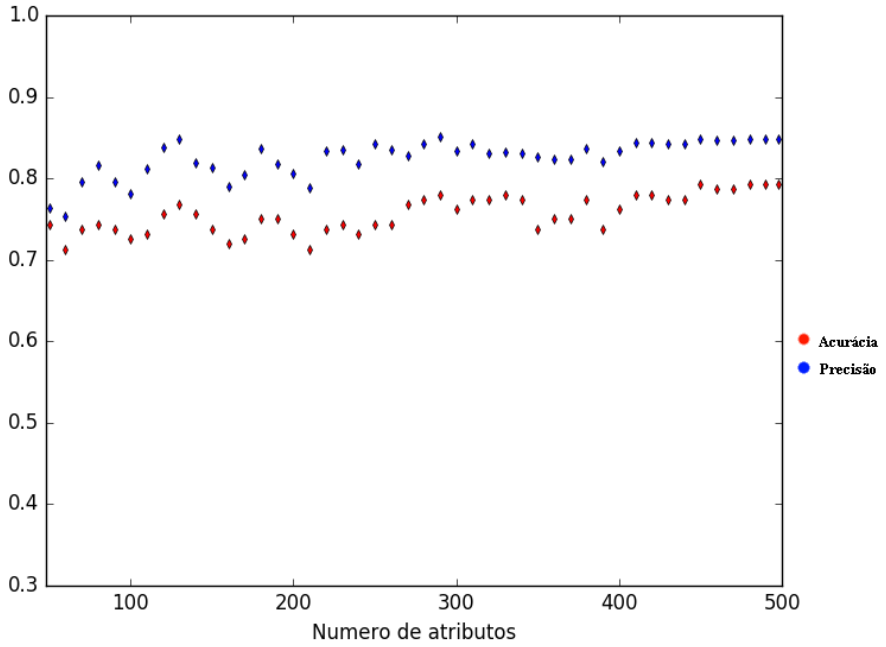

Figura 5.3: Medidas de acurácia e precisão após aplicar SVM em matrizes com diferentes números de atributos.

quantidade de atributos aumenta. A partir de 300 termos, essa variação diminui e as medidas se alternam em um intervalo de 81% a 84%. O melhor resultado, de 84% de precisão, é atingido para 410 atributos e é mantido até o final das iterações.

Os resultados das medidas de sensibilidade tem um comportamento diferente das duas medidas anteriores, como mostra a Figura 5.4. Não foram necessários muitos atributos para alcançar as melhores medidas de sensibilidade, que tem seu melhor resultado, de 73%, quando utilizada uma matriz com 130 atributos. Este resultado é mantido para 250 e 290 termos. A partir de 290, essa porcentagem começa a diminuir e só volta a aumentar com 410 atributos, atingindo 71%. Este valor é mantido até o final das iterações.

Assim como todos os testes anteriores, o algoritmo tende a acertar mais negativos do que positivos, resultado que permanece para quando são utilizadas matrizes reduzidas. As medidas de especificidade atingem valores acima de 70% para todas as dimensões testadas - tendo seu maior valor alcançado com 50 atributos, quando chegou a 87%. Comparando com os resultados das medidas de sensibilidade, temos a menor porcentagem de acertos positivos quando utilizados 50 atributos. Conforme

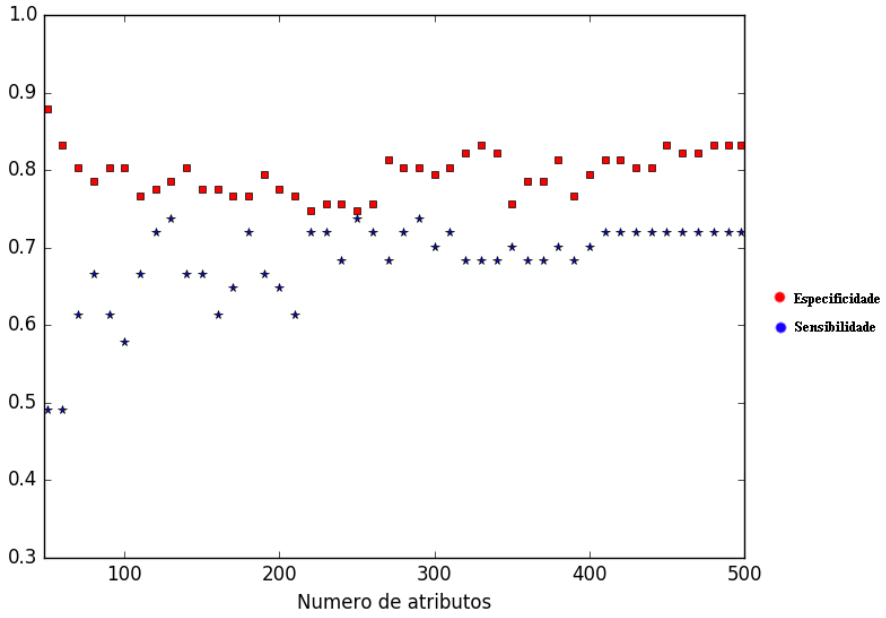

Figura 5.4: Medidas de sensibilidade e especificidade após aplicar SVM em matrizes com diferentes números de atributos.

a porcentagem de acertos negativos diminui, há um aumento na porcentagem de acertos positivos, o que aumenta a precisão. Com 80 atributos, a especificidade cai para 78%, há um aumento de 66% de sensibilidade e pela primeira vez temos 80% de precisão nos resultados. Assim, as melhores medidas de precisão são encontradas quando há um equilíbrio entre as medidas de sensibilidade e especificidade. Quando chegamos a um modelo com 85% de precisão e 78% de acurácia, temos também a melhor medida de sensibilidade encontrada durante os testes, que é de 73%. A especificidade tem uma queda de 7% em relação ao seu maior valor e atinge 80%.

Após essa análise, temos que o valor ideal para a segunda dimensão é de 290 atributos, que garantiu o último modelo citado acima. Entretanto, quando comparamos os resultados utilizando SVD com as medidas do melhor modelo treinado utilizando K-Fold, percebemos que não há melhorias no desempenho. Como mostra a Tabela 5.5, o melhor modelo treinado com a matriz original possui medidas maiores do que o melhor modelo treinado com uma matriz reduzida. Isso porque o SVM é resistente a *overfitting*, inclusive nos casos em que o número de atributos é muito maior do que o número de exemplos [5], por conta da regularização. O algoritmo

depende da distância da fronteira de decisão ao ponto mais próximo, mas é independente do número de atributos. Para evitar que o modelo treine demais e perca sua capacidade de generalização, é feita uma meticulosa estimativa dos parâmetros de regularização. SVMs não acabam com o problema de baixa generalização, mas ao invés de tratar do problema na função de adequação, a solução é encontrar um modelo com desempenho melhor apenas ajustando os parâmetros, como foi feito utilizando o GridSearchCV. O problema de *overfitting* ainda pode ocorrer caso a escolha dos parâmetros seja ruim.

5.3 CLASSIFICADOR UTILIZANDO NAIVE BAYES

Assim como para os métodos anteriores, também foram utilizadas implementações da biblioteca Scikit-Learn. No caso do algoritmo de classificação de Bayes, existem algumas variações disponíveis na biblioteca e duas delas foram utilizadas durante os testes: Naive Bayes Gaussiano [5] e Naive Bayes Multinomial [6]. Ainda há uma terceira implementação, chamada Bernoulli Naive Bayes [7], que é utilizada quando os atributos possuem valores binários. Como este não é o caso do conjunto de dados utilizado neste trabalho, este tipo de implementação não foi utilizado.

5.3.1 Naive Bayes Gaussiano

O Naive Bayes Gaussiano é um tipo de implementação do algoritmo de classificação de Bayes para valores contínuos ou distribuídos de forma normal (gaussiana). Como os dados deste trabalho são representados segundo a ponderação TF-IDF, temos uma matriz de valores contínuos e assim é possível aplicar o algoritmo.

Como o SVM, o classificador também possui um desempenho um pouco melhor quando utilizados os dados normalizados, ainda assim os resultados iniciais foram bem ruins. O método também obteve as piores médias quando utilizado o K-Fold, como mostram as tabelas 5.6 e 5.7, as médias dos modelos treinados não chegam a

[5] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.53	0.51	0.47	0.59
2	0.56	0.59	0.59	0.52
3	0.54	0.61	0.50	0.56
4	0.58	0.67	0.56	0.59
5	0.59	0.55	0.59	0.58
Média	**0.56**	**0.58**	**0.54**	**0.57**
Desvio Padrão	**0.02**	**0.05**	**0.08**	**0.05**

Tabela 5.6: Resultados Naive Bayes Gaussiano para K=5.

