UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Augusto Alves Zanchini

CRIAÇÃO DE UM MODELO DE CLASSIFICAÇÃO DE TWEETS DEPRESSIVOS UTILIZANDO MÁQUINA DE VETORES DE SUPORTE

Patos de Minas

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Augusto Alves Zanchini

CRIAÇÃO DE UM MODELO DE CLASSIFICAÇÃO DE TWEETS DEPRESSIVOS UTILIZANDO MÁQUINA DE VETORES DE SUPORTE

Trabalho apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia, como requisito parcial para aprovação da disciplina de Trabalho de Conclusão de Curso 2.

Orientador: Dra. Eliana Pantaleão

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Elétrica

Bacharelado em Engenharia Eletrônica e de Telecomunicações

Patos de Minas

2019

Resumo

O uso crescente das redes sociais em todo o mundo tem estimulado as pessoas a expressarem e compartilharem seus sentimentos e opiniões sobre determinado tópico ou assunto. A partir dos padrões destes dados pode-se criar modelos de predição utilizando técnicas de Processamento de Linguagem Natural em conjunto com algoritmos de Aprendizado de Máquinas. Este trabalho propõe a criação de um modelo de classificação de *tweets* depressivos utilizando o método de reconhecimento de padrões SVM (*Support Vector Machine*), tendo como fonte de dados dois *datasets* de domínio público retirados à partir da rede social Twitter. O ambiente de desenvolvimento utilizado foi o *Jupyter Notebook*, onde a partir de várias bibliotecas auxiliares foram implementadas técnicas de pré-processamento nos *tweets* para eliminar informações indesejáveis e o treinamento do modelo construído obtendo-se uma taxa de acerto de 99.7% com os dados de treinamento após o modelo ser otimizado utilizando a técnica de Grade de Busca. A partir do modelo criado obteve-se classificações corretas de *tweets* não utilizados no treinamento como pertencentes à classe depressiva ou não-depressiva.

Palavras-chave: Twitter, Processamento de Linguagem Natural, Aprendizado de Máquina, Depressão, Classificação de Dados, Máquinas de Vetores de Suporte, Scikit-learn, Otimização de Parâmetros.

Abstract

The increase of social network usage around the world has encouraged people to express and share their feelings and opinions on a particular topic or subject. From the patterns of these data, one can create prediction models using Natural Language Processing techniques in conjunction with Machine Learning algorithms. This work proposes the creation of a classification model of depressive tweets using the method of pattern recognition SVM (Support Vector Machine), having as data source two public domain datasets taken from the social network Twitter. The development environment used was the Jupyter Notebook, where, with the help of several auxiliary libraries, preprocessing techniques were implemented and applied on the tweets to eliminate undesirable information. The training of the constructed model obtaining a rate of success of 99.7% with the data of training after the model is optimized using the Search Grid technique. Using the created model, it was possible to label new tweets as depressive or non-depressive.

Keywords: Twitter, Natural Language Processing, Machine Learning, Depression, Data Classification, Support Vector Machines, Scikit-learn, Parameter Optimization.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface - Interface de Programação de Aplicação
ASCII	American Standard Code for Information Interchange - Código Padrão Americano para Intercâmbio de Informações
IDF	Inverse Document Frequency - Frequência de Documento Inverso
NLTK	Natural Language Toolkit - Kit de Ferramentas de Linguagem Natural
SVM	Support Vector Machine - Máquina de Vetores de Suporte
TF	Term Frequency - Frequência de Termo
TF-IDF	Term Frequency-Inverse Document Frequency - Frequência de Termo-Frequência de Documento Inverso

Sumário

1 Introdução

Juntamente com o crescimento do acesso à Internet, é notável o crescimento do uso das redes sociais na última década. Dentre as centenas de redes sociais, o Twitter é uma plataforma consolidada e existem milhares de pessoas que o usam constantemente com a intenção de exteriorizar suas opiniões e sentimentos sobre algum tópico em forma de texto e com uma restrição de 140 caracteres. Em abril de 2019, a empresa revelou um tráfego médio diário de 134 milhões de usuários, seja publicando ou lendo publicações de outros usuários (FOLHA, 2019). O conjunto de caracteres de cada uma das publicações é usualmente denominado de *tweet*.

Milhões de pessoas passam grande parte do tempo na Internet em redes sociais e isso permite a elas manifestarem o que estão sentindo e pensando em determinado momento, sobre determinado tópico ou assunto. Em 2007, estimava-se que 5000 *tweets* eram postados por dia (WEIL, 2010). O aumento da popularidade do Twitter e da presença das pessoas em redes sociais, no geral, fez o número crescer para 500 milhões de *tweets* diários em 2018 (ASLAM, 2018).

Devido a essa enorme quantidade de publicações através da plataforma, o Twitter se tornou uma grande fonte de informação sobre diversas questões da vida diária da sociedade mundial e diversas análises e experimentos de aprendizado de máquina (*machine learning*) têm sido utilizados para colher informações e analisar tendências. Nas eleições presidenciais americanas, por exemplo, o Twitter foi usado para monitorar a análise de sentimento dos usuários perante aos candidatos, sendo portanto, uma ferramenta estratégica importante para as chapas eleitorais saberem o que estava sendo repercutido de forma positiva ou negativa sobre as propostas de governo a serem implementadas (KAPKO, 2016).

O corpo de texto desses *tweets* pode conter padrões em que pode se identificar a predisposição e/ou a presença do distúrbio de depressão. Neste contexto, técnicas de processamento de linguagem natural e aprendizado de máquina são utilizadas no conjunto de dados de treinamento, que são os próprios *tweets* , com a finalidade de criar um modelo preditivo que irá classificar um *tweet* nunca antes visto como contendo sintomas de depressão ou não. Um exemplo de avaliação de sentimentos com base em *tweets* é a ferramenta *Tweet Sentiment Visualization*, disponibilizada pelo Departamento de Ciência da Computação da *North Carolina State University* (HEALEY; RAMASWAMY, 2011). Nesta ferramenta, é possível digitar uma palavra, como por exemplo a marca de um produto ou o nome de uma pessoa famosa, e *tweets* recentes contendo essa palavra são avaliados considerando duas dimensões: um eixo vai de

unpleasant/ até *pleasant*, enquanto o outro vai de *subdued* até *active*. RAMASWAMY (2011) explica a teoria utilizada na criação da ferramenta e sugere que a avaliação por palavras individuais é mais recomendada em *tweets* por sua natureza curta e desprovida de sutilezas gramaticais, apesar de suas limitações.

Muitas das pessoas que se sentem depressivas talvez nunca tiveram a chance de ir a um psicólogo ou por não terem condições ou por acharem que não é necessário. Nestes casos o Twitter acaba se tornando uma alternativa de desabafo e uma forma de diário pessoal para as pessoas discutirem sobre os sintomas que estão sentindo, o que tem afetado em sua vida e talvez até se informarem com dicas de como amenizar este estado mental. Estes textos de desabafo sobre sintomas de depressão podem indicar padrões de linguagem que são usados por pessoas com depressão e assim contribuir para que se possa classificar qualquer *tweet* em duas classes: depressivo ou não-depressivo. Utilizando a API que o Twitter disponibiliza gratuitamente, consegue-se filtrar e captar *tweets* em tempo-real que contenham palavras-chaves como: (i) *depression*, (ii) *anxiety* (iii) *mental health*, e assim pode-se ter um conjunto de dados mais direcionado às causas da depressão.

É necessário destacar que toda abordagem do trabalho até o alcance do seu objetivo foi em cima de textos na língua inglesa. Quando se lê um artigo científico ou se estuda a documentação de uma biblioteca em Python, é natural cada vez mais estar em contato com a língua inglesa pois é a língua mais utilizada no mundo e está somente atrás da língua chinesa (MYERS, 2015). Além disso, a maioria da comunidade científica, que contribui com bibliotecas de códigos para projetos de processamento natural de linguagem, direciona seus esforços intelectuais para resolução de problemas na língua inglesa. Portanto, utilizar-se deste lago de conhecimento à disposição de todos de forma livre favorece o trabalho e lhe concede um ganho e maior flexibilidade para criar novas soluções.

1.1 Objetivos

Neste trabalho criou-se um modelo de classificação de *tweets* depressivos, utilizando um classificador SVM (*Support Vector Machine* - Máquina de Vetores de Suporte), a partir de um conjunto de dados composto por *tweets* depressivos e não-depressivos, obtidos a partir de repositórios de domínio público. Os objetivos específicos incluem a avaliação de técnicas de pré-processamento de texto, com o uso de métodos provenientes da área de processamento de linguagem natural, a avaliação do modelo de classificação e a otimização dos parâmetros do classificador para tornar o modelo mais assertivo.

1.2 Organização do Trabalho

No Capítulo 2 apresenta-se fundamentações teóricas sobre os conceitos e principais técnicas de processamento de linguagens naturais utilizadas no pré-processamento de dados. Aborda-se ainda uma introdução sobre os tipos de aprendizado de máquinas existentes assim como uma explicação detalhada das máquinas de vetores de suporte (SVM), extração de características e otimização de parâmetros. No Capítulo 3 expõe-se a metodologia utilizada bem como a implementação do modelo de classificação e os resultados obtidos após a criação do modelo. Por fim, no Capítulo 4 apresenta-se as conclusões deste trabalho e sugestões de trabalhos futuros.

