Ana Valeria Vieira da Silva

Mestrado em Análise e Engenharia de Big Data

Detecção de contas anômalas e influência em redes sociais na propaganda política e em eleições

Dissertação de Mestrado em Análise e Engenharia de Big Data

2º Semestre, 2019/2020

Orientador: Ludwig Krippahl, Professor auxiliar, Universidade Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2020

Detecção de contas anômalas e influência em redes sociais na propaganda política e em eleições

"O problema com Eichmann era exatamente que muitos eram como ele, e muitos não eram nem pervertidos, nem sádicos, mas eram e ainda são terrível e assustadoramente normais. Do ponto de vista de nossas instituições e de nossos padrões morais de julgamento, essa normalidade era muito mais apavorante do que todas as atrocidades juntas, pois implicava que – como foi dito insistentemente em Nuremberg pelos acusados e seus advogados – esse era um tipo novo de criminoso, efetivamente hostis generis humani, *que comete seus crimes em circunstâncias que tornam praticamente impossível para ele saber ou sentir que está agindo de modo errado".*

(Hannah Arendt – Eichmann em Jerusalém – Um relato sobre a banalidade do mal)

Dedicado a todos que tentam, de alguma forma, manter o exercício do estado democrático de direito protegido em terras de liberdade de pensamento e existência, sendo nos tempos brasileiros atuais, resistência.

(autoria própria)

Agradecimentos

Agradeço primeiramente ao Professor Ludwig Krippahl pela orientação, confiança e conhecimento transmitido, sem o qual não teria sido possível a realização desta dissertação.

Ao Professor Pedro Manuel Corrêa Calvente Barahona pela disponibilidade, atenção e ajuda nos momentos de tormenta durante o mestrado.

Um obrigada a todos os professores do mestrado em Análise e Engenharia de Big Data da FCT pelos valiosos conhecimentos transmitidos.

Aos colegas do mestrado pelos momentos de descontração e troca de conhecimento: Pedro Cheira, Paulo Santos, Filipe Santos, Sandra Cordeiro, Bruno Miguel e Rafael Peixinho.

Aos meus pais Maria Audeci e Valter Vieira pelo afeto, educação, força, orientação e apoio incondicional. Aos meus irmãos Wagner André, Valber Henrique e Valter Alexandre pelo apoio, incentivo e motivação sempre.

A Joana Paz pelo afeto, apoio, incentivo, companhia e cuidado imensuráveis durante a elaboração deste trabalho.

A todos os meus amigos espalhados pelo mundo, que compreenderam a minha ausência e sempre deram afeto, apoio, força e entusiasmo para concluir esta dissertação.

A minha psicanalista Paula Rego pelo incentivo, dicas e reflexões que ajudaram este trabalho ser mais célere e mais prazeroso.

E agradeço a Deus e a todos os meus orixás pela proteção e por sempre me ajudar a ter foco e determinação.

Muito obrigada a todos!

E não menos importante:

FORA BOLSONARO!

Resumo

As redes sociais podem ser utilizadas em ambiente de propaganda política ou eleições como canal de informação aos seus usuários. A rede social Twitter tem sido utilizada como plataforma política em diversas eleições por todo mundo e essas propagandas politicas são coordenadas por contas com utilizadores humanos ou não humanos.

É fundamental discernir a atividade das contas normais de contas anômalas, pois a informação propagada pela rede pode ter diversos objetivos, incluindo a desinformação. A desinformação causada pelas contas anômalas possui maior probabilidade de alcance na rede social Twitter devido a elevada quantidade de contas que podem estar articuladas com este mesmo objetivo.

Esta dissertação desenvolveu dois modelos de detecção de contas anômalas a partir dos algoritmos SVM *one-class* e Autoencoder Ensembles. O estudo utilizou como dados reais os *tweets* publicados no Twitter durante as eleições ocorridas no Brasil em 2018 e Portugal em 2019.

As *features* da rede social Twitter foram importantes na análise do perfil do utilizador e foram selecionadas utilizando a técnica *Recursive Feature Elimination*.

Foram utilizadas apenas informações de contas normais previamente conhecidas para treinar os modelos de aprendizagem não supervisionada, o que permite reutilizar os modelos em outras eleições para qualquer país, sendo esta uma inovação proposta por este trabalho.

Os dois modelos permitiram detectar as contas anômalas nos dados de teste apresentando globalmente boa performance em seus resultados. O Autoencoder Ensembles foi desenvolvido pelo método de ensembles a partir do valor do erro de reconstrução com treinamento em sequência.

A partir da detecção das contas anômalas foram criados clusters sendo possível verificar que existe diferença de discurso político publicado pelas contas anômalas através de *tweets* nas eleições ocorridas no Brasil e em Portugal.

Os dados reais foram retirados da API desenvolvedora do Twitter, o que permitiu acessar as contas ativas durante o período eleitoral no ano de 2018 no Brasil e 2019 em Portugal.

O objetivo desta dissertação foi contribuir para o crescente estudo no campo dos Sistemas de Informação que utilizam técnicas de *machine learning* e redes neurais para detecção de contas reais ou anômalas, além da visualização dos clusters gerados a partir destas últimas.

Palavras-chave: *anômalas, detecção, machine learning, Autoencoder, rede social.*

Abstract

Social networks can be used in an environment of political campaigning or elections as an information channel for its users. Twitter's social network has been used as a political platform in several elections around the world and its political campaigns are coordinated by accounts with human or non-human users.

It is essential to discern the activity of normal anomalous accounts, as the information propagated by the network can have several objectives, including disinformation. The disinformation caused by anomalous accounts is more likely to reach the social network due to the high number of accounts that can be linked to this same objective.