60%. O melhor modelo da primeira rodada atingiu apenas 58% de acurácia, 67% de precisão, 56% de sensibilidade e 59% de especificidade. Contrariando os resultados do SVM, o classificador de Bayes acerta mais exemplos positivos do que negativos na segunda rodada e o melhor modelo obteve apenas 56% de acurácia, 69% de precisão, 63% de sensibilidade e 51% de especificidade.

Realizando os testes com SVD e analisando as medidas de acurácia para diferentes dimensões, apresentadas na Figura 5.5, podemos notar que os valores começam a diminuir quando utilizada uma matriz com 350 atributos, o que pode ser diagnosticado com a Maldição de Dimensionalidade [31]: termo associado a vários fenômenos que surgem em análises de dados em espaços com muitas dimensões (atributos). Quanto maior a dimensionalidade do vetor, mais exemplos serão necessários para a aprendizagem do modelo. Adicionar novas características não significa que sempre haverá uma melhora no desempenho do classificador. Em torno de 350 atributos, o algoritmo já não possui o mesmo desempenho e começa a errar mais classificações. Mesmo ainda com alguns picos, a medida de acurácia já é bem menor do que a encontrada para dimensões menores e não chega a 60%. A maior porcentagem de acurácia é encontrada para 50, 90, 110 e 150 atributos e é de 69%. Após 150, o algoritmo apresenta o mesmo valor de 67% até chegar a 240, que é quando o desempenho começa a cair novamente. Apesar de já começar a cair, esta queda não é tão brusca quanto ao usar 350 atributos, onde a acurácia cai para 58%.

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.46	0.60	0.50	0.43
2	0.50	0.55	0.65	0.37
3	0.56	0.69	0.63	0.51
4	0.52	0.58	0.54	0.50
5	0.56	0.60	0.60	0.51
6	0.58	0.54	0.59	0.56
7	0.54	0.61	0.63	0.46
8	0.55	0.60	0.54	0.55
9	0.55	0.60	0.54	0.55
10	0.56	0.61	0.54	0.57
Média	**0.54**	**0.59**	**0.58**	**0.50**
Desvio Padrão	**0.03**	**0.03**	**0.04**	**0.06**

Tabela 5.7: Resultados do Naive Bayes Gaussiano para K=10.

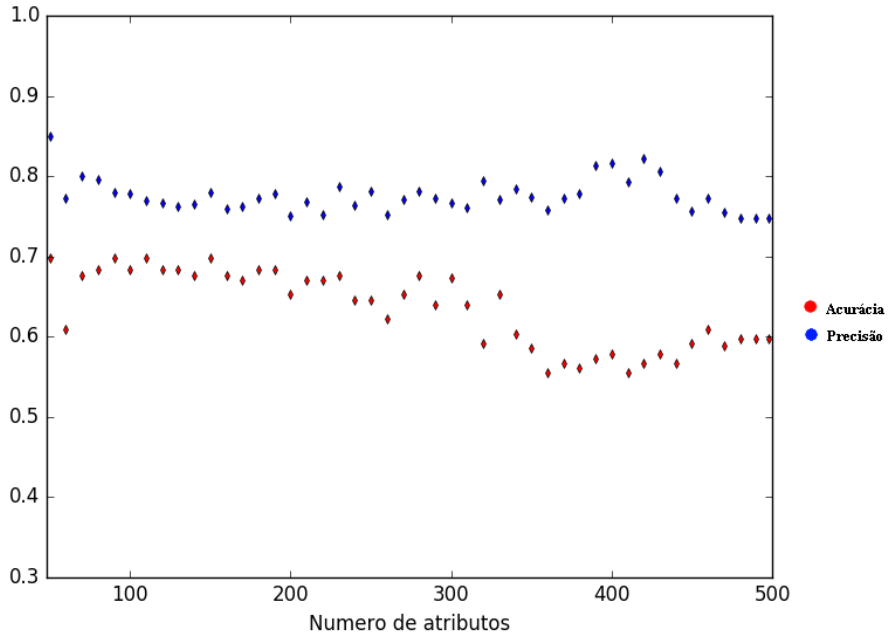

Figura 5.5: Medidas de acurácia e precisão após aplicar Naive Bayes Gaussiano em matrizes com diferentes números de atributos.

Mesmo que o desempenho do algoritmo piore conforme o número de atributos aumenta, as medidas de precisão não acompanham a queda das medidas de acurácia. O que acontece é que quando o número de atributos aumenta, o algoritmo tende a acertar mais exemplos positivos e a errar mais exemplos negativos. Isto mantém a precisão, porém diminui a acurácia.

Para alguns valores de dimensão, o algoritmo alcança 80% de sensibilidade, enquanto a medida de especificidade chega a 44%, como mostra a Figura 5.6. Como a porcentagem de acertos negativos é muito baixa, o algoritmo pode estar classificando de forma aleatória a maioria dos exemplos como positivos. É difícil escolher uma dimensão de termos que o método obteve melhor desempenho. As medidas de acurácia não chegam a 70% para nenhum dos casos. Avaliando apenas pela precisão, o maior valor é encontrado quando utilizados 50 atributos, que é de 85%, de um modelo com 69% de acurácia, 78% de sensibilidade e 63% de especificidade. Como o algoritmo de Bayes é um forte candidato a ser amaldiçoado pela maldição de dimensionalidade, faz sentido que seus melhores resultados sejam encontrados ao treinar o algoritmo com matrizes de dimensões menores. O modelo treinado com 50 atributos garante uma melhora significativa em relação ao melhor desempenho treinado utilizando o K-Fold, cujo melhor modelo obteve 56% de acurácia, 69% de precisão, 63% de sensibilidade e 51% de especificidade.

5.3.2 Naive Bayes Multinomial

Neste tipo de implementação, tem-se uma variante do algoritmo de Bayes para dados distribuídos de forma multinomial. Esta é uma das duas variantes clássicas do algoritmo usada para a classificação de textos e costuma funcionar muito bem com vetores TF-IDF.

Assim como o Naive Bayes Gaussiano, o Multinomial também teve um desempenho melhor com a base normalizada e obteve medidas melhores do que o método anterior: o método alcançou 73% de acurácia, 75% de precisão, 54% de sensibilidade e 85% de especificidade em um teste inicial. O algoritmo concorda com o SVM em acertar mais exemplos negativos do que positivos e isto fica claro quando utilizado o

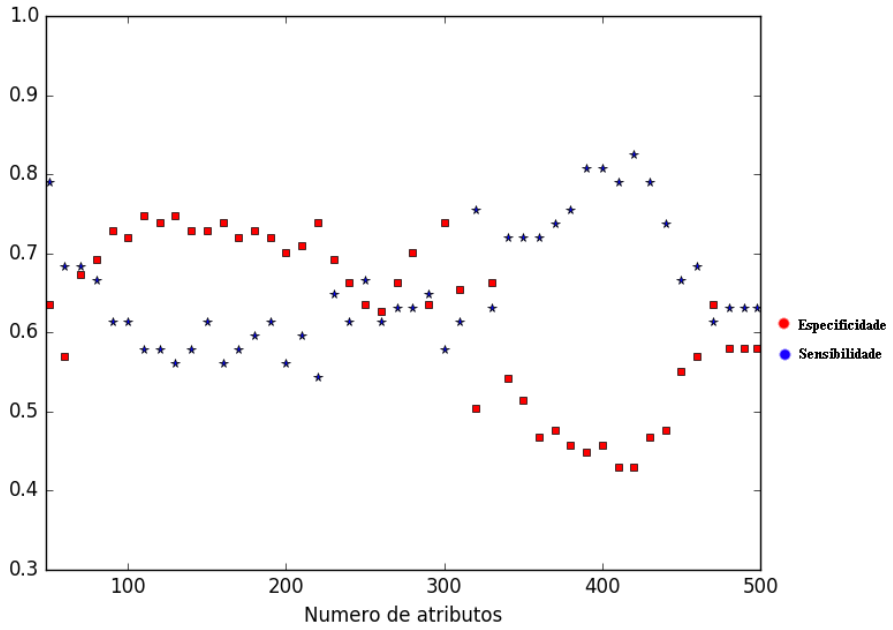

Figura 5.6: Medidas de sensibilidade e especificidade após aplicar Naive Bayes Gaussiano em matrizes com diferentes números de atributos.

K-Fold. A Tabela 5.8 apresenta os resultados dos cinco primeiros modelos treinados, na qual podemos reparar que a média de acurácia atinge 71%, porém a média de precisão continua abaixo de 70% e há uma grande diferença positiva na média de especificidade quando comparamos com o Bayes Gaussiano. A porcentagem de acertos negativos aumentou de menos 60% para 93%, enquanto a média de sensibilidade diminuiu e alcançou apenas 40%. O melhor modelo obteve 80% de acurácia, 77% de precisão, 54% de sensibilidade e 95% de especificidade.