2 Fundamentação Teórica

2.1 Processamento de Linguagem Natural

A atividade de pré-processar textos é uma importante etapa no estudo de técnicas de Mineração de Dados na área de Processamento de Linguagem Natural. Ela consiste em eliminar palavras, pontuações e símbolos que não carregam importância em um conjunto de texto e podem ser removidas sem nenhum problema maior para o entendimento do que foi dito. Consequentemente, somente palavras com maior informação, ou seja, as mais raras, são geralmente mantidas e utilizadas para treinamento, para que o modelo a ser criado possa ser eficiente.

Alguns importantes métodos de pré-processamento de texto são: palavras vazias, stemização (*stemming*), lematização (*lemmatizing*) e TF/IDF (*Term Frequency - Inverse Document Frequency*).

2.1.1 Palavras vazias

Usa-se esta técnica para eliminar a quantidade de palavras que não são importantes para o objetivo que se deseja alcançar com o processamento do texto. As palavras mais comuns de se encontrar em um texto são preposições, artigos e pronomes, como *the*, *a*, *an*, *in*. Embora gramaticalmente importantes na construção das sentenças, elas não possuem um significado relevante, portanto, elas são consideradas palavras vazias (*stop words*) e são retiradas no processo anterior ao de se analisar um texto em algoritmos de aprendizado de máquinas. Outra vantagem que se tem quando se usa esta técnica é a diminuição do tamanho do texto e do tempo de processamento destes dados.

2.1.2 Stemização

Em um texto podem ter palavras que têm a mesma semântica, porém, sufixos diferentes. A técnica de Stemização (*Stemming*) nada mais é do que relativizar todas as palavras para um único radical cortando seu sufixo. Um texto que contenha as palavras *agreed*, *agreeing* e *agreement*, depois ter sido aplicado a técnica de Stemização, as três palavras são convergidas em um único radical *agree*. É importante notar que a Stemização é melhor aproveitada em linguagens que possuem uma morfologia menos complexa, como o inglês, pois, quanto mais complexa a linguagem mais as palavras serão diferentes semanticamente fazendo com que na etapa de transformação para apenas um radical ocorra perda de informações.

2.1.3 Lematização

A técnica de Lematização (*Lemmatizing*) é usada para transformar os radicais de palavras em lemas com base em um vocabulário e análise morfológica das palavras. Ao contrário da técnica de Stemização, que apenas elimina o final das palavras sem analisar o contexto em que elas se encontram, a técnica de lematização transforma cada palavra em um radical analisando o contexto em que a palavra se encontra e sua classe gramatical. Por exemplo, a palavra *saw* se fosse "stemizada" iria se tornar 's'; no processo de lematização o algoritmo iria retornar '*see*' ou '*saw*' dependendo se a palavra for um substantivo ou um verbo dentro do contexto em que se encontra (MANNING; RAGHAVAN; SCHüTZE, 2008).

2.1.4 $TF - IDF$

A *TF-IDF* (*Term Frequency - Inverse Document Frequency*) é um dado estatístico que demonstra o quão importante uma palavra é para um documento qualquer dentro de uma coleção de documentos.

Cada documento possui um comprimento e quantidade de palavras diferente. Para se calcular a frequência de um termo conta-se a quantidade de vezes que ele aparece em um documento D dividido pela quantidade de termos que existem em D.

TF significa Frequência do Termo (*Term Frequency*) e mede o quão frequente um termo ou palavra aparece em um dado documento. Para saber a importância de um termo deve-se analisar a frequência deste termo nos outros documentos da coleção, ou em outras palavras, calcular a IDF, *Inverse Document Frequency* ou Inverso da Frequência nos Documentos.

$$TF(palavra) = \frac{\text{Número de vezes que a palavra aparece no documento}}{\text{Número total de termos do documento}}$$

Caso o termo em questão se repetir várias vezes também em outros documentos, considera-se que não é um termo importante e seu peso final será baixo, pois como é um termo comum supõe-se que ele não carrega tanta informação quando se analisa um documento de forma independente, ignorando a coleção.

Palavras de menor frequência perante o conjunto inteiro de documentos e maior frequência em documentos individuais, são consideradas palavras raras e, portanto, mais relevantes e carregadas de informações importantes para a compreensão do texto. O IDF é calculado usando a seguinte equação:

$$IDF(\text{palavra}) = log_e \left(\frac{\text{Número total de documentos}}{\text{Número de documentos que contêm palavra}} \right).$$

O escalar que representa *TF-IDF* é obtido multiplicando $TF \times IDF$. Um exemplo: considere um documento que contenha 100 palavras onde a palavra cachorro aparece 4 vezes. A frequência do termo (i.e., TF) para o cachorro é então $4/100 = 0{,}04$. Assuma

agora que existe uma coleção de 5 milhões de documentos e a palavra cachorro aparece em 1000 destes documentos. Então, a frequência de documento inversa (i.e., IDF) é calculada com o $log_e\left(\frac{5.000.000}{1000}\right) = 3{,}69$. Assim, o *TF-IDF* é o produto $0{,}04 \times 3{,}69 = 0{,}14$.

2.2 Aprendizado de máquina e reconhecimento de padrões

A construção de máquinas com capacidade de aprendizado por experiência tem sido um objeto de debate técnico e filosófico, nas últimas décadas. No aspecto técnico, o avanço de dispositivos eletrônicos de alta performance e da capacidade de operações lógicas/matemáticas que um processador faz por segundo, tem estimulado bastante o uso desta capacidade inerente dos computadores para resolverem problemas mundanos de uma forma confiável.

Existem milhares de problemas que não podem ser resolvidos por técnicas clássicas de programação. Imagine a situação em que deve-se criar um algoritmo que irá controlar os movimentos de um carro como o objetivo de levá-lo de um ponto A a um ponto B. A quantidade de objetos variáveis que os sensores do carro devem captar é grande, por exemplo, algumas características que devem ser analisadas seriam: o número de carros a frente e atrás, diferentes placas com significados diferentes, pedestres atravessando a via, etc. Os sensores e câmeras teriam que monitorar ao mesmo tempo todas essas características e ainda tomar decisões com base no que detectam no contexto do ambiente em que se encontram.

Seria complexo e árduo o trabalho se fosse utilizada a programação clássica e o arquiteto do sistema não conseguiria especificar o método pelo qual teria a resposta correta de saída a partir dos dados de entrada. Além disso, ele teria que se preocupar com todos os possíveis dados de entrada, que podem ser infinitos e com padrões distintos. Consequentemente, a criação de sistemas com capacidade de aprendizagem têm sido explorada para criar-se modelos matemáticos ou funções que generalizam a saída a partir de dados de entrada. Cada algoritmo de aprendizado de máquina tem estruturas que definem como será a modelagem e as decisões que devem ser tomadas com base nos dados de entrada. Os aprendizados supervisionados e os não-supervisionados são exemplos de paradigmas de aprendizado comumente usados.

2.2.1 Aprendizado Supervisionado

O aprendizado supervisionado é uma técnica em que se fornece dados de treinamento contendo o dado de entrada, juntamente com sua respectiva classe. Essa classe foi definida previamente baseado no problema para se solucionar. Quando se treina estes dados, é possível identificar padrões associados às classes e assim criar um

classificador que irá matematicamente identificar dados nunca antes vistos e classificá-los em sua respectiva classe.

2.2.2 Aprendizado Não-supervisionado

Ao contrário do aprendizado supervisionado, o aprendizado não-supervisionado não possui os dados já definidos em classes previamente classificadas. Como o algoritmo somente tem os dados como base de criação de um modelo, ele identifica dados que possam ter padrões semelhantes e são criados subconjuntos que se aglomeram em possíveis classes que são destinadas aos dados.

2.2.3 Aprendizado Semi-supervisionado

O aprendizado semi-supervisionado é uma junção dos paradigmas de aprendizado supervisionado e não-supervisionado. Este paradigma normalmente é usado quando existe alguma dificuldade em se obter dados rotulados para o problema. Dados sem rótulos são mais fáceis de se obterem devido à grande quantidade de informação não-estruturada que existe na internet. As técnicas de separação de classes em *clusters* são usada nos dados não-estruturados. Após distinção no espaço das possíveis classes, um humano irá rotular cada *cluster* e ir com isso melhorando a acurácia do modelo que agora irá agrupar os dados por similaridade e além disso saber a qual classe o *cluster* representa.

2.2.4 Máquinas de Vetores de Suporte

A máquina de vetores de suporte (SVM - *Support Vector Machine*) é um método de reconhecimento de padrões que possui uma elevada capacidade de generalização e arquitetura relativamente simples. Normalmente, é utilizado para tarefas de classificação devido a sua predisposição de distinguir padrões através de um hiperplano ótimo que tenha a maior margem possível entre dados pertencentes às classes envolvidas no problema.