This dissertation uses two models to detect anomalous accounts based on the SVM one-class and Autoencoder Ensembles algorithms. The study used real data as recently elections occurred in Brazil in 2018 and Portugal in 2019.

The features of Twitter's social network were important in the analysis of the user's profile and were selected using the Recursive Feature Elimination technique.

Only information from previously known normal accounts was used to train unsupervised learning models, which allows the models to be reused in other elections for any country, this being an innovation proposed by this work.

Both models made it possible to detect anomalous accounts in the test data, presenting overall good performance in their results. The Autoencoder Ensembles was developed by the ensembles method from the value of the reconstruction error with training in sequence.

From the detection of anomalous accounts, clusters were created and it is possible to verify that there is a difference in the political discourse published by anomalous accounts through tweets in the elections that took place in Brazil and Portugal.

The actual data was taken from the Twitter developer API, which enabled the access to active accounts during the electoral period in 2018 in Brazil and 2019 in Portugal.

The objective of this dissertation was to contribute to the growing study in the field of Information Systems that use machine learning techniques and neural networks to detect real or anomalous accounts, in addition to visualizing the clusters generated from these latter.

Keywords: anomalous, detection, machine learning, Autoencoder, social network.

Índice

Lista de Figuras

Lista de Tabelas

1 Introdução

A utilização de redes sociais por governos, empresas e usuários é crescente e cada vez mais está relacionada a influência que as informações divulgadas nestes meios exercem sobre seus usuários. Em épocas de propaganda política e eleições, as redes sociais tornam-se ferramentas importantes para divulgar a campanha eleitoral, realizar debates políticos e tentar influenciar o voto dos usuários. No caso da rede social Twitter, a ferramenta tem sido utilizada para promover a abrangência dos candidatos políticos e suas plataformas, estimular o debate sobre os programas dos partidos políticos, podendo influenciar a decisão dos seus usuários no voto. Esta rede social tem atualmente mais de 330 milhões de usuários ativos mensais em 2019 segundo análise da empresa Statista (Statista, 2019).

Na aplicação Twitter é possível existir contas de utilizadores humanos e não humanos. As contas de usuários humanos podem ser contas de comportamento "normal" ou contas falsas. Em relação as contas não humanas estas são contas automatizadas também conhecidas como *bots.* De acordo com pesquisas acadêmicas realizadas nos últimos anos, instituições governamentais e a própria rede social, existem no Twitter vários exemplos do uso de contas falsas e *bots,* seja para fins legítimos ou não legítimos, como por exemplo espalhar comunicação de maneira informativa, divulgar a plataforma de determinados partidos, compartilhar notícias de jornais e até mesmo influenciar as discussões políticas na plataforma social (Araújo, 2011; Arnaudo, 2017; European Union, 2018; Grimme et al., 2017).

Nas últimas eleições presidenciais ocorridas em vários países no mundo, existem diversas evidências que foram utilizados *bots* para divulgar informações sobre os candidatos. Isso pode ser observado através de publicações acadêmicas que analisaram as eleições presidenciais dos EUA em 2016 (Ferrara et al., 2016b; Woolley et al., 2016), na França em 2017 (Ferrara, 2017) e no Brasil em 2018 (Ruediger, 2018). Também foi observado o mesmo comportamento no Irão em 2018 (Thieltges et al., 2018), na comunicação política na Russia (Stukal et al., 2019a), nas eleições para o Parlamento na Alemanha em 2017 (Brachten et al., 2017), nas eleições gerais do Japão em 2014 (Schäfer et al., 2017).

Os perfis automatizados no Twitter motivaram até 20% do debate político-eleitoral no Brasil durante as eleições presidenciais de 2014, manifestação pelo impeachment de Dilma Roussef em 2016 e as greves gerais ocorridas em 2017 (Ruediger et al., 2018). Howard et al. (2016) através da

análise de influência das contas automatizadas sobre o referendo Brexit concluiu que um terço dos *tweets* publicados sobre o Brexit foi gerado por apenas 1% das contas do Twitter. Isso demonstra a potencialidade na propagação de notícias na rede social Twitter.

Esta potencialidade durante propaganda política e eleições também é possível ser observada em contas de utilizadores humanos na rede social Twitter (Cotrim, 2018). Existem contas de utilizadores humanos e contas falsas manipuladas por usuários humanos com o intuito de realizar propaganda política, aumentar a participação das pessoas, ganhos eleitorais e melhorar o desempenho eleitoral, tendo como principal foco influenciar a decisão do voto dos utilizadores, podendo gerar desinformação (Rahmawati, 2014; Muntean, 2015; Suuronen, 2018; Nielsen et al., 2019; Martín, 2019).

Em nível internacional, existem diversas publicações de artigos acadêmicos sobre a utilização de *bots* durante as campanhas eleitorais e as eleições em vários países no mundo. Podemos destacar o trabalho de Sippo Rossi (2019) *Detecting and analyzing bots on Finnish political twitter*, tese de mestrado onde o autor desenvolveu um modelo de *machine learning* para detectar contas automatizadas e verificar o comportamento dos *bots* na eleição para o Parlamento Finlandês em 2019. Outro contributo foi realizado por Josemar Alves Caetano (2018) na sua tese de mestrado C*haracterizing Politically engaged users during the 2016 US presidential campaign using twitter*, onde foi desenvolvido um modelo para identificação de tipos de *bots* ou usuários humanos no Twitter aplicado na campanha Presidencial dos EUA em 2016 utilizando técnicas de Análise de Sentimento através da ferramenta *SentiStrength*, ainda utilizou a ferramenta *Botometer* e *Data Mining* (*K-Means*).