Executando o K-Fold novamente e treinando mais modelos, as médias finalmente atingem 70%, sendo 72% de acurácia e 70% de precisão. Os resultados estão apresentados na Tabela 5.9. Neste caso, a média de especificidade foi de 91%, o dobro da média de sensibilidade, que foi de apenas 45%. O método não só garantiu as melhores médias entre as dois classificadores de Bayes, mas também garantiu o modelo com maiores medidas de acurácia e precisão dentre os quatro testes. Este atingiu acurácia de 86%, precisão de 81%, 65% de sensibilidade e 100% de especificidade.

Os resultados demonstram que o algoritmo parece ser influenciado pelo leve desbalanceamento no número de *tweets* positivos e negativos. Isso acontece porque,

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.80	0.77	0.54	0.95
2	0.71	0.69	0.35	0.93
3	0.70	0.65	0.41	0.94
4	0.69	0.65	0.34	0.96
5	0.66	0.65	0.38	0.86
Média	**0.71**	**0.68**	**0.40**	**0.93**
Desvio Padrão	**0.04**	**0.04**	**0.07**	**0.03**

Tabela 5.8: Resultados para Naive Bayes Multinomial para K=5

quando não há uma evidência clara da classe de um documento, a classe escolhida pelo algoritmo é a que possui frequência relativa mais alta, que neste caso, é a classe de *tweets* negativos.

Quando aplicado SVD, os resultados encontrados para diferentes números de atributos não foram satisfatórios, já que as medidas raramente atingiram 70%, tanto para acurácia quanto para a precisão, como mostra a Figura 5.7. Analisando os resultados, é possível perceber que quando há um aumento na acurácia, há uma diminuição de precisão. Isso se dá pelo alto número de *tweets* classificados corretamente como negativos e pelos poucos *tweets* classificados corretamente como positivos. Para a maioria das matrizes testadas, a medida de especificidade está acima de 90%, como mostra a Figura 5.8. Vale ressaltar que a menor porcentagem de acertos negativos encontrada nos testes com o SVD é de 85%. Totalmente o oposto das medidas de sensibilidade, nas quais é raro encontrar resultados acima de 50%, o que acontece apenas duas vezes para as matrizes de 130 e 140 termos. Os resultados também mostram que o algoritmo é influenciado pelo número de *tweets* negativos ser maior do que o número de *tweets* positivos.

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.72	0.73	0.44	0.87
2	0.62	0.60	0.41	0.80
3	0.72	0.62	0.51	0.95
4	0.74	0.77	0.40	0.88
5	0.68	0.64	0.31	0.96
6	0.66	0.60	0.29	1.00
7	0.76	0.75	0.41	0.93
8	0.78	0.74	0.59	0.92
9	0.86	0.81	0.65	1.00
10	0.70	0.72	0.47	0.83
Média	**0.72**	**0.70**	**0.45**	**0.91**
Desvio Padrão	**0.06**	**0.07**	**0.10**	**0.06**

Tabela 5.9: Resultados para Naive Bayes Multinomial para K=10

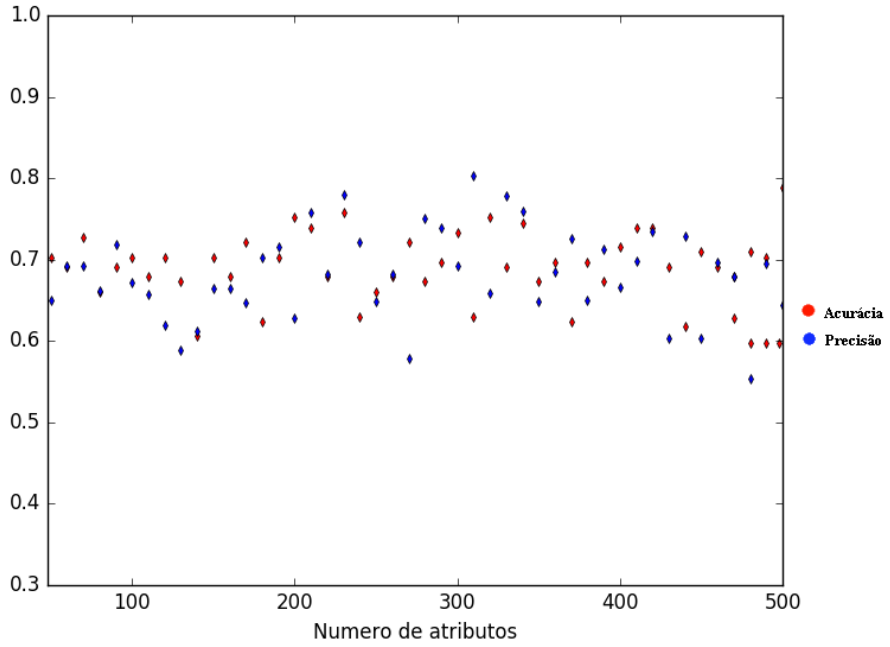

Figura 5.7: Medidas de acurácia e precisão após aplicar Naive Bayes Multinomial em matrizes com diferentes números de atributos.

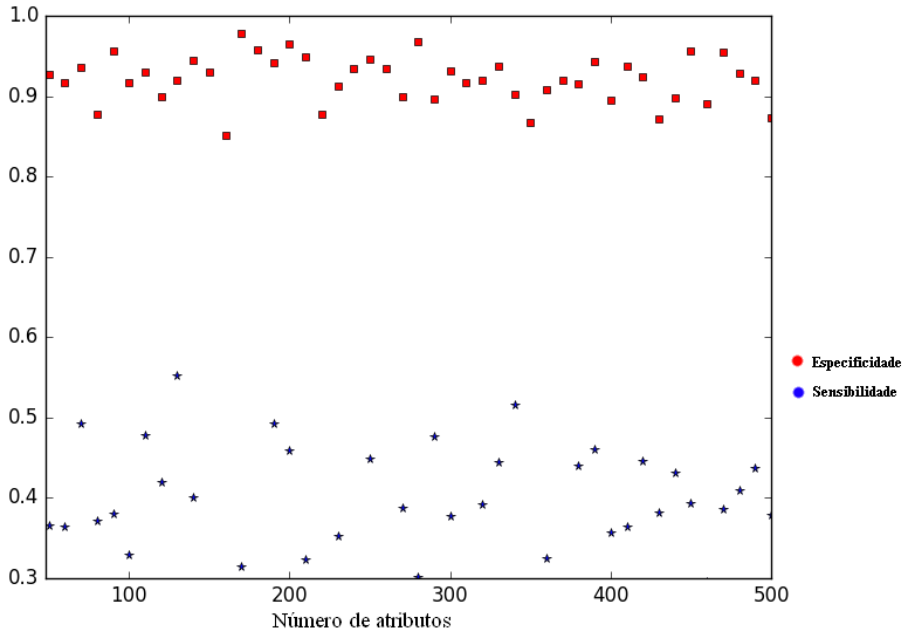

Figura 5.8: Medidas de sensibilidade e especificidade após aplicar Naive Bayes Multinomial em matrizes com diferentes números de atributos.

5.4 CLASSIFICADOR UTILIZANDO REDES NEURAIS ARTIFICIAIS

O primeiro passo é encontrar o melhor tipo de rede para o problema. Para este trabalho, foi escolhida a rede de classificação *Multi-layer Perceptron Classifier*, ou MLPClassifier [8], também da biblioteca Scikit-Learn, que implementa um algoritmo de multicamadas de neurônios que são treinados usando *backpropagation*.

A rede pode ser treinada usando diferentes algoritmos de otimização na fase de treinamento. O algoritmo mais comum para treinar uma rede neural é o Gradiente Descendente [24], que tem sido aplicado com sucesso em problemas esparsos e de larga escala, frequentemente encontrados em classificação de textos e problemas de PLN. Mesmo sendo muito indicado para o problema, o algoritmo escolhido para treinar a rede foi o *Limited memory Broyden-Fletcher-Goldfarb-Shanno* (L-BFGS) [14]. A principal razão para escolher o L-BFGS no lugar do Gradiente Descendente está em algumas desvantagens encontradas no algoritmo implementado na biblioteca Scikit-Learn, que é o Gradiente Descendente Estocástico [9] (*Stochastic Gradient*

[8] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[9] https://scikit-learn.org/stable/modules/sgd.html

Descent - SGD). Os SGDs, como são chamados, requerem muito ajuste manual dos parâmetros de otimização (taxa de aprendizagem e critérios de convergência), o que pode tornar muito difícil encontrar uma boa taxa de aprendizado ou um bom critério de convergência [14]. A principal estratégia seria treinar o algoritmo com vários parâmetros de otimização e escolher o modelo que oferece melhor desempenho para o conjunto de validação. Além de tomar muito tempo e ser computacionalmente caro, é preciso pesquisar sobre um grande espaço de possíveis parâmetros de otimização, o que torna SGDs difíceis de treinar. Já para a função de ativação, foi escolhida a *Rectified Linear Units* (ReLu), que popularizou recentemente e promete convergir mais rápido do que a tangente hiperbólica e a função sigmóide, além de evitar o problema de desaparecimento do gradiente [14]. Na primeira camada, são usados 2066 neurônios, que são os componentes do vetor de entrada e apenas um único neurônio é utilizado na saída.