Os hiperplanos são essenciais ferramentas para criar máquinas de vetores de suporte para problemas de visão computacional e processamento de linguagem natural. A dimensão de um hiperplano é sempre uma abaixo do espaço que está inserido, de forma que para um dado espaço N, o hiperplano terá dimensão $N - 1$. O hiperplano sempre divide o espaço em duas metades. Portanto, existirá um hiperplano ótimo que irá satisfazer as condições do problema e separar o espaço em classes diferentes referente aos dados analisados (Wikipedia contributors, 2019).

O objetivo do método de máquinas de vetores de suporte (*Support Vector Machines* - SVM) é encontrar uma reta que separa os dados positivos dos negativos. Existem infinitas retas que podem ser traçadas para separar os dados, porém, existe somente uma que irá ter a maior margem possível entre os dados positivos e negativos. O problema resume-

se à resolução de uma equação que maximiza a distância entre os dados positivos dos negativos e usa-se técnicas de álgebra linear e multiplicadores de Lagrange para atingir este objetivo.

Suponha-se que exista um vetor w qualquer que é ortogonal ao hiperplano ou reta que separa o espaço das amostras em duas regiões.

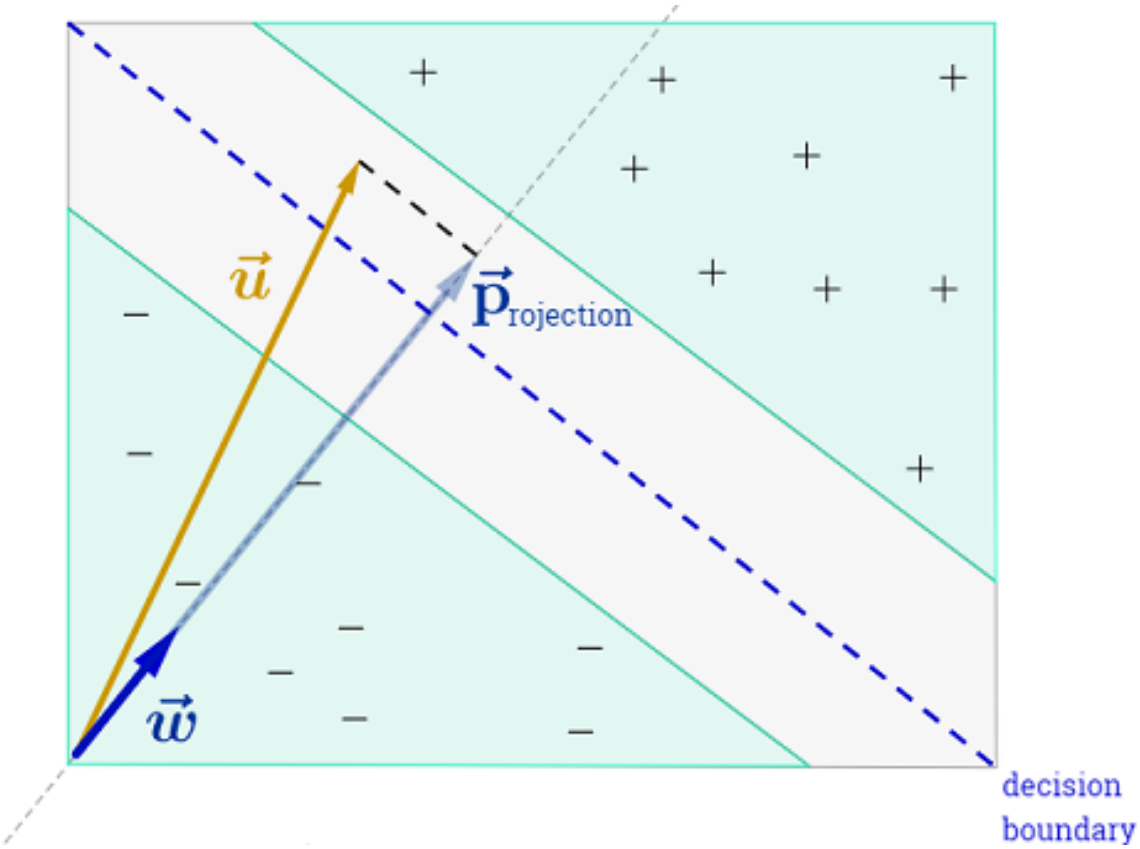

Figura 1 – Representação de um hiperplano separando os dados positivos dos negativos.

Fonte: Parellada (2017)

A Figura 1 possui três regiões: a região dos dados positivos, a dos negativos e a região central que é a região que separa os dados e contém a mediana que representa a borda de decisão. Usando como analogia uma rua, a distância entre as bordas da calçada na parte direita e esquerda é a maior distância entre os dados possível.

O vetor u representa a posição no espaço de um dado desconhecido e quando se faz o produto escalar de w com u, para identificar como u deve ser classificado, percebe-se que quanto mais o vetor u é maior, mais próximo dos dados da classe positiva ele está. Ao contrário, quanto menor o vetor u, mais próximo dos dados da classe negativa ele está.

O produto escalar de w e u sempre é maior ou igual a uma constante

$$w \cdot u \geq C \tag{2.1}$$

Para generalizar este caso, considera-se que $b = -C$. Então:

$$w \cdot u + b \geq 0 \tag{2.2}$$

Se a Equação 2.2 é verdadeira, então u é uma amostra positiva. Essa é definida como nossa regra de decisão.

Por conveniência, a primeira restrição é que para um dado ser uma amostra positiva, a equação da regra de decisão deve ser maior ou igual a 1. Para que seja uma amostra negativa, a regra de decisão deve ser menor ou igual a -1. As equações são mostradas a seguir:

$$w \cdot x_+ + b \geq +1 \tag{2.3}$$

$$w \cdot x_- + b \leq -1 \tag{2.4}$$

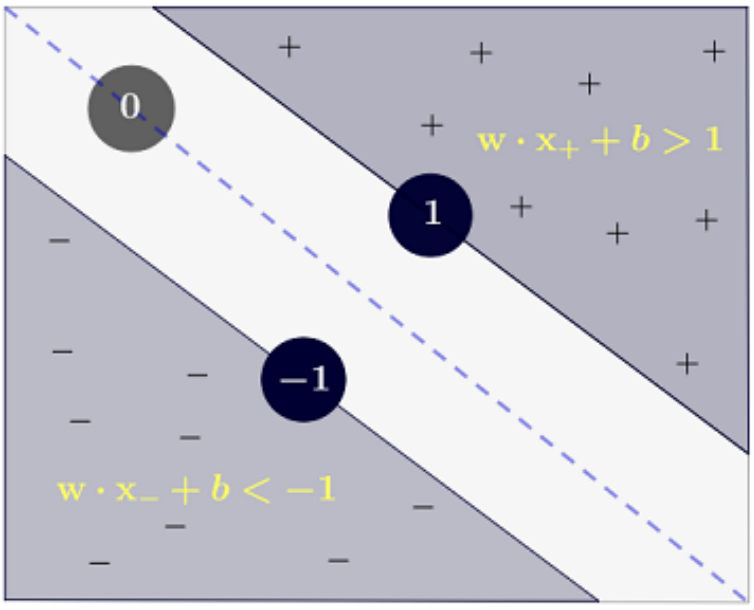

Figura 2 – Representação dos valores nas bordas e na mediana.

Fonte: Parellada (2017)

A Figura 2 nos mostra que para valores na borda direita, os dados sempre serão positivos (+1) e na borda direita sempre negativos (-1).

Cria-se por uma conveniência matemática uma variável Y_i tal que $Y_i = +1$ para amostras positivas e $Y_i = -1$ para amostras negativas. Portanto, de agora em diante para cada amostra tem-se o valor de Y_i e este valor irá determinar se é uma amostra positiva ou uma amostra negativa.

Multiplica-se a variável Y_i pela Equação 2.12, onde $Y_i = 1$, e 2.16, onde $Y_i = -1$:

$$Y_i(w \cdot x_i + b) \geq 1 \tag{2.5}$$

$1(w \cdot x_i + b) \geq 1$

$w \cdot x_i + b \geq 1$

$w \cdot x_i + b - 1 \geq 0$

$$Y_i(w \cdot x_i + b) \leq -1 \tag{2.6}$$

$-1(w \cdot x_i + b) \leq -1$

$-w \cdot x_i - b \leq -1$

$(-1) - w \cdot x_i - b \leq (-1) - 1$

$w \cdot x_i + b \geq 1$

$w \cdot x_i + b - 1 \geq 0$

Devido à variável Y_i, inserida por conveniência, é possível notar que o valor para as duas equações das bordas se torna o mesmo.

$$Y_i(w \cdot x_i + b) - 1 \geq 0 \tag{2.7}$$

É importante salientar que para amostras que se encontram nas bordas entre os dados positivos e negativos, a equação anterior resume-se a:

$$Y_i(w \cdot x_i + b) - 1 = 0 \tag{2.8}$$

É necessário descobrir a largura da distância de separação do hiperplano em relação aos dados, que pode ser dado pela diferença de dois vetores que se encontram na borda do lado positivo e do negativo.