Desenvolver um método para identificar contas automatizadas é algo considerado desafiador na análise de redes sociais e existe uma variedade de pesquisas neste sentido (Ferrara et al., 2016b; Davis et al., 2016; Stukal et al., 2017; Varol et al., 2017; Efthimion et al., 2018; Walt et al., 2018, Stukal et al., 2019b; Yang et al., 2020). Os métodos de classificação observados nas pesquisas citadas acima consideram na sua maioria que os modelos de aprendizagem de máquina supervisionado funcionam melhor que o não supervisionado, apesar da necessidade de existir dados rotulados para o treinamento do modelo supervisionado e estes representarem apenas uma parte das possíveis características de comportamento de contas automatizadas limitando o resultado do modelo.

É possível encontrar detectores on-line como o "*Bot*ometer" (Davis et al., 2016), que utiliza um modelo de aprendizado supervisionado (*Random Forest*) e fornece a API em Python para utilização. Outro modelo disponível on-line é o "*tweetbot*ornot" desenvolvido por Michael W. Kearney

(Kearney, 2019) que consiste em um pacote R desenvolvido com modelo de *machine learning.* Ainda é possível encontrar on-line o detector "De*bot*" (Chavoshi et al., 2017) que utiliza aprendizado de máquina não-supervisionado, não sendo necessário parâmetros definidos. O detector de *bots* "*Bot* Sentinel" (2018) utiliza *machine learning* para classificar contas como confiável e não confiável e identificar *bots*, é direcionado para contas em inglês. Outro detector de *bots*, o "Pega*bot*" (Instituto de Tecnologia e Sociedade do Rio de Janeiro, 2018) usa javascript e utiliza características de algumas *features* para inferir se a conta é *bot* ou não.

A limitação destas ferramentas é a taxa de consulta ser restrita a pouco mais de mil contas, não permitir visualizar os detalhes das contas e não exibir as contas que foram suspensas, protegidas, excluídas ou estão em quarentena. Estas últimas são muito importantes para o modelo a ser desenvolvido neste trabalho, pois muitas contas são criadas próximas ao período das eleições e em seguida são desativadas de alguma forma (Ferrara, 2017), o que poderia mascarar o resultado de qualquer classificador.

Para este trabalho, é importante considerar que contas anômalas podem ser contas falsas manipuladas por usuários humanos ou *bots*. A definição de um *bot* é algo variável na literatura e não há uma nomenclatura padrão para essa definição e tipologias. Basicamente, um *bot* é um algoritmo programado para operar como uma conta de usuário de forma completamente automatizada. Os tipo de *bots* também não são padronizados, mas de forma geral na literatura encontrada (Abokhodair et al., 2015; Efthimion et al., 2018; Gorwa et al., 2018; Rossi, 2019) podem ser divididos em *Web Robots* (*Crawlers* e *Scrapers*), *Chatbots, Spambots, Social bots, Sockpuppets e Trolls, Cyborgs e Hybrid Accounts.* Os *bots* podem executar desde tarefas simples como espalhar conteúdos baseados em scripts, aumentar número de seguidores ou *likes* em uma conta, como também podem ser complexos em que ajustam o comportamento de uma conta baseado em outras contas, estes últimos são de difícil detecção pois aprenderam a imitar o comportamento humano de forma quase imperceptível (Grimme et al., 2017).

Os Twitter *bots* são *bots* da internet que operam a partir de uma conta do Twitter, onde as principais tarefas que um Twitter *bot* executa são publicar *tweets*, *retweets*, *likes*, podendo estes ser divididos em dois tipos: benignos e maliciosos. Nos termos de serviço do Twitter (Twitter, 2019d) é permitida a utilização de *bots* desde que essas contas não executem ações de *tweet*ing em massa de forma automática, aumente artificialmente o número de seguidores, direcione e manipule discussões ou que possuam *links* enganosos em suas mensagens. Essas definições tem grande importância na classificação de contas de usuários "normais" e contas suspeitas, pois permite diferenciar o propósito de cada conta.

Na rede social Twitter para que seja possível identificar os diferentes tipos de contas é fornecida uma API (*Application Programming Interface*) *development(search)* que permite a coleta de milhões de dados em formato de JSON. Este arquivo JSON (Twitter, 2019b) retorna informações históricas e atuais dos *tweets* (texto, data e hora do *tweet*, idioma, links) assim como os metadados relacionados (características da conta que enviou o *tweet*, incluíndo a ID do autor e o apelido na rede, número de seguidores e amigos, etc).

O arquivo JSON mencionado acima contém um objeto pai que pode conter além de outros objetos filho (*User, Entities, Extended Entities, Places*), mais de 150 *features.* Estas *features* podem ser divididas em textual, imagens e vídeos, metadados relacionados e *network* (Wijeratne et al., 2017). Análises de *features* feitas em outros trabalhos (Davis, et al., 2016; Ferrara, 2017; Varol, et al., 2017; Yang, et al., 2020) demonstraram que as classes de *features* mais determinantes para detecção das diversas contas são os metadados com as *features* de atividades associadas a cada conta de usuário.

Uma definição importante para o entendimento deste trabalho é o significado de um *tweet*. Um *tweet* pode ser um texto curto (máximo de 280 caracteres), fotos ou vídeos curtos que são publicados no Twitter, ou seja, é o ato de enviar um *tweet* e são mostrados na linha do tempo do Twitter ou incorporados em sites e blogs (Twitter, 2019a).

A presente dissertação teve como objetivo desenvolver um modelo capaz de detectar contas anômalas na rede social Twitter durante a última eleição presidencial ocorrida no Brasil e as eleições nas assembleias legislativas de Portugal ocorridas em 2019.