Utilizando a configuração inicial, com poucas camadas ocultas e poucos neurônios, esperava-se resultados bem ruins, já que a quantidade de neurônios na primeira camada é muito maior. Porém, a rede alcançou 61% de acurácia, 76% de precisão, 69% de sensibilidade e 56% de especificidade em um primeiro teste. Quando utilizado o algoritmo K-Fold, as médias foram bem próximas das médias encontradas quando utilizado o SVM para treinar os cinco primeiros modelos. A rede alcançou uma média de 75% de acurácia, 76% de precisão, 60% de sensibilidade e 85% de especificidade, como mostra a Tabela 5.10. O melhor modelo treinado alcançou 78% de acurácia e 80% de precisão, mas o que surpreende são as medidas de sensibilidade e especificidade. Estas ficaram bem próximas, atingindo 78% e 77%, respectivamente.

Na segunda execução do K-Fold, as médias são praticamente mantidas: 74% de acurácia, 76% de precisão, 62% de sensibilidade e 83% de especificidade, como mostra a Tabela 5.11. O melhor modelo treinado usando a rede acertou quase todos os exemplos positivos e teve ótimo desempenho ao classificar *tweets* como negativos. Este alcançou medidas de acurácia de 88%, 95% de precisão, 95% de sensibilidade e 81% de especificidade.

Como mostram as tabelas 5.10 e 5.11, o melhor modelo treinado se afasta muito

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.73	0.75	0.64	0.79
2	0.77	0.78	0.52	0.89
3	0.78	0.80	0.78	0.77
4	0.71	0.66	0.45	0.94
5	0.76	0.78	0.63	0.82
Média	**0.75**	**0.76**	**0.60**	**0.85**
Desvio Padrão	**0.02**	**0.04**	**0.11**	**0.06**

Tabela 5.10: Resultados da rede neural com poucas camadas ocultas para K=5

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.64	0.66	0.47	0.75
2	0.82	0.90	0.78	0.83
3	0.60	0.59	0.45	0.73
4	0.88	0.95	0.95	0.81
5	0.70	0.75	0.65	0.73
6	0.78	0.77	0.68	0.85
7	0.72	0.69	0.42	0.93
8	0.72	0.75	0.57	0.80
9	0.77	0.75	0.52	0.93
10	0.85	0.80	0.73	0.96
Média	**0.74**	**0.76**	**0.62**	**0.83**
Desvio Padrão	**0.08**	**0.10**	**0.15**	**0.08**

Tabela 5.11: Resultados da rede neural com poucas camadas ocultas para K=10

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.79	0.77	0.71	0.85
2	0.77	0.85	0.74	0.78
3	0.71	0.82	0.80	0.63
4	0.70	0.70	0.66	0.73
5	0.74	0.81	0.70	0.76
Média	**0.74**	**0.79**	**0.72**	**0.75**
Desvio Padrão	**0.06**	**0.07**	**0.10**	**0.05**

Tabela 5.12: Resultados da rede neural com única camada oculta de 100 neurônios para K=5.

da média. Para tentar melhorar as médias em cada rodada e consequentemente obter modelos melhores, foi feita uma busca pela melhor quantidade de camadas e de neurônios nessas camadas. Também foram ajustados alguns parâmetros a fim de melhorar os resultados. Duas das principais mudanças nos parâmetros para treinar uma nova rede foram o coeficiente de aprendizado, que passou a ser adaptativo, e o uso de *early stopping*. Nos novos testes, foi utilizada uma única camada com 100 neurônios, que garantiu aumento na precisão. As tabelas 5.12 e 5.13 mostram os resultados das duas execuções do K-Fold e, analisando as duas em conjunto, podemos perceber que a média de acurácia se mantém, mas há um aumento de 3% na média de precisão. O melhor modelo encontrado atinge 82% de acurácia, 90% de precisão, 91% de sensibilidade e 74% de especificidade e pode-se dizer que o novo modelo é melhor que o anterior por seus resultados não se afastarem tanto da média.

É difícil interpretar os resultados encontrados ao treinar a rede com matrizes de diferentes dimensões de atributos, já que redes neurais são modelos muito complexos. O que é indiscutível é como os resultados foram muito melhores do que os resultados com os métodos anteriores, como mostra a Figura 5.9. Surpreendentemente, para diferentes dimensões de atributos, temos a maioria dos resultados com mais de 75% de precisão e a maior precisão alcançada é de 85%. Além disso, os resultados usando a rede neural demonstram um melhor equilíbrio entre sensibilidade e especificidade,

	Acurácia	Precisão	Sensibilidade	Especificidade
1	0.66	0.82	0.64	0.66
2	0.82	0.90	0.91	0.74
3	0.76	0.81	0.76	0.75
4	0.62	0.65	0.54	0.67
5	0.80	0.86	0.77	0.81
6	0.80	0.85	0.80	0.80
7	0.80	0.79	0.74	0.85
8	0.78	0.83	0.73	0.80
9	0.68	0.69	0.60	0.74
10	0.70	0.69	0.68	0.72
Média	**0.74**	**0.79**	**0.72**	**0.75**
Desvio Padrão	**0.06**	**0.07**	**0.10**	**0.05**

Tabela 5.13: Resultados da rede neural com única camada oculta de 100 neurônios para K=10.

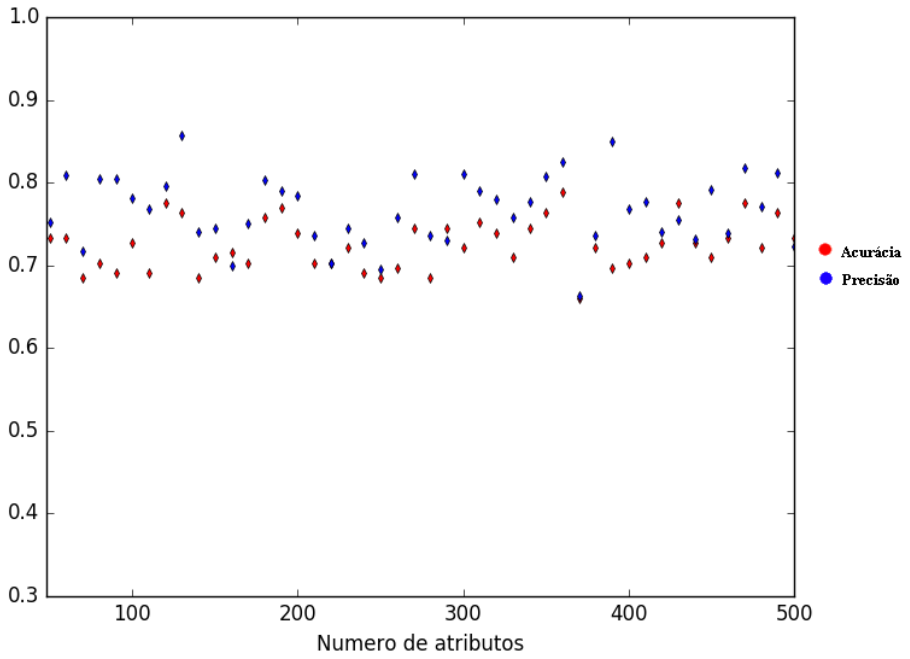

Figura 5.9: Medidas de acurácia e precisão após aplicar a Rede Neural em matrizes com diferentes números de atributos.

ao contrário dos outros métodos que possuem medidas de especificidade sempre muito maiores do que as de sensibilidade, como mostra a Figura 5.10.

O melhor modelo encontrado durante os testes foi treinado com uma matriz de 130 atributos, o que garantiu 76% de acurácia, 85% de precisão, 81% de sensibilidade e 72% de especificidade. Não é possível afirmar se a rede trabalha melhor a partir de um certo número de atributos, já que temos bons resultados para matrizes com muitos e poucos termos. Mesmo com bons resultados, nenhum dos testes alcançou resultados melhores do que a melhor rede treinada usando a matriz original.