Assumindo novamente que existe um vetor w perpendicular à borda de decisão, a largura máxima é definida como o produto escalar da diferença entre um ponto negativo

e outro positivo que estejam na borda com o vetor unitário perpendicular, w.

$$Largura = (x_+ - x_-) \cdot \frac{w}{\|w\|} \tag{2.9}$$

Considerando a Equação 2.9, a expressão que representa dados que estão na borda próxima aos dados positivos, x_+, levando em consideração que $Y_i = 1$, é:

$$Y_i(w \cdot x_i + b) - 1 = 0 \tag{2.10}$$

$$+1(w \cdot x_i + b) = 1 \tag{2.11}$$

$$w \cdot x_i = 1 - b \tag{2.12}$$

A expressão que representa dados que estão na borda próxima aos dados negativos, x_-, levando em consideração que Y_i = -1, é:

$$Y_i(w \cdot x_i + b) - 1 = 0 \tag{2.13}$$

$$-1(w \cdot x_i + b) = 1 \tag{2.14}$$

$$-w \cdot x_i - b = 1 \tag{2.15}$$

$$w \cdot x_i = -1 - b \tag{2.16}$$

Assim, voltando à Equação 2.9 e substituindo o valor de 2.12 e 2.16, encontra-se a relação:

$$Largura = (x_+ - x_-) \cdot \frac{w}{\|w\|} \tag{2.17}$$

$$Largura = \frac{w \cdot x_{i+} - w \cdot x_{i-}}{\|w\|} \tag{2.18}$$

$$Largura = \frac{(1 - b) - (-1 - b)}{\|w\|} \tag{2.19}$$

$$Largura = \frac{2}{\|w\|} \tag{2.20}$$

Agora que o valor da *Largura* foi definido, é fundamental maximizar essa distância entre os dados que é chamada de "largura da rua", para garantir que os dados das duas classes tenham a maior separação possível. Deve-se maximizar $\frac{2}{\|w\|}$, então pode-se dizer analogamente que é possível também minimizar $\|w\|$ ou minimizar $\frac{1}{2} \cdot \|w\|^2$, sendo a última forma a mais matematicamente conveniente.

Até agora tem-se um problema de minimização com algumas condições a serem cumpridas. Para esse tipo de problema, onde existe a minimização ou maximização de algum parâmetro respeitando condições impostas, faz-se o uso de multiplicadores de Lagrange.

$$L = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i [Y_i(w \cdot x_i + b) - 1] \tag{2.21}$$

Deriva-se a Equação 2.21 em relação ao vetor w,

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i Y_i x_i = 0 \tag{2.22}$$

$$w = \sum_i \alpha_i Y_i x_i \tag{2.23}$$

Deriva-se a Equação 2.21 em relação ao valor de b,

$$\frac{\partial L}{\partial b} = -\sum_i \alpha_i Y_i = 0 \tag{2.24}$$

$$\sum_i \alpha_i Y_i = 0 \tag{2.25}$$

Aplicando o valor de w na Equação 2.21, o vetor de decisão é uma soma linear das amostras:

$$L = \frac{1}{2}(\sum_i \alpha_i Y_i x_i)(\sum_j \alpha_j Y_j x_j) - (\sum_i \alpha_i Y_i x_i)(\sum_j \alpha_j Y_j x_j) - \sum_i \alpha_i Y_i b + \sum_i \alpha_i \tag{2.26}$$

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j Y_i Y_j x_i \cdot x_j \tag{2.27}$$

A otimização que visa maximizar a distância entre os dados e o hiperplano depende somente do produto escalar das amostras.

Voltando para a regra de decisão, substitui-se o w encontrado na expressão $w \cdot u + b \geq 0$ e descobre-se que $\alpha_i \cdot yi \cdot xi \cdot u + b \geq 0$, representando assim uma amostra positiva neste caso.

Entende-se então que a regra de decisão do modelo de máquinas de vetor suporte somente depende do produto escalar das amostras de treinamento com os dados desconhecidos que serão classificados pelo algoritmo.

Nas implementações deste método, é comum o uso de um parâmetro C, que representa o valor máximo permitido para α_i, os multiplicadores de Lagrange. Este parâmetro atua como uma penalidade para os pontos que ficam com a classificação incorreta. Ou seja, quanto maior o valor de C, menos pontos de treinamento com classificação incorreta são permitidos na criação do hiperplano. Isso cria um modelo mais rígido, com menor poder de generalização.

Existem casos em que os dados de treinamento não podem ser separados corretamente por um hiperplano utilizando o conceito de maximizar a margem de separação entre dados de classes diferentes. Nestes casos em que os dados não são linearmente separáveis, cria-se um novo espaço de atributos em que estes dados possam ser separáveis.

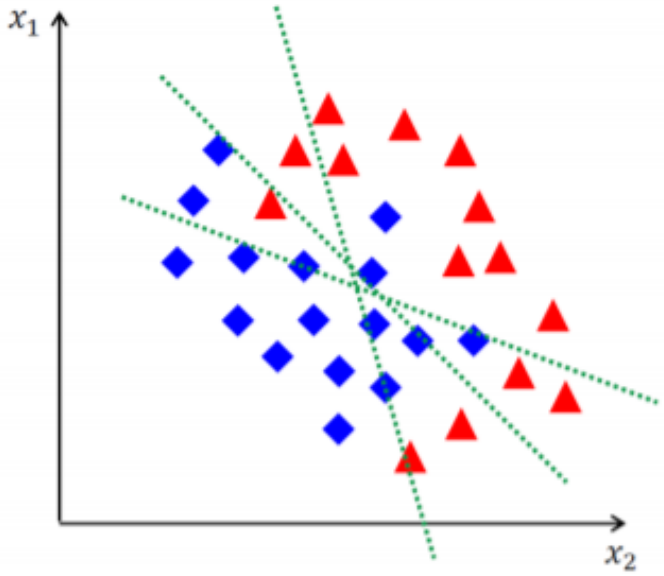

Figura 3 – Espaço onde os dados não podem ser separados linearmente por um hiperplano.

Fonte: (NEGRI, 2013)

O que transpõe os dados de um espaço para o outro é uma função K denominada *kernel* que é aplicada no produto escalar $x_i \cdot x_j$ da Equação 2.27, resultando na Equação 2.28.

$$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j Y_i Y_j K(x_i \cdot x_j) \tag{2.28}$$

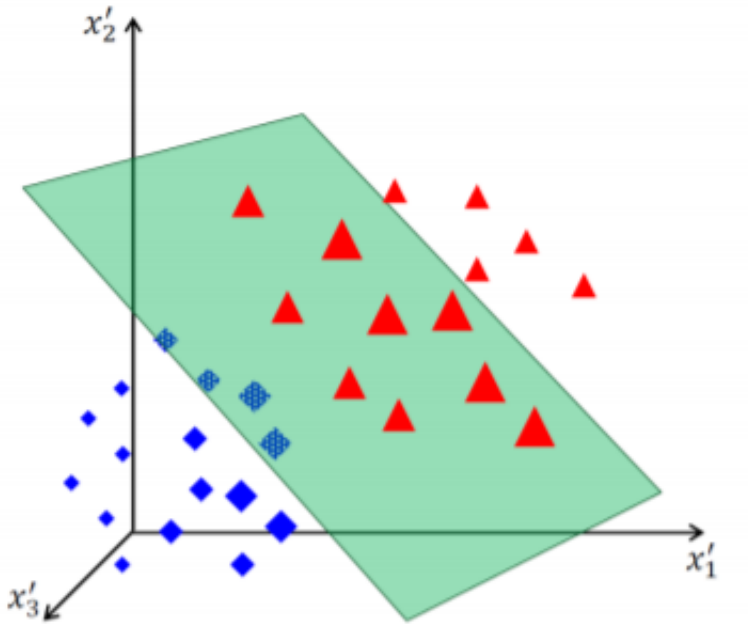

Figura 4 – Novo espaço onde os dados podem ser separados linearmente por um hiperplano.

Fonte: (NEGRI, 2013)

2.3 Extração de Características

Os valores que são usados pelo classificador para estabelecer a fronteira de decisão são organizados em vetores, usualmente denominados de vetores de características. Cada vetor corresponde a um padrão de treinamento, ou seja, um objeto de entrada. A etapa de construção e organização desses vetores é conhecida como a fase de extração de características. Um exemplo de método de extração de características é o *CountVectorizer*, que pode ser aplicado sobre palavras isoladas ou agrupadas, como bigramas, trigramas ou n-gramas.

2.3.1 Bigramas

Bigrama é por definição um n-grama onde o valor de n=2. O bigrama é a junção de dois *tokens* adjacentes em um conjunto de letras, sílabas ou palavras.