O trabalho realizado está dividido em três etapas distintas. A primeira etapa consistiu em selecionar as principais *features* a serem utilizadas nos detectores de anomalias a partir da utilização da técnica *Recursive Feature Elimination* (RFE).

A utilização da RFE automatizou o processo de escolha das principais *features*, ao contrário dos trabalhos existentes na literatura. De partida algumas *features* foram consideradas importantes como por exemplo a feature ‘favorites_count’. A maior parte das contas de usuários “normais” possuem muitas contas favoritadas, o que permite detectar que determinada conta de usuário que tenha muitas contas favoritas tem uma grande chance de ser uma conta de um utilizador de comportamento normal (Yang et al., 2020).

A segunda etapa foi realizada através da utilização dos algoritmos SVM *one-class* e Autoencoder Ensembles utilizando as *features* selecionadas na primeira etapa para os dados de

treinamento, validação e teste. Esta etapa teve como objetivo criar um detector de anomalias em alternativa aos classificadores já existentes para identificar os usuários no Twitter.

Os modelos utilizando aprendizagem não supervisionada foram treinados apenas com as informações das contas normais para que na utilização dos dados de teste as contas anômalas fossem identificadas como ruídos ou *outliers*, sendo consideradas como uma anomalia. Esta consideração se pauta no fundamento de que ao utilizar uma base de dados para treinamento, onde as contas de usuários reais já estão assinaladas, o perfil de comportamento dos usuários "normais" não tende a ter grandes mudanças, ao contrário das contas falsas ou de *bots*, que tendem a ser modificadas com o intuito de enganar os classificadores já treinados (Schafer et al., 2017). Esta abordagem permite ser ajustada e reutilizada em outras eleições para qualquer país, sendo esta uma inovação proposta por este trabalho.

Por fim, a última etapa deste trabalho consistiu em gerar uma clusterização dos *tweets* para permitir analisar os grupos de *tweets* das contas classificadas como anômalas. Para isso o conteúdo dos *tweets* publicados por estas contas foram representados em vetores contínuos de tamanho fixo utilizando Word Embedding (Word2Vec) e em seguida criados clusters utilizando a rede neural não supervisionada Autoencoder para redução de dimensionalidade.

Para os dados de treinamento e validação foram utilizados dois conjuntos de dados rotulados (cresci-2017 e verified-2019). Para os dados de testes foi realizada a coleta dos dados utilizando a API desenvolvedora do Twitter, onde foram coletados todos os *tweets* publicados referentes ao período das últimas eleições ocorridas no Brasil e em Portugal, o que permitiu uma análise dos diversos detalhes das contas, além de exibir as contas que foram suspensas, protegidas, excluídas ou estão em quarentena.

Para o detector de anomalias as métricas de avaliação utilizadas foram Precision, Recall, F1-Score e matriz de confusão sobre os dados de validação para verificar a performance dos modelos.

O resultado final deste trabalho pretendeu criar um detector de anomalias a partir de aprendizado não supervisionado para permitir a identificação de contas anômalas e criar uma visualização dos clusters destas contas, as quais foram identificadas pelo detector de anomalias, com o propósito de verificar a existência de tipos diferentes de *tweets* publicados por estas contas anômalas.

Este trabalho foi escrito utilizando o acordo ortográfico atual do português do Brasil.

2 Estado da Arte

Neste capítulo é apresentada a revisão bibliográfica feita até o momento em trabalhos similares considerando modelos que melhor performaram em outros trabalhos, as fontes de dados e a análise crítica dos algoritmos a serem utilizados neste trabalho.

2.1 Melhores Trabalhos

Os trabalhos apresentados a seguir são considerados os mais importantes até o momento presente sobre a classificação de usuários reais, falsos ou *bots* na rede social Twitter. Estes em sua maioria utilizaram bases de treinamento rotuladas e disponíveis para o treinamento de modelos de *machine learning* supervisionado.

2.1.1 Trabalhos *on-line*

O detector on-line "*Bot*ometer" (Davis et al., 2016) utiliza para o modelo de aprendizado supervisionado a técnica *Random Forest* com precisão maior que 95% nos dados de treinamento e fornece a API em Python para utilização. Este modelo possibilita analisar mais de 1000 *features* e inclui análise de sentimento. Para *dataset* de treinamento usa atualmente a base de dados varol-2017 e no final da classificação apresenta uma pontuação que diz se a conta é um *bot* ou um usuário humano.

Outro modelo disponível on-line é o "*tweetbot*ornot" desenvolvido por Michael W. Kearney (Kearney, 2019) que consiste em um pacote R desenvolvido com modelo de *machine learning*, mas que na revisão da literatura não foram encontrados maiores detalhes sobre essa aplicação.

Ainda é possível encontrar on-line o detector "De*bot*" (Chavoshi et al., 2017) que utiliza aprendizado de máquina não-supervisionado, não sendo necessário parâmetros definidos. Utiliza a técnica de correlação de atividade para detectar *bots*, filtrando contas que possuem comportamento parecidos.

O detector de *bots Bot* Sentinel utiliza *machine learning* para classificar contas como confiável e não confiável e identificar *bots*. A classificação das contas não confiáveis é feita manualmente e está direcionada para contas que possuem o inglês como linguagem definida, tendo sido criada em 2018. Não foram encontrados maiores detalhes sobre as técnicas de *machine learning* utilizada.

Outro detector de *bots*, o Pega*bot* desenvolvido pelo Instituto de Tecnologia e Sociedade do Rio de Janeiro, lançado em 2018 e em fase de testes, é um software livre escrito em javascript que utiliza cálculos de comprimento de texto e proporção de algumas *features* para inferir se a conta é *bot* ou não.