5.5 ESCOLHA DOS CLASSIFICADORES PARA FILTRAR *TWEETS* DE ASSALTO

Diante dos resultados obtidos nas seções anteriores, podemos perceber como cada método se comporta diante dos dados apresentados e que todos apresentam melhores resultados utilizando os dados normalizados. Após as primeiras análises individuais, as conclusões sobre cada um dos métodos são apresentadas a seguir. Os melhores

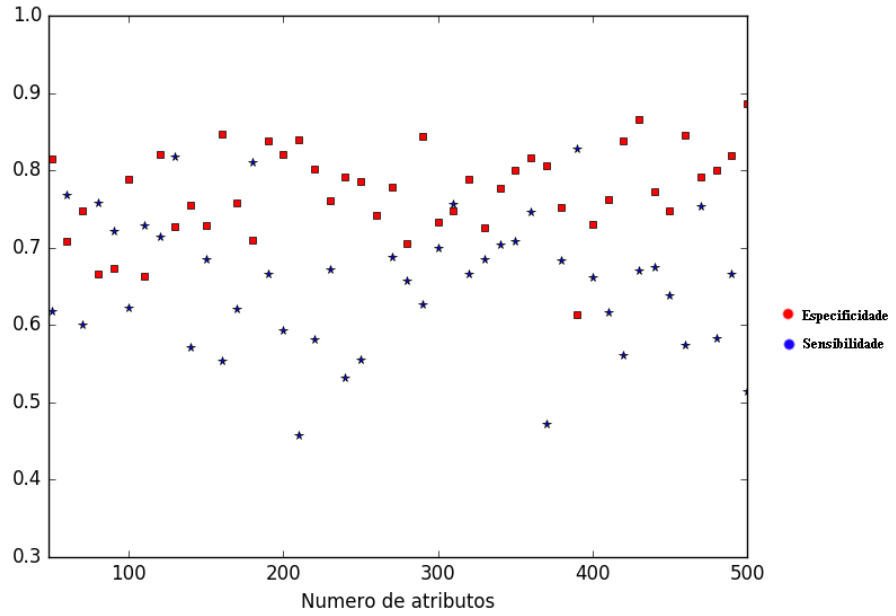

Figura 5.10: Medidas de sensibilidade e especificidade após aplicar a Rede Neural em matrizes com diferentes números de atributos.

modelos foram usados em um comitê para a classificação dos demais *tweets* da base de dados. Um comitê consiste em combinar modelos de aprendizado diferentes para melhorar a classificação de uma instância. É um sistema de votação, onde cada modelo vota pela classificação que acredita ser a correta.

Antes de começar os experimentos propriamente ditos, foi necessário utilizar um algoritmo de particionamento para entender melhor a natureza e a estrutura dos dados. O fato do algoritmo agrupar a maioria dos dados em torno de um mesmo *cluster* chamou atenção para a base de dados e como estes são representados. Realizando uma análise mais profunda, foi identificado que o problema está na frequência dos termos: a maioria dos termos só aparece uma única vez no *corpus* e, os que aparecem mais vezes, só aparecem uma vez por documento, o que ocasiona em documentos com representações muito parecidas. Em uma tentativa de contornar o problema, foram removidos os termos pouco frequentes e pouco relevantes dos textos. A tática não funcionou, já que 70% dos termos só aparecem uma única vez em toda a base. Uma outra forma de tentar melhorar o agrupamento foi utilizar o SVD, que melhorou um pouco os resultados para a base normalizada, mas não

mostrou uma melhora expressiva. Todos os testes mostraram uma base de dados muito homogênea e o agrupamento ruim não invalida a possibilidade de sucesso dos métodos de aprendizado supervisionado.

Após os resultados ruins usando o K-Means, o primeiro método de aprendizado supervisionado testado foi o SVM. Por ser um dos mais citados nos artigos de referência, era esperado o bom desempenho do mesmo, que nos testes iniciais não supriu as expectativas devido a escolha inicial dos parâmetros. Para encontrar os melhores parâmetros, foi feita uma busca exaustiva e o método já apresentou resultados muito melhores após mudar o kernel para o tipo linear. As boas médias nas duas execuções do K-Fold mostraram que o método tem bom desempenho diante do conjunto de dados apresentado. Por já ser um método resistente a overfitting, as tentativas de melhorar esses resultados utilizando SVD não tiveram sucesso. O melhor modelo treinado foi selecionado a partir dos resultados da segunda execução do K-Fold e obteve 84% de acurácia, 88% de precisão, 75% de sensibilidade e 88% de especificidade.

Também constantemente citado em artigos de classificação de textos, o NBC pode ser utilizado em suas duas implementações: Gaussiana e Multinomial. Como os dados neste trabalho são representados pela ponderação TF-IDF, foi possível utilizar a implementação Gaussiana, que é indicada para dados de valores contínuos. Com médias muito inferiores às médias do SVM, durante as execuções do K-Fold, o melhor modelo teve apenas 56% de acurácia, 69% de precisão, 63% de sensibilidade e 51% de especificidade. É o único método que as medidas de sensibilidade são maiores que as de especificidade e também é o único que teve uma melhora nos resultados após utilizar SVD. Por ser um forte candidato a ser amaldiçoado pela maldição de dimensionalidade, quanto maior o número de atributos, mais exemplos são necessários para seu aprendizado. O modelo com melhores medidas foi encontrado utilizando uma matriz de apenas 50 atributos e obteve 69% de acurácia, 85% de precisão, 78% de sensibilidade e 63% de especificidade.

Dentre os diferentes tipos de NBCs, o Naive Bayes Multinomial costuma ser o mais recomendado para classificação de textos e costuma ter bom desempenho diante

de vetores TF-IDF. Os resultados iniciais foram bem melhores do que os testes com a implementação Gaussiana e, durante as execuções do K-Fold, o algoritmo mostrou ser tendencioso ao ter porcentagens de acertos negativos muito maiores do que a de acertos positivos, o que também aconteceu para todas as matrizes durante os testes com o SVD. Uma justificativa pode ser o número de *tweets* negativos ser maior do que o número de *tweets* positivos, já que o modelo escolhe a classe de frequência relativa mais alta caso não tenha uma evidência clara para a classe do documento. Um dos modelos conseguiu acertar todos os exemplos negativos, garantindo 100% de especificidade, o que não foi um problema já que as medidas de acurácia e precisão foram ótimas. Assim, o modelo escolhido para integrar o comitê foi um multinomial com 86% de acurácia, 81% de precisão, 65% de sensibilidade e 100% de especificidade.

Redes neurais são modelos muito complexos e muitas vezes os resultados são difíceis de serem interpretados. Assim como o SVM, também foi preciso ajustar os parâmetros de acordo com o problema a ser resolvido. Escolhidos os algoritmos de otimização da fase de treinamento e a função de ativação, foi feita uma busca pelo número de camadas e de neurônios por camada que melhorassem o desempenho do método. Em um primeiro teste, foram usadas poucas camadas, poucos neurônios e os parâmetros de configuração inicial, o que garantiu resultados positivamente inesperados. Após o ajuste dos parâmetros, a mudança no número de camadas ocultas e no número de neurônios, foram obtidas as maiores médias dentre todos os outros métodos nas execuções do K-Fold, o que faz da rede neural o método com melhor desempenho entre todos os métodos testados. Este também treinou o modelo com as melhores medidas dentre todos os outros, que foi escolhido para fazer parte do comitê, com 82% de acurácia, 90% de precisão, 91% de sensibilidade e 74% de especificidade. Os resultados também foram ótimos nos testes com o SVD, mantendo as medidas de acurácia e precisão sempre iguais ou acima de 70%.

5.6 ANÁLISE QUALITATIVA DOS MODELOS TREINADOS

Avaliar o desempenho dos modelos treinados através de métricas de validação é um dos passos mais importantes em AM. Além de análises quantitativas, também é muito importante realizar uma análise qualitativa, principalmente quando estamos lidando com a classificação de textos tão complicados como *tweets*.

Para um teste realista, foram selecionados textos diretamente do Twitter, que foram pré-processados e depois representados numericamente como feito para os textos da base anotada. Para cada termo do novo *tweet*, é calculada sua frequência no texto e depois verificado se o termo está presente na matriz de termos e frequências. Em caso afirmativo, será usado o IDF correspondente ao termo para o cálculo do seu peso. Em caso negativo, é atribuído peso zero ao termo. Depois que todos os novos *tweets* foram representados, o vetor de cada documento foi utilizado para classificação.