2.3.2 CountVectorizer

Para que seja possível criar um modelo de aprendizado de máquinas que classifica textos é necessário extrair as principais características dos documentos que serão analisados. A maioria dos algoritmos de *machine learning* não estão preparados para receber entradas de texto, portanto, estes documentos devem passar por um

processo de vetorização para que sejam convertidos em valores numéricos, tornando assim possível os dados se acomodarem em um modelo matemático que irá generalizar os problemas a serem classificados. A biblioteca *scikit-learn* fornece em sua biblioteca *feature_extraction* (extração de características) a função *CountVectorizer()*, que é responsável pelo processo de vetorização. A função *CountVectorizer()* é o encarregado de tokenizar as sentenças de todos os documentos e contar o número de ocorrências de cada *token* em cada documento, além de normalizar os *tokens* por importância baseando-se na quantidade de vezes que cada *token* aparece em cada documento dividido pelo número total de documentos. A característica ou *feature* de um documento é definida pela quantidade de tokens que estão presentes neste documento. As *features* das frase *"How have you been?"* e *"It's been a while!"*, são [*'how', 'have', 'you', 'been'*] e [*'it', 'is', 'been', 'a', 'while'*], respectivamente. O vocabulário fixo das frases anteriores seria [*'how', 'have', 'you', 'been', 'it', 'is', 'a', 'while'*] e a representação vetorizada de cada frase nesse vocabulário seria:

Tabela 1 – Representação vetorizada de dois documentos.

Vocabulário	**how**	**have**	**you**	**been**	**it**	**is**	**a**	**while**
Frase1	1	1	1	1	0	0	0	0
Frase2	0	0	0	1	1	1	1	1

Em casos reais, o vocabulário chegaria a uma dimensão de 100.000 palavras e cada documento teria muito mais zeros do que um. Para que o custo computacional não seja um problema no momento de treinamento dos dados, a matriz de saída do método *CountVectorizer()* é uma matriz esparsa (*sparse matrix*). Matrizes esparsas são matrizes onde a maioria dos dados são bytes vazios. A matriz é então comprimida contendo apenas os dados reais e somente na etapa de decodificação que os zeros são novamente gerados. Esta técnica aumenta muito a eficiência computacional quando se trabalha com uma enorme quantidade de dados. Desta forma, cada coluna contém um *token* do dicionário e cada linha i contém um documento i que é representado por uma combinação linear das características do documento com o dicionário onde contém todos os *tokens*.

2.4 Otimização de parâmetros

Quando se treina um classificador SVM é necessário estipular alguns parâmetros que irão determinar o comportamento daquele modelo. No caso do SVM, a combinação de parâmetros iniciais é C e o *kernel*. A variação destes parâmetros gera sempre um modelo diferente do outro e com capacidades de generalizações diferentes, porém, existe um conjunto de parâmetros ótimos que irá criar o melhor modelo possível.

O método de otimização de parâmetros se utiliza do método da Grade de Busca (*Grid-searching*). Este método irá construir um modelo para cada combinação possível

de parâmetros que lhe foi dado e será feito uma validação cruzada (*cross-validation*) para cada modelo. A função da Grade de Busca consegue avaliar a performance de acerto para cada modelo resultante de cada combinação e no final determina qual é a melhor combinação dos parâmetros que deve ser utilizada para se obter a melhor generalização do modelo.

Este método de otimização pode ser aplicado para diversos modelos dentro da área de aprendizado de máquina para calcular os melhores parâmetros para qualquer modelo.

2.4.1 Validação Cruzada

Validação cruzada (*Cross-validation*) é um método estatístico para avaliar e comparar algoritmos de aprendizado por meio da divisão dos dados em dois segmentos: um utilizado para treinar um modelo e outro usado para validar o modelo (REFAEILZADEH; TANG; LIU, 2009). O método se mostra uma forma mais complexa e menos tendenciosa de analisar a performance de um modelo.

O processo de validação cruzada tem como parâmetro único uma variável chamada k que se refere ao número de grupos em que um *dataset* irá ser repartido. O valor de k normalmente é 5 ou 10, mas tudo depende dos dados em que se está trabalhando. A Figura 5 representa o método de validação cruzada para $k = 5$.

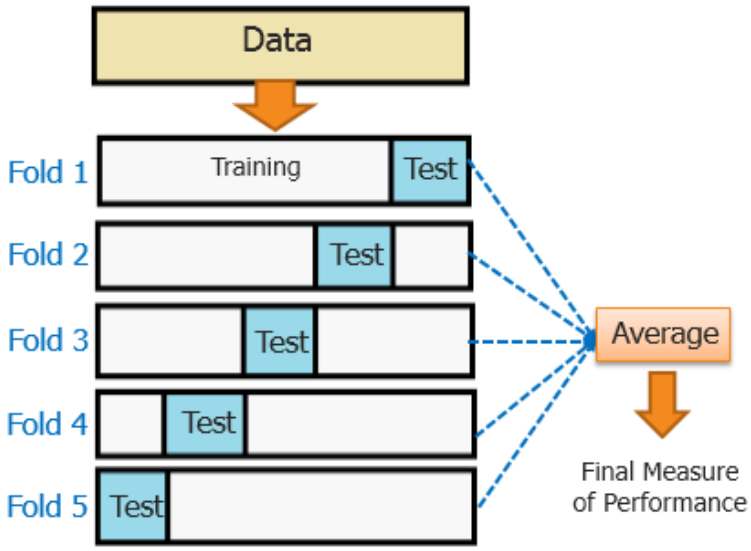

Figura 5 – Representação do método de validação cruzada para $k = 5$.

Fonte: Srinidhi (2018)

Existe um risco de se perder informações quando se divide os dados entre dados de treinamento e validação, pois, podem existir padrões importantes nos dados de validação que também podem ser usados para alimentar o modelo durante o treinamento.

No método de validação cruzada os dados de treinamento são divididos em k subconjuntos. Cada subconjunto é utilizado uma vez como os dados de validação e o restante $k-1$ para treinamento. Este procedimento é realizado k vezes. Após a execução do procedimento tem-se o erro estimado para cada subconjunto e assim pode-se tirar a média de erro do modelo e descobrir qual é o melhor arranjo possível para se treinar os dados em que o modelo terá a melhor generalização.

3 Desenvolvimento

3.1 Metodologia

A metodologia utilizada neste trabalho foi divida em duas partes: uma parte de pré-processamento e outra de treinamento. A Figura 6 ilustra os procedimentos empregados.

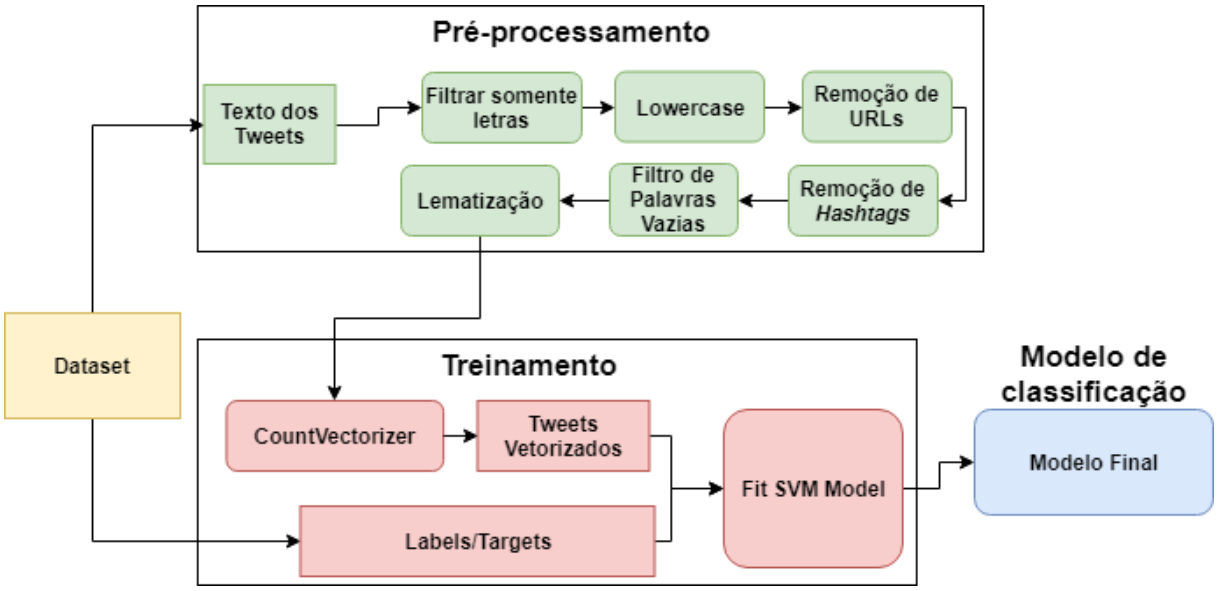

Figura 6 – Fluxograma da metodologia utilizada para o desenvolvimento do modelo de classificação.

Fonte: o autor

O *dataset* é composto por duas partes: o texto do *tweet* e um rótulo (*Labels/Targets*) que indica se o *tweet* é depressivo ou não-depressivo. O texto do *tweet* passa inicialmente pela etapa de pré-processamento. Nessa etapa, os *tweets* passam por filtros que eliminam caracteres indesejados como *hashtags*, *websites*, palavras vazias e até uma função de lematização. Em seguida, o resultado dessa etapa e mais os rótulos são usados na etapa de Treinamento onde os *tweets* são vetorizados e preparados para serem inseridos no modelo a ser treinado. Após o treinamento obtemos um modelo SVM capaz de classificar *tweets* em duas classes: depressivos ou não-depressivos.

3.2 Implementação

3.2.1 Ambiente de desenvolvimento

O ambiente que foi utilizado para realizar todos os experimentos foi o Jupyter Notebook (JUPYTER, 2019). Nele é possível organizar o código em blocos independentes onde é possível ter uma saída instantânea para cada bloco, conceito conhecido como *live-code*.