As ferramentas citadas acima possuem como limitação a taxa de consulta com restrição a pouco mais de mil contas no máximo, não permitir visualizar os detalhes das contas, não exibir as contas que foram suspensas, protegidas, excluídas ou estão em quarentena.

2.1.2 Trabalhos em pesquisas acadêmicas

Ferrara (2017) para identificar a desinformação e os *social bots* nas eleições Presidenciais Francesas em 2017, utilizou uma combinação de *machine learning* com *cognitive behavioral modeling*. Criou um conjunto de palavras-chave através das *hashtags*, utilizou as mesmas *features* de seu trabalho anterior (Ferrara et al., 2016), usou a API *Search* do Twitter coletando 2 milhões de contas de usuários com 17 milhões de *tweets* e os *datasets* utilizados para o treinamento foram cresci-2017 e varol-2017. O autor usou vários métodos de aprendizado de máquina para escolher o que obteve o melhor resultado, sendo este a regressão logística que obteve 92% de acurácia na fase de treinamento e no modelo utilizou 10-*fold cross-validation*. Descobriu que 18% das contas analisadas pertenciam a *bots*.

No trabalho sobre identificação de *bots* no Twitter, Varol (Varol et al., 2017) utilizou 1150 *features* divididas em seis classes, coletou 2.6 milhões de *tweets* de *bots* e 3 milhões de *tweets* de usuários humanos, utilizou o *dataset* caverlee-2011 e enriqueceu o conjunto de treinamento com rótulo manual de 3000 contas que incluem *bots* e usuários humanos. Para a escolha do melhor modelo, fez treinamento de quatro modelos (*Random Forest, AdaBoost, Regressão Logística e Decision Tree*) juntamento com 5-*fold cross-validation*. O modelo escolhido foi *Random Forest* com 95% de acurácia na fase de treinamento. Concluiu que as contas de *bots* no Twitter estão numa faixa de 9% a 15% do total de contas do aplicativo.

Outros trabalhos utilizaram a mesma metodologia na escolha da melhor técnica para a identificação de *bots*, Walt (Walt et al.,2018), Fernquist (Fernquist et al., 2018) e Rossi (2019) utilizaram vários métodos como *Random Forest, AdaBoost,* Regressão Logística, *Naive-Bayes e Support-vector Machine* (SVM) e o modelo que melhor teve resultados foi o *Random Forest* nos três casos.

Em sua tese de mestrado para identificar os Twitter *bots* e verificar sua influência na eleição para o Parlamento Finlandês em 2019, Rossi (2019) utilizou a base de dados cresci-2017 para o treinamento do modelo e 11 *features* divididas em quatro grupos, mas como o modelo teve dificuldade em separar os *bots* dos usuários humanos, o autor criou um *dataset* manual com 2000 contas das duas classes e aumentou o número de *features* para 14, usou 10-*fold cross-validation*, obtendo uma acurácia de 83% no seu conjunto de validação.

Fernquist avaliou a importância das *features*, onde cita as 10 mais importantes *features* para o seu modelo como proporção de *likes* por amigos, proporção de seguidores e seguidos, tempo máximo entre re*tweets* e conclui que os *bots* re*tweet*am um pouco menos que humanos e ao mesmo tempo incluem mais URLs externos em seus *tweets* em comparação com contas de usuários humanos. Este trabalho tambem utilizou conjunto de dados para treinamento com rótulo manual e obteve boa acurácia em seus modelos.

Walt realizou uma pesquisa para detectar contas falsas criadas por usuários humanos. Utilizou os modelos *Random Forest, AdaBoost e Support Vector Machine* e concluiu que as *features* usadas para detectar *bots* e os modelos de *machine learning* utilizados não podem ser usados para detectar contas falsas criada por seres humanos. Isso pode ser atribuído ao fato que os seres humanos têm diferentes características e comportamentos do que os *bots*, que não podem ser modelados de maneira semelhante.

A pesquisa feita por Stukal (Stukal et al., 2019b) para detectar a orientação política dos Twitter *bots* na Russia entre 2015 e 2017 aplicou um modelo de deep learning. O autor analisou o texto contido nos *tweets* dos *bots* e construiu uma rede neural *feedforward* MLP *(Multilayer perceptron*), dividindo o conjunto de treinamento, validação e teste em 80%, 10% e 10% respectivamente. Nas funções de ativação das camadas escondidas utilizou transformação não linear através da função ReLU (*Rectified linear unit*) para acelerar a fase de treinamento e diminuir o problema de *vanishing gradient*, que é a não atualização dos pesos da rede. Com este modelo o autor conseguiu uma acurácia superior a 90%, identificando os *political bots*.

O trabalho mais recente publicado é de Yang (Yang et al., 2020) que desenvolveu um framework para detectar *bots* em *real time* presentes no Twitter, para isso empregou *Random Forest* como método construindo vários classificadores. Como dataset de treinamento foi utilizado um subconjunto de um conjunto de vários conjuntos de dados rotulados manualmente. Como o objetivo do trabalho foi garantir escalabilidade e generalização do framework, apenas as *features* relacionadas às informações de perfil dos usuários foram utilizadas. Conseguiu com esta

metodologia atingir valores de acurácia desde 58% até 99% nos melhores modelos escolhidos, dependendo do dataset utilizado para os testes.

A Tabela 1 apresenta um resumo dos principais modelos de *machine learning* para classificação de utilizadores no Twitter, bem como a configuração e os resultados encontrados nos testes. É possível observar que os modelos possuem diferença quanto a base de dados utilizada para a fase de treinamento e o número de *features.* O algoritmo com melhor resultado observado dentre as pesquisas é o *Random Forest*. Apesar do algoritmo *Random Forest* apresentar o maior valor de acurácia, deve-se testar outros modelos, pois a classificação de usuários depende também de outras variáveis como a base de dados e as *features* escolhidas.