Para avaliar o comportamento de cada modelo selecionado na seção anterior, foram coletados *tweets* que representassem bem *tweets* negativos e positivos, independente de citar ou não um bairro. A Tabela 5.14 apresenta os textos utilizados durante os testes, a classificação atribuída por cada modelo e a classificação por humanos, chamada de coluna de "classificação correta". Um exemplo de *tweet* que representa bem um exemplo negativo é o primeiro texto da tabela. Nele podemos encontrar elementos que não seriam utilizados para relatar assaltos, como "jogar", "primeiro tempo" e "Grêmio". Na base de *tweets* anotados, temos um exemplo parecido e, talvez por isso, temos unanimidade na classificação e todos os modelos acertaram ao classificá-lo como negativo.

Já os textos 4, 5, 7 e 8, representam muito bem exemplos positivos, já que podemos observar elementos mais característicos na descrição de assalto ou uma tentativa de assalto, como na construção "quase fui assaltado", que é apenas relacionada a *tweets* positivos. Os termos relacionados ao envio de mensagens estranhas e objetos roubados são característicos de *tweets* de usuários que foram assaltados e por isso foram importantes durante o treinamento. Outro elemento presente na base

		Classificação			
	Texto	SVM	Naive Bayes	Rede Neural	Classificação Correta
1	“Grêmio foi assaltado no primeiro tempo... Madson é muito fraco, não pode jogar no grêmio... Cícero também deixando a desejar... dá pra melhorar com os caras do banco.”	Negativo	Negativo	Negativo	Negativo
2	“Ou você é assaltado ou assalta.”	Negativo	Negativo	Negativo	Negativo
3	“to me sentindo meio estranho sei la tipo meio assaltado”	Negativo	Negativo	Negativo	Negativo
4	“Fui assaltado hoje a tarde, se alguém enviar alguma coisa estranha saiba que não fui eu.”	Positivo	Positivo	Positivo	Positivo
5	“Tava vindo da casa do Isaac os cara fecharam o pai dele e já vieram de fuzil atrás, pai dele meteu o pé, conseguiu fugir, ia ser geral assaltado.”	Positivo	Positivo	Positivo	Positivo
6	“Bom dia pra você que vai pra Copacabana ser assaltado."	Negativo	Negativo	Positivo	Negativo
7	“Mal cheguei em Copacabana e quase fui assaltado em frente à tradicional Charutaria Lolló. Mas o Pezão acha que está tudo sob controle...”	Positivo	Positivo	Positivo	Positivo
8	“Boa noite. Meu pai foi assaltado ontem, por volta de 16:00h. Se alguém ver esse carro por favor, me avisem. Se puder compartilhar iria ajudar muito. Obrigado! ”	Positivo	Positivo	Positivo	Positivo

Tabela 5.14: Desempenho dos três classificadores para diferentes *tweets*

anotada é o verbo fugir, relacionado à *tweets* positivos de usuários que quase foram assaltados. Mais uma vez temos unanimidade na classificação e todos os modelos os classificaram corretamente.

Dois *tweets* que podem ser considerados mais difíceis de serem classificados corretamente são os textos 2 e 3. No caso do texto 2, removendo as *stopwords*, restam apenas os termos “assaltado” e “assalta”. Como o verbo “assaltado” é constantemente encontrado no conjunto de treinamento sendo associado a ambas classificações, a classificação atribuída por cada modelo era uma dúvida. Entretanto, os modelos surpreendem e todos classificam mais um texto corretamente como negativo. No texto 3, temos uma das palavras-chave combinada com termos que normalmente não estão relacionados a assalto, mas que também não aparecem relacionados a *tweets* negativos. Talvez por serem termos diferentes dos presentes na base anotada, os três modelos o classificaram como negativo.

De todos os textos usados, apenas um não foi classificado de forma unânime. No caso do texto 6, apenas a rede neural o classificou de forma incorreta, mas para os outros dois classificadores, este seria um verdadeiro negativo. O principal termo que evidencia um *tweet* negativo está na conjugação do verbo “ir” que, neste caso, indica que algo ainda não aconteceu, mas irá acontecer. Mesmo com alguns exemplos na base anotada, estes não foram suficientes e o exemplo mostra que seria bom treinar a rede com mais exemplos do tipo para garantir melhores resultados.

6 RESULTADOS DO MAPEAMENTO

De abril de 2018 à novembro de 2018, foram coletados 4.998.400 *tweets* apenas através das requisições. De todos os *tweets* vindos da API, 2.345.796 foram descartados ao passar pelo primeiro filtro, que identifica textos que com certeza não denunciam assaltos, como citado na Seção 4.2. Contando apenas com os restantes, 30.126 relataram tentativas de assaltos ou furtos e 67.815 citavam algum bairro da cidade do Rio de Janeiro.

O baixo número de *tweets* que citam bairros já tinha sido notado desde o início do projeto e uma das soluções para contornar o problema foi buscar conversas entre os usuários, como também citado na Seção 4.2. Como a busca por menções informando o local onde o assalto ocorreu não deu certo, restou apenas apostar em um grande volume de dados para que, mesmo descartando a maioria, ainda tivéssemos uma quantidade de informações significativa para fazer um mapeamento dos bairros mais perigosos. Para a classificação dos *tweets*, os modelos foram organizados em um comitê e um texto é considerado positivo se pelo menos dois classificadores concordarem em classificá-lo como positivo. O mesmo ocorre para o caso de classificação negativa. Diante dos poucos textos mencionando bairros, foram utilizados todos os *tweets* do banco de dados para obter o número de denúncias por mês. Para a contagem, foram selecionados apenas os *tweets* classificados como positivos pelo comitê. Os resultados são apresentados a seguir.

Na Figura 6.1, estão alguns dos bairros do Rio de Janeiro mencionados pelos usuários e o número de denúncias de assalto em cada um. Já a Figura 6.2, mostra a mesma informação em um mapa da cidade [12]. De acordo com as informações da Figura 6.1, o bairro com maior número de denúncias de assalto é o bairro Vasco da Gama e o Flamengo está em terceiro lugar. Mesmo identificando *tweets* que falam sobre futebol como negativos na base usada para treinamento e validação, erros de classificação ainda são possíveis e, ao analisar esses *tweets* em específico, realmente parecem ser relatos de assalto, um exemplo é: “O Vasco foi assaltado naquela espelunca ontem!” Para nós humanos, é fácil classificá-lo como negativo, já que Vasco normalmente não é nome de pessoa, mas é perfeitamente compreensível

que os modelos errem essa classificação, já que não estão presentes nenhum dos termos relacionados a jogos de futebol. No caso do Flamengo, os *tweets* se referem a um treino do time em que grande parte dos torcedores foi assaltada, o que gerou postagens como "fui assaltada saindo do treino do Flamengo" e "quase fui assaltado saindo do treino do Flamengo". O que não configura um erro dos classificadores, realmente uma pessoa foi assaltada e outra quase assaltada, o problema em questão é o local que não foi identificado corretamente.

O segundo colocado no *ranking* dos bairros é o Centro da cidade. É comum centros de grandes cidades terem a fama de serem perigosos e isto não é diferente para o Centro do Rio de Janeiro. Por ser uma região que sempre teve fama de perigosa, faz sentido estar no topo da lista. Também faz sentido termos bairros como Botafogo, Ipanema, Lagoa e Copacabana nesta lista, já que as notícias recentes (citadas no Capítulo 1) relatam que o número de ocorrências tem aumentado em bairros da Zona Sul. Mesmo com um resultado coerente com as notícias recentes, era esperado que bairros como Copacabana e Ipanema estivessem no topo da lista.

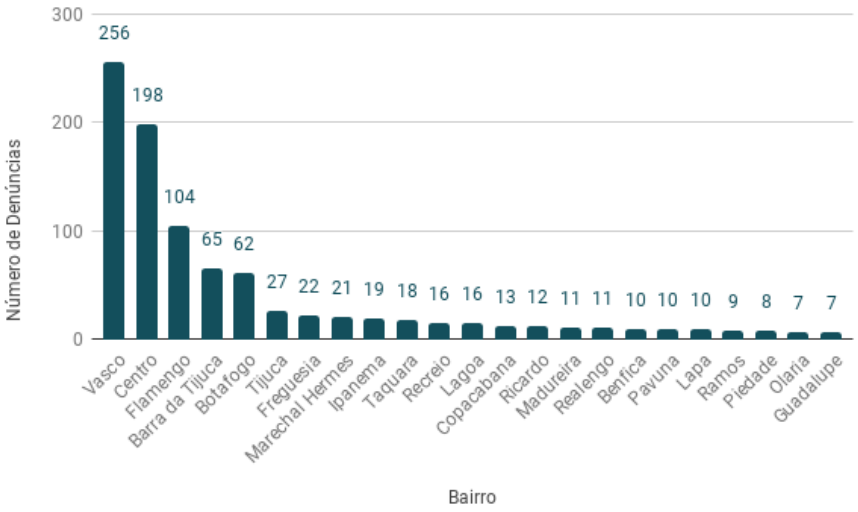

Figura 6.1: Número de ocorrências por bairro segundo dados coletados do Twitter.