Todos os códigos do trabalho são feitos na linguagem de programação Python, portanto, o *kernel* utilizado no Jupyter Notebook foi o do Python 3.6, porém, a plataforma tem suporte para linguagens como Julia e C++.

Python é uma linguagem amplamente utilizada na solução de problemas de aprendizado de máquina devido à existência de inúmeras bibliotecas destinadas à manipulação de dados, treinamento, otimização de parâmetros, estrutura de dados, etc. Neste trabalho, foi usada a biblioteca Pandas para o armazenamento dos dados, *Numpy* para o processamento matemático, *Scikit-learn* para o aprendizado de máquina e NLTK para o processamento de linguagem natural.

A biblioteca Pandas (NUMFOCUS, 2019) é uma biblioteca que é utilizada para o armazenamento dos dados em estruturas convenientes chamadas de *DataFrame* e *Series*. Um *DataFrame* é composto por 2 ou mais *Series*, que equivalem a colunas. A biblioteca também disponibiliza diversos métodos para converter formatos de arquivos.

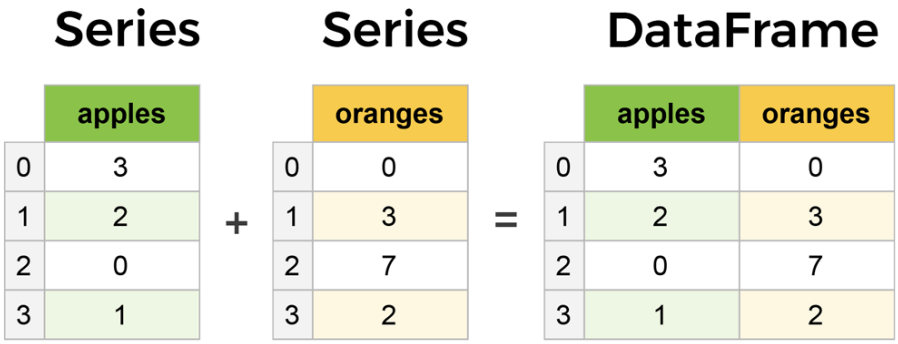

Figura 7 – Estruturas do panda.

Fonte: (NUMFOCUS, 2019)

O *Numpy* (SCIPY, 2019) é uma biblioteca otimizada para processamento de funções matemáticas que exigem alto nível de processamento. Ela transporta os dados para a memória e os aloca com o mesmo tipo, portanto, não é necessário a checagem dos tipos de dados antes de fazer qualquer tipo de operação possibilitando o processamento rápido de grandes volumes de dados.

A biblioteca NLTK (BIRD; KLEIN; LOPER, 2009) contém funções essenciais para o processamento de linguagem natural como a stemização, lematização, palavras vazias e é otimizado para a língua inglesa, porém, oferece suporte também para outros 14 idiomas, como Português, Russo e Espanhol.

A biblioteca *Scikit-learn* (PEDREGOSA et al., 2011) é uma das mais utilizadas para a resolução de problemas de aprendizado de máquinas devido sua enorme gama de modelos de aprendizado disponíveis e das demais ferramentas que são utilizadas para manipular os dados e deixá-los preparados para os modelos. Ela foi criada em cima de estruturas do *Numpy* para que o custo computacional seja menor, tornando-a extremamente atraente para cientistas e pesquisadores.

3.2.2 Obtenção dos *Datasets* e Pré-processamento dos dados

Para se criar um classificador de *tweets* depressivos, deve-se fornecer exemplos do que seriam *tweets* depressivos e não-depressivos para a etapa de treinamento. Assim, após o treinamento supervisionado, o modelo estará pronto para distinguir dados das duas classes.

Os *tweets* depressivos obtidos em (ASHWANTHRAMJI, 2017) foram extraídos utilizando a *StreamAPI* do Twitter filtrando pelas palavras "*anxiety*", "*depression*" e "*mental health*". A partir disto, obteve-se 3574 *tweets* depressivos para treinamento.

Os *tweets* não-depressivos foram coletados de um *dataset* muito utilizado chamado *Sentiment140*. Esse *dataset* possui 1,6 milhões de *tweets* divididos em 3 classes: positivos, neutros e negativos. Para haver um balanceamento das classes no aprendizado supervisionado dos dados, precisamos da mesma quantidade de *tweets* não-depressivos. Os *tweets* não-depressivos foram extraídos do *Sentiment140*, reordenados de uma forma aleatória e após isso coletou-se 3574 *tweets* que serão usados para treinamento.

A seguir está a sequência do processo utilizado para obtenção dos dados necessários para treinamento assim como a descrição do processo de treinamento e criação do modelo de classificação. Importa-se a biblioteca *Pandas*, que foi usada para oferecer a estrutura de dados em que os dados são armazenados e fornecer as operações para se manipular as tabelas de uma forma conveniente. Esta biblioteca é muito utilizada na resolução de problemas de análise de dados e aprendizado de máquina (MCKINNEY, 2011).

```
import pandas as pd
PATH = 'C:\\Users\\vinic\\test_directory\\tcc\\
↪  training.1600000.processed.noemoticon.csv'
COLUMNS = ["target", "ids", "date", "flag", "user", "text"]

df = pd.read_csv(PATH, encoding="ISO-8859-1", names=COLUMNS)
```

```
df_copy = df
df_copy = df_copy.drop(['ids', 'date', 'flag', 'user'], axis=1)
```

O processo de manipulação dos dados é inicialmente focado nos *tweets* da classe não-depressiva do *dataset Sentiment140*. No código acima foi utilizada a função *read_csv()* para importar o arquivo CSV contendo os 1,6 milhões de *tweets* do *dataset Sentiment140* e inserí-lo num *DataFrame* do *Pandas* que foi chamado apenas de *df*. O *dataset* original contém as colunas "*target*", "*id*", "*date*", "*flag*", "*user*", "*text*", porém só iremos utilizar as colunas de "*target*"e "*text*". As demais colunas nós eliminamos do *DataFrame* utilizando a função *drop()*.

Como dito anteriormente, o *dataset Sentiment140* possui 3 classes (0 - negativo, 2 - neutro, 4 - positivo), porém, estamos apenas interessados na parte não-depressiva ou positiva deste *dataset*. Para isso iremos filtrar todos os *tweets* que tiverem o valor de *target* valendo 4, classe na qual indica positividade, e coletar 3574 *tweets*. O número 3574 foi usado para garantir o balanceamento das classes durante o treinamento dos dados, pois, a quantidade de *tweets* coletados da classe depressiva é de 3574 *tweets*. Para que o resultado do procedimento de coleta dos 3574 *tweets* seja sempre o mesmo, foi definido um estado randômico como sendo 27.

```
not_depressive_tweets_df = df_copy[df_copy.target == 4]
not_depressive_tweets_df = not_depressive_tweets_df.sample(n=3574,
↪  random_state=27)
```

Foi definido que para os *tweets* não-depressivos, o rótulo é +1, portanto, usa-se a biblioteca *Numpy* para se criar um *array* contendo números "1" para que seja atribuído na coluna "*target*" do *DataFrame* que armazena os *tweets* não-depressivos, assim como mostra o código abaixo.

```
target_class_one = np.ones((not_depressive_tweets_df.shape[0],),
↪  dtype=int)

not_depressive_tweets_df['target'] = target_class_one
```

Se executarmos no *DataFrame* a função *head()*, temos uma visão dos primeiros 5 itens da estrutura que armazena os *tweets* não depressivos apresentado na Figura 8.

```
not_depressive_tweets_df[['target', 'text']].head()
```

Até então, tem-se a parcela do *dataset* que contém *tweets* não-depressivos. O próximo passo agora é manipular os dados dos *tweets* depressivos. Os *tweets* depressivos

	target	text
1399954	1	as of today i am 16! - thatboy: aw thank you ...
1163529	1	@SummerAmes hi, how are you?
1204131	1	bonjour! going to class. im going to do workou...
1581722	1	@riiaa No way! I would rather shoot them and f...
986704	1	@kennethjor yeahhhh and how many would you nee...

Figura 8 – Representação dos 5 últimos *tweets* não-depressivos e seus rótulos.

obtidos a partir da *StreamAPI* do Twitter não vêm no formato correto para se inserir em um *DataFrame* , portanto, transforma-se a *stream* obtida em uma lista de objetos do tipo *json* para que se possa utilizar o método *read_json()* da biblioteca Pandas e assim criar um *DataFrame* a partir dos dados.

```
file_to_open =
↪  Path("C:/Users/vinic/test_directory/tcc/Depression-Sentiment-Analysis
↪  -Twitter-Data-master", "tweetdata.txt")
with open(file_to_open) as datafile:
    content = datafile.read()
depressive_tweets_df = pd.read_json('[' + content.replace('}\n', '},') +
↪  ']')
```

Foi definido que para os *tweets* depressivos, o rótulo é -1, portanto, cria-se uma *Series* ou coluna com o rótulo -1 para todos os *tweets* depressivos, como mostra o código abaixo.

```
target_class_minus_one = pd.Series(np.ones(depressive_tweets_df.shape[0],
↪  dtype=int)) * -1
depressive_tweets_df['target'] = target_class_minus_one
```

Se executarmos no *DataFrame* a função *tail()*, temos uma visão dos últimos 5 itens da estrutura que armazena os *tweets* depressivos apresentado na Figura 9.

```
depressive_tweets_df[['target', 'text']].tail()
```

Com os *tweets* depressivos e não-depressivos já com seus rótulos e um em cada *DataFrame*, agora precisamos concatenar os *DataFrames* e formar um só.

	target	text
3569	-1	I never get this much anxiety until I'm home :/
3570	-1	Hospital staff warn of more 'adverse outcomes'...
3571	-1	RT @Jniewalker: มีนย้อกถามซอสยอนว่า depression...
3572	-1	RT @philuhmena: 7 months ago today I was raped...
3573	-1	my anxiety is thru the fucking roof.