Alguns modelos utilizaram poucas *features* com bons resultados como em Stukal (Stukal et al., 2019b) que utilizou 40 *features*, Yang (Yang et al., 2020) com 20 *features* e Rossi (2019) que utilizou 14 *features*. Outros modelos mais complexos utilizaram grupos com muitas *features* e também obtiveram alta acurácia como em Davis (Davis et al., 2016) Ferrara (Ferrara et al., 2016b) e Varol (Varol et al., 2017) que utilizaram mais de 1000 *features*.

Autor	**Base de dados**	***Features***	**Modelo**	**Acurácia**
Ferrara (2017)	varol-2017 e cresci-2017	1150	Regressão Logística	>80%
Varol (Varol et al., 2017)	caverlee-2011 e manual	1150	*Random Forest*	>90% humanos e >70% *bots*
Walt (Walt et al.,2018)	manual	22	*Random Forest* AdaBoost SVM	87% 86% 68%
Fernquist (Fernquist et al., 2018)	manual	140	*Random Forest*	Não disponível
Rossi (2019)	cresci-2017 manual	14	*Random Forest*	83%
Stukal (Stukal et al., 2019b)	manual	40	Multilayer Perceptron	>90%

Yang (Yang et al., 2020)	Subconjunto (caverlee, varol-icwsm, cresci-17, pron*bots*, celebrity, vendor-purchased,*bot*ometer-feedback, political-*bots*)	20	*Random Forest*	58% - 99%

Tabela 1: Modelos avaliados e configurações.

Os modelos acima descritos possuem como restrição listas de palavras-chave (Stukal et al., 2017; Fernquist et al., 2018) ou mesmo a linguagem escolhida para a consulta dos dados, o que pode limitar a utilização quanto ao idioma escolhido e ao contexto analisado, devendo assim serem adaptados de forma a contornar a escolha de um novo idioma (Rossi, 2019).

Os métodos de classificação observados nas pesquisas citadas acima utilizaram na sua maioria modelos de aprendizagem de máquina supervisionado. Considerando a necessidade de existir dados rotulados para o treinamento do modelo supervisionado e estes representarem apenas uma parte das possíveis características de comportamento de contas automatizadas limitando o resultado do modelo, foi escolhido para este trabalho modelos de aprendizagem de máquina não supervisionado. Isso permite que os modelos sejam treinados sem a necessidade de dados rotulados, ou seja, é utilizada uma base de dados para treinamento onde as contas de usuários reais já estão assinaladas e o perfil de comportamento dos usuários "normais" não tende a ter grandes mudanças, ao contrário das contas falsas ou de *bots*, que tendem a ser modificadas com o intuito de enganar os classificadores já treinados (Schafer et al., 2017).

Uma observação a ser considerada nesta seção é que não foram avaliados os modelos de classificação de usuários anteriores ao ano de 2015 devido à desatualização dos *datasets* para a fase de treinamento e a descontinuidade de algumas *features* do Twitter.

2.2 Fontes de dados

O Twitter é uma rede social em formato de microblog (forma curta de um blog) que teve seu lançamento em 2006 por Jack Dorsey (Java et al., 2007). Possibilita a publicação de *tweets* e também permite que seus usuários possam seguir outras contas de usuários sem a necessidade de aprovação mútua (Twitter, 2019a).

As mensagens publicadas na ferramenta podem estar na forma de um *tweeting* (um usuário publica sua mensagem), *replying* (resposta à mensagem de outro usuário), *direct tweeting* (resposta direta a outro usuário de formato não público). Para que os usuários possam mencionar-se entre si é

utilizado o símbolo "@ (*mentions*)", para se relacionar com tópicos específicos ou palavras-chave utilizam o símbolo "# (*hashtag*)" criando um agrupamento com todos os *tweets* que contém o tópico especificado, outra forma de comunicação é o *like* e o re*tweet* (do próprio usuário ou de outros), também podem ser divulgados nos *tweets* URLs com *links* para outros sites.

Os componentes de um *tweet* são enumerados abaixo (Twitter, 2019a):

1. *Profile photo*: A imagem pessoal associada a conta do usuário;
2. *Account name*: O nome da conta do usuário;
3. *@Username:* Como o usuário é identificado no Twitter, sempre é precedido de @ e pode ser diferente do account name;
4. *Timestamp:* A data que o *tweet* foi postado no Twitter;
5. *Tweet text*: Uma mensagem com no máximo 280 caracteres;
6. *Hashtags*: Uma palavra ou uma frase precedida sempre por #;
7. *Links*: URL para outros sites;
8. *Tweet actions*: permite ao usuário comentar, re*tweet*, like ou enviar por email;

Estes componentes tem o seguinte formato apresentado na Figura 1 abaixo (Twitter, 2019c):

Figura 1: Os componentes de um *tweet.*

Na coleta dos *tweets*, a seleção do *dataset* a ser utilizado e as *features* possuem extrema importância no resultado dos modelos de classificação, seja para treinamento ou para validação do modelo. Esta importância se deve ao fato de que o modelo apenas poderá ter alta acurácia se as contas de usuários reais e suspeitas possuírem *features* similares às usadas na base de dados para o treinamento do modelo.