Apenas 1.5% dos *tweets* que mencionam bairros foram classificados como positivos e, para não haver perda de informação sobre denúncias de crimes, os *tweets* que não mencionam bairros também foram utilizados para contabilizar o número de denúncias por mês, apresentados na Figura 6.3. Para comparar com denúncias re-

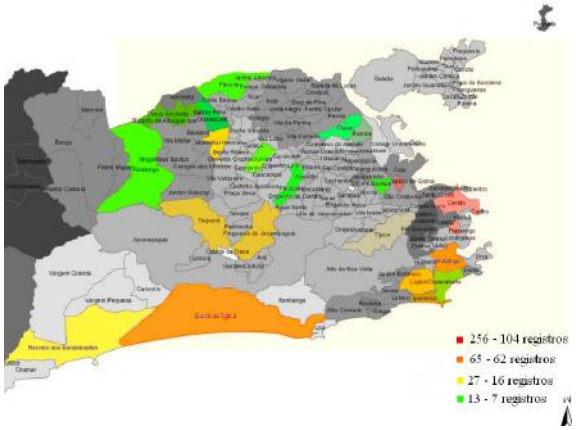

Figura 6.2: Assaltos por região no mapa da cidade do Rio de Janeiro.

ais, foram coletados dados do ISP, disponíveis no próprio site [1], referentes aos meses apresentados na Figura 6.3. Os dados são fornecidos por cada uma das delegacias da cidade e foi considerada apenas a quantidade de roubos e furtos disponibilizada em cada mês. Como ainda não foram divulgados os dados de novembro, este mês foi omitido do gráfico da Figura 6.4, que compara os dados das duas fontes.

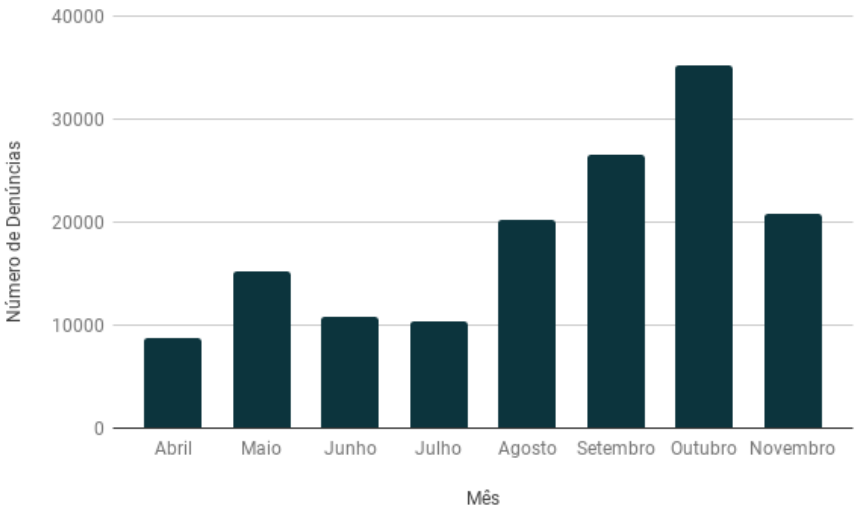

Figura 6.3: Número de denúncias de assalto por mês segundo dados coletados do Twitter.

Comparando os resultados na Figura 6.4, o único mês que as denúncias feitas no Twitter se aproximam dos registros de ocorrência do ISP é o mês de maio. Nos

[1] http://www.isp.rj.gov.br/

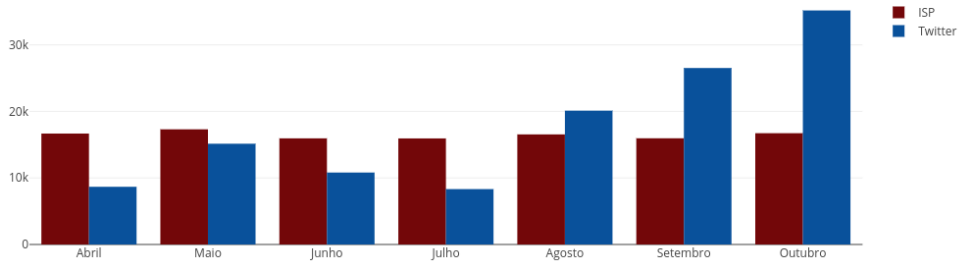

Figura 6.4: Dados coletados do Twitter comparados com os dados do ISP.

primeiros quatro meses, o número de denúncias no Twitter é menor do que os registros formais e a situação se inverte nos três últimos meses. As denúncias no Twitter crescem em setembro e outubro, período próximo às eleições, que movimentou a rede social e muitos usuários, reais ou não, usaram a plataforma para expor roubos e escândalos de corrupção dos candidatos e seus partidos.

A diferença nos números de registros formais e de denúncias via Twitter já era de se esperar, mas não era esperado que o número de denúncias na rede social fosse menor do que os dados disponibilizados pelo ISP, como aconteceu nos primeiros quatro meses. Contudo, contar denúncias através de um comitê de classificadores ainda não é tão acurado quanto contar o número de registros de uma delegacia, já que classificadores podem errar, como já foi demonstrado, e talvez nem todas denúncias foram classificadas como positivas. Por isso, ainda não é possível dizer que a suspeita de que existem mais denúncias em redes sociais do que registros de ocorrência está errada.

7 CONCLUSÃO

7.1 CONCLUSÃO

O objetivo do projeto foi encontrar os bairros mais perigosos do Rio de Janeiro, segundo informações dos usuários do Twitter, e comparar o número de denúncias feitas na rede social com o número de registros de ocorrência nas delegacias da cidade. No fim, o trabalho consistiu em um estudo da aplicação de métodos de AM em *tweets* para identificação de relatos de assalto. Além disso, verificar se é possível identificar os bairros mais perigosos da cidade, o que é bem difícil sem geolocalização e poucas pessoas mencionarem o local.

Com os estudos realizados, foi possível desenvolver um sistema que coleta e classifica *tweets* para identificar os bairros em que ocorreram crimes e o número de denúncias feitas por mês na rede social. Além disso, o trabalho relatou o processo para construção de um banco de dados e gerou uma base que pode ser utilizada em outros trabalhos. A pouca quantidade de *tweets* na base anotada foi um dos problemas encontrados durante o desenvolvimento do trabalho. Por falta de voluntários, não foi possível montar uma base maior, rica em exemplos, para encontrar resultados ainda melhores do que os apresentados. Problemas de AM normalmente funcionam melhor com mais exemplos e geralmente são usadas bases bem maiores do que 500 exemplos. Mesmo que alguns dos métodos tenham alcançado bons resultados, o número de exemplos ainda é pequeno para o problema que tentamos resolver.

Sobre a extração de dados, pode-se dizer que o pouco número de requisições disponível e a falta de alguns recursos na versão gratuita da API atrapalharam um pouco a coleta dos dados. Muitas vezes a API reportou erros de acesso durante as requisições, impedindo a coleta dos dados essenciais para o trabalho. A função de buscar conversas entre usuários seria bem útil e poderia ter ajudado a coletar muito mais informações sobre os bairros, mas há pouco tempo esta foi retirada da versão gratuita da API. Não podemos esquecer da falta de *tweets* com geolocalização e o funcionamento ruim do serviço. Mesmo utilizando as coordenadas geográficas e o raio de cobertura nas requisições, ainda eram coletados *tweets* de outras cidades e

até outros estados brasileiros.

A limpeza dos dados foi um dos grandes desafios e demandou mais tempo do que o esperado. Apesar de existirem bibliotecas como o NLTK, que já possuem muitos métodos que ajudam na limpeza, retirar anomalias como links e risadas demandou muito tempo e estudo sobre expressões regulares. O dicionário de abreviações também demandou tempo, já que foi preciso ser feito um estudo de todas as abreviações que apareciam nos textos da base de dados. Além das constantes atualizações desse dicionário, algumas abreviações não foram mapeadas pela dificuldade de entender o que o usuário quis dizer. Com certeza o conteúdo do dicionário ainda não é suficiente devido à grande quantidade de palavras que são abreviadas na Internet. Nesta etapa do projeto, a falta de boas ferramentas para processamento de textos em língua portuguesa foi um grande obstáculo.