Figura 9 – Representação dos 5 últimos *tweets* depressivos e seus rótulos.

```
all_tweets = pd.concat([not_depressive_tweets_df, depressive_tweets_df],
↪  sort=False)
```

O código acima concatena os *DataFrames*, porém, deve-se ainda utilizar a função sample() para criar uma nova tabela em que os *tweets* depressivos e não-depressivos estão reordenados de uma forma aleatória e reconfigurar o índice da tabela para ir de 0 a 7147, totalizando 7148 *tweets*. A Figura 10 mostra a forma como os dados estão arranjados após serem concatenados e reordenados.

```
all_tweets = all_tweets.sample(frac=1).reset_index(drop=True)
```

	target	text
7138	1	@jasonridge1 ooh where did you go eat? what ar...
7139	1	i love love my mum, she got us row 7 for BGT t...
7140	-1	Hmm pie charts might work better for me 👌 htt...
7141	1	Big day ahead of us! Vocal coaching at 11 ! Th...
7142	1	i guess thats what they meant by work smart.no...
7143	1	Oh wow Sorry about that Shonta, Morning Ms. B ...
7144	-1	RT @Salon: His son's death spurs "No One Cares...
7145	-1	Sometimes I block people ou because I'm thinki...
7146	-1	RT @CroydonMH: Does this help with Youth Menta...
7147	-1	@boodlebrain You're a wonderful grandma punk! ...

Figura 10 – Representação dos 10 últimos *tweets* depressivos e não-depressivos concatenados e seus rótulos

O próximo passo é criar uma função normalizadora que aplica para cada *tweet* uma serie de pré-processamentos. A função se chama *normalizer()* e recebe como argumento um *tweet* e é definida como:

```
1  import string
2  from nltk.corpus import stopwords
3  import re
4
5  def normalizer(tweet):
6      punct = list(string.punctuation)
7      stop_words = stopwords.words('english')
8      additional_stop_words = ['RT', 'rt', 'via', '...', 'http', 'twitpic',
       ↪  'tinyurl' ,'www']
9      stopword_list = punct + stop_words + additional_stop_words
10
11     tweet = re.sub("(@[A-Za-z0-9]+)|(#[A-Za-z0-9]+)", " ", tweet)
12     tweet_ = re.sub("(\w+:\/\/\S+)", " ", tweet)
13     tweet__ = re.sub("[^a-zA-Z]", " ", tweet_)
14     lemmatized = lemmatize_with_postag(tweet__)
15     tokens = nltk.word_tokenize(lemmatized)[2:]
16     lower_case = [l.lower() for l in tokens]
17     filtered_result = list(filter(lambda l: l not in stopword_list,
       ↪  lower_case))
18     return filtered_result
```

Como se pode ver no código anterior, da linha 6 até à linha 9 de código definiu-se quais serão os elementos da lista de palavras vazias (*stop words*). A lista é composta por: a) todas as pontuações existentes no padrão ASCII; b) as palavras vazias da língua inglesa que a biblioteca *NLTK* fornece; e c) uma lista de strings de palavras que aparecem comumente em diversos *tweets* e que não possuem relevância para o modelo a ser criado.

Na linha 11, retira-se todas as *hashtags* e menções de usuários. Na linha 12, elimina-se qualquer endereço de URL. Na linha 13, elimina-se todas palavras que não começam com letras de A até Z independentemente se são minúsculas ou não. Na linha 14, aplicamos a função de lematização. Na linha 15, transforma-se o *tweet* em *tokens*. Na linha 16 coloca-se todos os *tokens* em caixa-baixa. Na linha 17, filtra-se todos os *tokens* que não estão na lista de palavras vazias e na linha seguinte se retorna todo o resultado filtrado.

A função de lematização anterior é definida como:

```
from textblob import TextBlob
```

```
def lemmatize_with_postag(sentence):
    sent = TextBlob(sentence)
    tag_dict = {"J": 'a',
                "N": 'n',
                "V": 'v',
                "R": 'r'}
    words_and_tags = [(w, tag_dict.get(pos[0], 'n')) for w, pos in
    ↪  sent.tags]
    lemmatized_list = [wd.lemmatize(tag) for wd, tag in words_and_tags]
    return " ".join(lemmatized_list)
```

A biblioteca importada *TextBlob* recebe uma sentença e identifica a categoria gramatical (*part-of-speech tag*) para cada palavra da sentença que lhe foi passada. Para simplificar a forma como a função *lemmatize* do *TextBlob* trabalha deve-se apenas considerar adjetivos, substantivos, verbos e advérbios. A partir da lista de tuplas que é criada, onde contém cada palavra e sua classe gramatical correspondente, para cada palavra usa-se o método *lemmatize()* utilizando como argumento a classe gramatical ou *tag* da palavra que esta sendo lematizada. Se a frase de entrada na função de lematização fosse: *"The striped bats are hanging on their feet for best"*, a saída seria: *"The striped bat be hang on their foot for best"*

Após definir as funções de normalização, agora aplicamos ela no *DataFrame* que contém todos os *tweets* .

```
all_tweets['normalized_tweet'] = all_tweets.text.apply(normalizer)
```

Agora que temos todos os *tweets* normalizados iremos adaptá-los para inserir no algoritmo de treinamento. Para isso, utilizamos a função *CountVectorizer*.

```
from scipy.sparse import hstack
from sklearn.feature_extraction.text import CountVectorizer

count_vectorizer = CountVectorizer(ngram_range=(1,2))

vectorized_data =
↪  count_vectorizer.fit_transform(all_tweets['normalized_tweet'])

indexed_data =
↪  hstack((np.array(range(0,vectorized_data.shape[0]))[:,None],
↪  vectorized_data))
```

No código acima foi instanciado um objeto do *CountVectorizer* e foi inserido os *tweets* normalizados para que eles sejam vetorizados. Como visto anteriormente, existiram milhares de valores de 0 devido ao vocabulário ser muito grande, portanto, é necessário criar uma matriz esparsa utilizando a função *hstack*. Após isso, temos na variável *indexed_data* todos os *tweets* vetorizados e prontos para serem treinados juntamente com seus rótulos.

3.2.3 Treinamento dos dados

Treina-se os dados utilizando a função svm.SVC(), função que simula o SVM dentro da biblioteca *Scikit-learn*. Inicialmente, precisa-se separar parte dos dados para que possamos testar o modelo criado e validá-lo. Como um teste inicial, foram separados 30% dos dados para validação do modelo e os 70% restantes para treinamento.

```
from sklearn.model_selection import train_test_split
data_train, data_test, targets_train, targets_test =
↪  train_test_split(indexed_data, targets, test_size=0.3,
↪  random_state=0)
data_train_index = data_train[:,0]
data_train = data_train[:,1:]
data_test_index = data_test[:,0]
data_test = data_test[:,1:]
```

O modelo SVM foi inicialmente criado com os valores *default* de $C = 1$, e *kernel* linear. A biblioteca também possui suporte para os *kernels* pré-definidos como: RBF(*Radial Basis Function* - Função de Base Radial) e Polinomial, além de permitir a criação de um *kernel* próprio.

```
from sklearn import svm

clf = svm.SVC(C= 1, kernel='linear')
clf.fit(data_train, targets_train)
```

3.2.4 Validação do modelo

Para validar o modelo SVM criado, utiliza-se a função *score*() com os dados separados para teste. O método *score* irá avaliar se o modelo consegue identificar todos os rótulos de todos os *tweets* que estão em *data_test*.

```
clf.score(data_test, targets_test)
```

O resultado para o *score* do classificador criado foi de 97%.

3.2.5 Otimização de parâmetros do modelo

Como explicado anteriormente, o *grid-search* é utilizado neste momento para obtenção dos melhores parâmetros do modelo svm.SCV() criado anteriormente, pois, naquele momento o modelo foi criado utilizando *C*, *gamma* e *kernel* padrões para o tipo de modelo.

```
from sklearn.model_selection import GridSearchCV
hiperparameters = {
                    'kernel': ('linear', 'rbf', 'poly'),
                    'C':[1.5, 10, 100, 1000],
                    'gamma': [1e-7, 1e-4, 1e-2, 0.1]
            }
grid = GridSearchCV(svm.SVC(), hiperparameters, cv=5)
grid.fit(data_train, targets_train)

print(grid.best_params_)
print(grid.best_score_)
```

A saída para este código e o *score* do modelo foram:

Saída: 'C': 1.5, 'gamma': 0.01, 'kernel': 'rbf' *Score*: 0.9974015590645613

Importa-se o método *GridSearchCV()* da biblioteca *Scikit-learn* e injeta-se o modelo - que no caso é o Classificador SVM - no parâmetro *estimator*, um vetor de cada parâmetro que pode ser variado. Após treinarmos o *Grid-search* de acordo com os dados de entrada e saída, teremos o atributo *best_score_* e *best_params_*, contendo o modelo que mais pontuou na etapa de teste, ou seja, classificou de forma correta exemplos desconhecidos para seu modelo e a melhor combinação dos parâmetros que geraram esse modelo.