O maior desafio do trabalho foi treinar os classificadores para obter o melhor desempenho possível. Ter poucos exemplos pode ter sido prejudicial nesta etapa do projeto, mas os resultados foram bem satisfatórios. Ajustar os parâmetros de alguns métodos demandou muito tempo e pesquisa, foi preciso entender melhor os métodos e os parâmetros que deveriam ser ajustados. Esta dificuldade foi maior para as redes neurais, durante a escolha das funções de otimização e de ativação. Analisar e compreender os resultados também foi uma tarefa difícil, já que alguns métodos não mostravam ter um padrão em seus resultados, como o Naive Bayes Multinomial e as redes neurais.

Com os classificadores prontos e testados, foram analisados todos os *tweets* coletados durante os meses de desenvolvimento. Os resultados mostraram que as pessoas procuram mais as delegacias do que as redes sociais. O resultado é subjetivo, já que nenhum dos classificadores está livre de errar classificações, e os valores reais podem ser maiores ou menores do que os encontrados durante a pesquisa. Talvez os resultados pudessem ter sido mais realistas se o número de *tweets* coletados fosse maior, o que talvez fosse resolvido se não tivéssemos tantas limitações da API.

Treinar um bom classificador de *tweets* é uma tarefa difícil, mas deixando de lado as dificuldades, os resultados foram surpreendentemente satisfatórios e o sistema

pode ser utilizado para processar informações em tempo real futuramente.

7.2 TRABALHOS FUTUROS

Evoluir o sistema para suportar dados em tempo real e disponibilizar esses dados para a comunidade através de um portal para consultas do mapeamento por bairro e do número de assaltos por mês.

Alguns ajustes ainda devem ser feitos para melhorar o desempenho do sistema. Novos exemplos podem ser adicionados à base de dados para tentar melhorar o desempenho dos classificadores. No caso da rede neural, continuar a busca por novos parâmetros para melhorar o desempenho da mesma. Uma das formas de melhorar o desempenho dos métodos que não foi utilizada neste trabalho, mas que pode ser utilizada em trabalhos futuros, é a seleção de atributos. Através de uma pesquisa mais profunda pelos atributos mais significativos, talvez poderiam ser obtidos resultados melhores após a seleção desses atributos. Uma outra forma de melhorar as classificações seria utilizar *Stacking* no lugar do comitê.

Na parte de PLN, outro fator importante a ser melhorado é a identificação dos bairros que ocorreram os assaltos. Como o exemplo do Flamengo citado no Capítulo 6, nem sempre o bairro que é identificado no *tweet* é o local onde ocorreu o assalto. Por conta da informalidade dos *tweets* e a falta de boas práticas gramaticais, identificar se o bairro citado é o local onde ocorreu o assalto é difícil.

É esperado que esse material sirva de base para pesquisas futuras da área de aprendizado de máquina, seja pela base anotada de *tweets* em português, pela criação do banco de dados, pelo Capítulo 3 que explica os principais conceitos da área ou pelos resultados encontrados nos experimentos.

REFERÊNCIAS

[1] Aramaki, E., Maskawa, S., e Morita, M. Twitter catches the flu: Detecting influenza epidemics using twitter. In *EMNLP* (2011), ACL, pp. 1568–1576.

[2] Archana, S. H., e Winster, S. G. Drugs categorization based on sentence polarity analyzer for twitter data. *2016 Second International Conference on Science Technology Engineering and Management (ICONSTEM)* (2016), 28–33.

[3] Baeza-yates, R., Ribeiro-neto, B., Wives, L., e Moreira, V. *Recuperação De Informação: CONCEITOS E TECNOLOGIA DAS MÁQUINAS DE BUSCA*. BOOKMAN COMPANHIA ED, 2013.

[4] Bisio, F., Meda, C., Zunino, R., Surlinelli, R., Scillia, E., e Ottaviano, A. Real-time monitoring of twitter traffic by using semantic networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015* (New York, NY, USA, 2015), ASONAM '15, ACM, pp. 966–969.

[5] Cawley, G. C., e Talbot, N. L. On over-fitting in model selection and subsequent selection bias in performance evaluation. *J. Mach. Learn. Res. 11* (agosto de 2010), 2079–2107.

[6] Estadão. 16 dos 27 indicadores de violência caíram no rio no carnaval, 2018. `https://brasil.estadao.com.br/noticias/rio-de-janeiro,16-dos-27-indicadores-de-violencia-cairam-no-rio-no-carnaval,70002192304`, Acessado em 2018-03-23.

[7] Exame. Fuja do perigo com apps que mapeiam ocorrências de crimes, 2015. `https://exame.abril.com.br/tecnologia/fuja-do-perigo-com-apps-que-mapeiam-ocorrencias-de-crimes/`, Acessado em 2018-03-23.

[8] Fox, D., Hightower, J., Liao, L., Schulz, D., e Borriello, G. Bayesian filtering for location estimation. *IEEE Pervasive Computing 2*, 3 (2003), 24–33.

[9] G1 - Globo.com. #mapadocrime: número de roubos bate recorde no rio, e copacabana tem aumento de 128% nos últimos 2 anos, 2017. https://g1.globo.com/rio-de-janeiro/noticia/mapadocrime-numero-de-roubos-bate-recorde-no-rio-e-copacabana-tem-aumento-de-2ghtml, Acessado em 2018-03-23.

[10] G1 - Globo.com. Ocorrências no centro do rio aumentam 700% em dias de blocos; veja 10 dicas para se prevenir, 2018. https://g1.globo.com/rj/rio-de-janeiro/carnaval/2018/noticia/roubos-e-furtos-aumentam-700-em-dias-de-blocos-no-centro-do-rio-diz-policia-veghtml, Acessado em 2018-03-23.

[11] Hightower, J., e Borriello, G. Particle filters for location estimation in ubiquitous computing: A case study. In *In Proceedings of International Conference on Ubiquitous Computing (UbiComp* (2004), pp. 88–106.

[12] Instituto Pereira Passos. Município do rio de janeiro - divisões administrativas, 2008. http://www.processamentodigital.com.br/wp-content/uploads/2012/05/RJ_Limite_Adm.png, Acessado em 2018-12-18.

[13] Jornal O Globo. Índices de violência sobem no estado do rio, 2018. https://oglobo.globo.com/rio/indices-de-violencia-sobem-no-estado-do-rio-22407369, Acessado em 2018-03-23.

[14] Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., e Ng, A. Y. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (USA, 2011), ICML'11, Omnipress, pp. 265–272.

[15] Manning, C. D., Raghavan, P., e Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[16] Maxwell - PUC Rio. Classificador naive bayes. https://www.maxwell.vrac.puc-rio.br/9947/9947_5.PDF, Acessado em 2018-08-02.

[17] Mitchell, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[18] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[19] Narr, S., Hülfenhaus, M., e Albayrak, S. Language-independent twitter sentiment analysis, 2012.

[20] Omnicore Agency. Twitter by the number: Stats, demographics & fun facts, 2018. https://www.omnicoreagency.com/twitter-statistics/, Acessado em 2018-04-02.

[21] Pak, A., e Paroubek, P. Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)* (2010), European Languages Resources Association (ELRA).

[22] Pavalanathan, U., e Eisenstein, J. Confounds and consequences in geotagged twitter data. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (2015), Association for Computational Linguistics, pp. 2138–2148.

[23] PEREIRA, SILVIO L. Processamento de linguagem natural. https://www.ime.usp.br/~slago/IA-pln.pdf, Acessado em 2018-08-29.

[24] Ruder, S. An overview of gradient descent optimization algorithms. *CoRR abs/1609.04747* (2016).

[25] Russell, S., e Norvig, P. *Artificial Intelligence: A Modern Approach*, third ed. Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, NJ, 2010.

[26] Sakaki, T., Okazaki, M., e Matsuo, Y. Earthquake shakes twitter users: Real-time event detection by social sensors. In *In Proceedings of the Nineteenth International WWW Conference (WWW2010). ACM* (2010).

[27] Sas The Power To Know. Machine learning. o que é e qual sua importância?, 2018. `https://www.sas.com/pt_br/insights/analytics/machine-learning.html`, Acessado em 2018-04-02.

[28] Stonebraker, M. Sql databases v. nosql databases. *Commun. ACM 53*, 4 (abril de 2010), 10–11.

[29] STRAUCH, C. Nosql databases, 2009.

[30] Theodoridis, S., e Koutroumbas, K. *Pattern Recognition, Fourth Edition.* Academic Press, 2009.

[31] Verleysen, M., e François, D. The curse of dimensionality in data mining and time series prediction. In *IWANN* (2005), J. Cabestany, A. Prieto, e F. S. Hernández, Eds., vol. 3512 of *Lecture Notes in Computer Science*, Springer, pp. 758–770.