3.2.6 Exemplo de classificação

Cria-se um exemplo de *tweet* nunca antes visto e que não fez parte da etapa de treinamento para que possamos classificá-lo e determinar se o resultado foi correto de acordo com o que imaginávamos.

```
tweet_to_predict = ["I think I have depression. I'll search for help."]
x = count_vectorizer.transform(tweet_to_predict)
clf.predict(x)
```

A saída da predição deste *tweet* foi de -1, ou seja, o *tweet* foi considerado pelo modelo como pertencente à classe depressiva.

3.3 Resultados e Discussões

O programa implementado foi executado sobre o conjunto de dados citado na Seção 3.2.2 e criou-se um modelo de classificação SVM capaz de receber entradas de *tweets* e classificá-los como depressivos ou não-depressivos. A frase "*I think I have depression. I'll search for help.*" foi testada no modelo e resultou em -1, ou seja, indicando um *tweet* depressivo de forma correta. A frase "*I am very happy today.*", resultou em 1, indicando um *tweet* não-depressivo, também corretamente. Entretanto, ao avaliar o *tweet* "*I want to die.*", o resultado foi 1, indicando um *tweet* não-depressivo. Esse comportamento aconteceu porque essas palavras não estavam presentes nos exemplos usados no treinamento e, portanto, ficaram fora do modelo criado. Muitos dos *tweets* depressivos dos usuários podem conter um conteúdo considerado depressivo mas não explicitamente citar palavras como morte, portanto, é necessário uma quantidade maior de *tweets* depressivos para que se tenha um modelo capaz de generalizar grande parte dos possíveis *tweets* que venham a ser avaliados.

4 Conclusão

Conclui-se que o objetivo de criar um modelo de classificação de *tweets* depressivos utilizando SVM foi realizado com sucesso e obteve-se uma taxa de acerto de 99.7% com os dados de treinamento otimizados pelo método de Grade de Busca. Para as amostras utilizadas, os melhores parâmetros encontrados para o classificador foram o valor de penalidade $C = 1.5$ e o *kernel* RBF, com $\gamma = 0.01$.

O uso do ambiente *Jupyter Notebook* foi um grande facilitador no desenvolvimento deste trabalho e seu uso é bastante recomendado na construção de protótipos, pois é possível ver o resultado de comandos individuais assim que são adicionados. A linguagem Python mostrou-se bastante satisfatória para testar o modelo, pois as bibliotecas disponíveis fornecem recursos suficientes para todas as etapas da metodologia adotada.

A presença de erros de classificação quando o modelo foi testado com dados não provenientes de *tweets* reais pode ser explicada pela ausência dos termos utilizados na base de treinamento e, como trabalhos futuros, podem ser realizadas melhorias com o intuito de aumentar essa taxa de acerto. A primeira sugestão seria ampliar o conjunto de dados de treinamento. Existem diversas bases de dados com milhares de *tweets* disponíveis, e esse aumento pode aumentar a eficácia do modelo. Outra possibilidade seria melhorar o pré-processamento. No inglês, assim como no português, é comum se utilizar simplificações de palavras que são usadas constantemente e que são de fácil entendimento. Caso haja alguma forma no futuro de normalizar todas estas simplificações, o modelo de classificação conseguiria se tornar mais performático. Também seria possível usar outro modelo de características, como trigramas ou n-gramas, ou ainda redes semânticas para representar os textos dos *tweets*.

Por último, sugere-se que o modelo seja usado em uma aplicação de análise de texto, seja do Twitter, outra rede social ou outro aplicativo, para a identificação de comportamento indicativo de depressão.

Referências

ASHWANTHRAMJI. *Depression-Sentiment-Analysis-with-Twitter-Data*. 2017. Disponível em: <https://github.com/AshwanthRamji/Depression-Sentiment-Analysis-with-Twitter-Data>. Acesso em: 04 jun. 2019. Citado na página 27.

ASLAM, S. *Twitter by the Numbers: Stats, Demographics Fun Facts*. 2018. Disponível em: <https://www.omnicoreagency.com/twitter-statistics/>. Acesso em: 04 jun. 2019. Citado na página 8.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit*. O'Reilly, 2009. Disponível em: <http://www.nltk.org/book_1ed/>. Citado na página 27.

FOLHA, S. P. *Twitter registra crescimento em usuários diários e nas receitas*. 2019. Disponível em: <https://www1.folha.uol.com.br/mercado/2019/04/alteracoes-na-plataforma-fazem-twitter-melhorar-resultados.shtml>. Acesso em: 04 jun. 2019. Citado na página 8.

HEALEY, C. G.; RAMASWAMY, S. S. *Visualizing Twitter Sentiment*. 2011. Disponível em: <https://www.csc2.ncsu.edu/faculty/healey/tweet_viz/>. Acesso em: 04 jun. 2019. Citado na página 8.

JUPYTER, P. *Jupyter*. 2019. Disponível em: <https://jupyter.org/>. Acesso em: 04 jun. 2019. Citado na página 26.

KAPKO, M. *Twitter's impact on 2016 presidential election is unmistakable*. 2016. Disponível em: <https://www.cio.com/article/3137513/twitters-impact-on-2016-presidential-election-is-unmistakable.html>. Acesso em: 04 jun. 2019. Citado na página 8.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008. ISBN 978-0-521-86571-5. Disponível em: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>. Citado na página 12.

MCKINNEY, W. pandas: a foundational python library for data analysis and statistics. *Python High Performance Science Computer*, 01 2011. Citado na página 27.

MYERS, J. *Which languages are most widely spoken?* 2015. Disponível em: <https://www.weforum.org/agenda/2015/10/which-languages-are-most-widely-spoken/>. Acesso em: 04 jun. 2019. Citado na página 9.

NEGRI, R. G. *Máquinas de vetores suporte adaptativa ao contexto: formalização e aplicações em sensoriamento remoto*. Tese (Doutorado) — INPE, São José dos Campos, 2013. Disponível em: <http://mtc-m16d.sid.inpe.br/rep/sid.inpe.br/mtc-m19/2013/07.10.14.32?metadatarepository=sid.inpe.br/mtc-m19/2013/07.10.14.32.48&ibiurl.backgroundlanguage=pt&ibiurl.requiredsite=mtc-m16d.sid.inpe.br+806&requiredmirror=sid.inpe.br/mtc-m19@80/2009/08.21.17.02.53&searchsite=bibdigital.

sid.inpe.br:80&searchmirror=sid.inpe.br/bibdigital@80/2006/11.11.23.17&choice=briefTitleAuthorMisc>. Acesso em: 4 jun. 2019. Citado 2 vezes nas páginas 20 e 21.

NUMFOCUS. *Python Data Analysis Library*. 2019. Disponível em: <https://pandas.pydata.org/>. Acesso em: 04 jun. 2019. Citado na página 26.

PARELLADA, A. *How does a Support Vector Machine (SVM) work*. 2017. Disponível em: <https://stats.stackexchange.com/questions/23391/how-does-a-support-vector-machine-svm-work>. Acesso em: 04 jun. 2019. Citado 2 vezes nas páginas 15 e 16.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 27.

RAMASWAMY, S. S. *Visualization of the Sentiment of the Tweets*. 2011. Disponível em: <https://repository.lib.ncsu.edu/bitstream/handle/1840.16/7151/etd.pdf?sequence=2>. Acesso em: 04 jun. 2019. Citado na página 9.

REFAEILZADEH, P.; TANG, L.; LIU, H. Cross-validation. In: _____. *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009. p. 532–538. ISBN 978-0-387-39940-9. Disponível em: <https://doi.org/10.1007/978-0-387-39940-9_565>. Citado na página 23.

SCIPY. *NumPy*. 2019. Disponível em: <https://www.numpy.org/>. Acesso em: 04 jun. 2019. Citado na página 26.

SRINIDHI, S. *Different types of Validations in Machine Learning (Cross Validation)*. 2018. Disponível em: <https://blog.contactsunny.com/data-science/different-types-of-validations-in-machine-learning-cross-validation>. Acesso em: 04 jun. 2019. Citado na página 23.

WEIL, K. *Measuring Tweets*. 2010. Disponível em: <https://blog.twitter.com/en_us/a/2010/measuring-tweets.html>. Acesso em: 04 jun. 2019. Citado na página 8.

Wikipedia contributors. *Hyperplane — Wikipedia, The Free Encyclopedia*. 2019. [Online; accessed 4-June-2019]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Hyperplane&oldid=897671749>. Citado na página 14.