Os dados a serem utilizados neste trabalho são divididos em dois conjuntos de dados. O primeiro dataset a ser utilizado para o treinamento dos modelos possui contas já rotuladas manualmente como contas de usuários reais e contas falsas ou automatizadas e são cresci-2017 (Cresci et al., 2017) e verified-2019 (Yang et al, 2020). O conjunto de dados cresci-2017 contém mais de 9.000 contas rotuladas divididos em grupos de *spambots* sociais, *spambots* tradicionais, seguidores falsos e contas genuínas. E o conjunto de dados verified-2019 contém 1.987 contas verificadas como contas de usuários humanos.

O conjunto de dados a ser utilizado posteriormente para a validação do trabalho foi retirado da coleta da API desenvolvedora do Twitter no período relativo as eleições presidenciais ocorridas no Brasil no ano de 2018 e nas eleições da assembléia legislativa para Portugal em 2019.

Esta coleta retorna informações históricas e atuais dos *tweets* (texto, data e hora do *tweet*, idioma, links) assim como os metadados relacionados (características da conta que enviou o *tweet*, incluíndo a ID do autor e o apelido na rede, número de seguidores e seguidos, entre outros).

A API do Twitter fornece um arquivo JSON (Twitter, 2019b) com vários objetos, o objeto pai *tweet* pode conter além de outros objetos filho, mais de 150 *features* e segue a hierarquia abaixo:

- *Tweet* – Também conhecido como um objeto "Status", tem muitos atributos de "nível raiz", pai de outros objetos.
    - *User* – metadados no nível conta do Twitter. Inclui todos os aprimoramentos disponíveis no nível da conta.
    - *Entities* – Contém arrays de objetos de #hashtags, @mentions, $ symbols, URLs e mídia.
    - *Extended Entities* – Contém até quatro fotos nativas ou um vídeo ou GIF animado.
    - *Places* – Pai para o objeto “coordenadas de localização”.

Estas *features* podem ser divididas em textual, imagens e vídeos, metadados relacionados e *network* (Wijeratne et al., 2017), apesar de outros trabalhos agruparem essas *features* em 6 categorias como Network, User, Friends, Temporal, Content e Sentiment (Davis et al., 2016; Ferrara et al., 2016b). A análise da literatura revelou que existem classes de *features* mais importantes que outras, as principais encontradas são as associadas à conta de usuário onde podem ser feitas estatísticas e os metadados associados à essa conta e aos *tweets* (Davis et al.., 2016; Ferrara et al., 2016b; Varol et al.., 2017; Stukal et al., 2017; Fernquist et al., 2018; Walt et al., 2018;

Rossi, 2019, Yang et al., 2020). A conta de um usuário possui uma quantidade expressiva de metadados, por outro lado os *tweets* são limitados a 280 caracteres. Existem características das contas de usuário que são importantes para a análise a ser feita neste trabalho, estas podem ser observadas abaixo:

- Perfil do usuário está na forma padrão ou foi personalizado (geralmente são os usuários humanos que personalizam o perfil como imagem de perfil e fundo de capa);
- Nome da conta de usuário de forma aleatória (contas falsas e *bots* tendem a usar nomes aleatórios);
- Atividade da conta como número de *tweets* publicados e frequência das publicações (contas falsas e *bots* tendem a publicar de forma excessiva e com muita frequência);
- Proporção de seguidores por seguidos (quando essa razão é baixa é provável que sejam contas falsas ou *bots*);
- Proporção de re*tweets* por *tweets* (contas falsas e *bots* usam o re*tweet* para espalhar publicação muito mais que o *tweet*);
- Data de criação da conta (contas falsas e *bots* são em geral recentes e próximas ao período de eleições);
- A língua adotada na criação da conta (não necessariamente a língua utilizada nos *tweets*);
- Número de # e URLs no *tweet* (contas falsas e *bots* não costumam personalizar a mensagem);
- Número de *likes* dados a amigos e seguidores (contas falsas e *bots* não costumam interagir em favoritar amigos e seguidores).

As características mencionadas acima são importantes na produção dos melhores resultados em classificar contas reais e contas anômalas. Verificar se as *features* mencionadas acima foram selecionadas pelo modelo de classificação pode encaminhar para uma boa performance na detecção das contas.

2.3 Apresentação dos algoritmos

A técnica de classificação consiste na construção de classificadores a partir de um conjunto de dados de entrada. O classificador emprega um algoritmo de aprendizagem para identificar um modelo (classificador) que melhor ajusta a relação entre o conjunto de atributos e a categoria rotulada dos dados de entrada, com o objetivo de prever corretamente os rótulos da categoria de

novos exemplos. Sendo assim, o objetivo principal do algoritmo de aprendizagem é construir classificadores capazes de prever com acurácia os rótulos de categoria de exemplos anteriormente desconhecidos (Tan et al., 2006).

Os algoritmos a serem utilizados neste trabalho são descritos abaixo de forma a apresentar a motivação para a sua aplicação na identificação de utilizadores reais e anômalos, bem como na análise do conteúdo presente nas publicações.

2.3.1 Regressão Logística

Regressão logística ou regressão *logit* é usada para estimar a probabilidade de que uma instância possa pertencer a uma determinada classe específica (Géron, 2017). Sendo usada como um classificador com o objetivo de obter um hiperplano decisão que separe diferentes classes, pode-se supor que encontrar uma função dos parametros $\widetilde{w}^1$ e as *features* $\vec{x}$, onde estas informam a probabilidade de um exemplo com *features* $\vec{x}$ pertencer a classe C_1 é dado por:

(1) $$g(\vec{x}, \widetilde{w}^1) = P(C_1|\vec{x})$$

Quando se deseja encontrar o hiperplano decisão onde a probabilidade de encontrar um exemplo da classe C_1 é igual a probabilidade de encontrar um exemplo da classe C_0, assumindo que são somente essas duas classes é dado por:

(2) $$P(C_1|\vec{x}) = P(C_0|\vec{x}) = 1 - P(C_1|\vec{x})$$

Que é equivalente a:

(3) $$ln\frac{P(C_1|\vec{x})}{1-P(C_1|\vec{x})} = 0$$

Então, pode-se escrever:

(4) $$ln\frac{P(C_1|\vec{x})}{1-P(C_1|\vec{x})} = ln\frac{g(\vec{x},\widetilde{w}^1)}{1-g(\vec{x},\widetilde{w}^1)} = \widetilde{w}^T\vec{x} + w_0$$

Rearranjando, obtem-se:

(5) $$g(\vec{x}, \widetilde{w}) = \frac{1}{1+e^{-(\widetilde{w}^T\vec{x}+w_0)}}$$

Sendo a função logística ou sigmóide dada pela função abaixo:

(6) $$f(x) = \frac{1}{1+e^{-k(x-x_0)}}$$

Ao se escolher um valor de corte em (3) onde se separou em duas classes, o modelo de regressão é transformado em um classificador linear. O objetivo da regressão logística é predizer a probabilidade de cada observação pertencer a uma das classes da resposta de interesse. Se a probabilidade estimada for maior que 50%, o modelo prevê que a instância pertence a essa classe (chamada de classe positiva, rotulada como "1"); caso contrário, ela prevê que não (ou seja, pertence à classe negativa, rotulada como "0"). Isso o torna um classificador binário (Géron, 2017) com a saída do modelo um valor de probabilidade.

Em termos gerais, a Regressão logística é um algoritmo de classificação para variáveis categóricas, que tenta predizer um valor alvo discreto ou categórico Para esse algoritmo, a variável dependente deve ser contínua e caso seja categórica, deve ser codificada para um valor contínuo. Pode ser utilizada para classificação binária ou multiclasse, pode também ser utilizada para obter resultados probabilísticos, para obter uma fronteira de decisão linear e para entender a importância dos atributos em um conjunto de dados.

O processo de treinamento do algoritmo fornece um valor de erro total que é chamado de função de custo, que tem como definição a diferença entre o valor no conjunto de treinamento e o valor predito pelo modelo, definido assim para a maioria dos algoritmos de *machine learning*. Um dos objetivos no treinamento é minimizar essa função de custo, que pode ser feito através da escolha otimizada dos parâmetros do modelo (Géron, 2017).

As vantagens do uso do algoritmo Regressão logística é a facilidade para lidar com variáveis independentes categóricas, possuir um alto grau de confiabilidade, ter como saída uma probabilidade de que o ponto de entrada especificado pertença a uma determinada classe (usuário real ou não). Este algoritmo foi um dos escolhidos para ser utilizado na seleção das *features* com o RFE devido a boa performance encontrada na literatura.

2.3.2 *Ensemble*

A classe de métodos ensemble consiste em combinar diferentes hipóteses, seja de modelos de regressão ou classificadores, para melhorar a previsão. Uma maneira de fazer isso é treinar diferentes instâncias de algum modelo com conjuntos de treinamento diferentes e em seguida, agregar a resposta, com média, para problemas de regressão, ou com voto majoritário para classificação (Alpaydin, 2010). Na classe de modelos *Ensemble* existem duas técnicas chamadas como *Bagging* e *Boosting*.

Em *Bagging*, método proposto por Breiman (1994), os classificadores são treinados separadamente e reamostrados com reposição diversas vezes e em seguida agregados através de

métodos de combinação como, por exemplo, a média de votos. O método *Boosting* também é treinado por amostras individuais, mas para esta técnica o método de combinação é uma ponderação do resultado de desempenho de cada modelo.

2.3.2.1 *AdaBoost (Adaptive Boosting*)

O algoritmo *adaptive boosting* ou *AdaBoost* desenvolvido por Freund e Schapire (1995), utiliza uma combinação linear de classificadores fracos, em que cada classificador individual recebe um peso, de modo que a média ponderada das respostas do classificador minimiza o erro de classificação, gerando assim um classificador forte.

Pode-se entender um classificador fraco como um algoritmo simples que tem um desempenho ruim. Um classificador fraco não deve apresentar um bom resultado por ele mesmo apenas, pois isso poderia acarretar *overfitting* na etapa de treinamento (Susjnak, 2009).

Este algoritmo funciona treinando cada classificador com o mesmo conjunto de dados, mas atribuindo pesos diferentes a pontos de dados diferentes, pesando mais fortemente aqueles que foram previamente classificados incorretamente, sendo assim adaptativo para os classificadores subsequentes.

A equação final para a classificação pelo *AdaBoost* pode ser representada como:

(7) $$f(x) = sign(\sum_{m=1}^{M} \alpha_m y_m(x))$$

Onde y_m representa o m-ésimo classificador fraco e α_m é o peso correspondente. É exatamente a combinação ponderada de M classificadores fracos. Durante a fase de treinamento os classificadores iniciam com pesos igualmente distribuídos. Após o inicio das iterações, os pesos dos dados classificados de forma errada cresce enquanto diminui os pesos dos dados classificados corretamente. Ao fim do treinamento o *AdaBoost* combina todas as classificações intermediárias de forma ponderada gerando uma classificação única final (Freund e Schapire, 1999).

Se forem combinados vários classificadores com a seleção do conjunto de dados a cada iteração e atribuir a quantidade certa de peso na votação final, é possível obter uma boa performance para o classificador geral.

Um respaldo que deve ser feito sobre a utilização deste algoritmo é a sua sensibilidade a *outliers* e ruídos no conjunto de dados utilizado no treinamento, pois pesos atribuídos a ruídos e *outliers* pode impactar negativamente a performance do classificador com a função de custo exponencial, já

que as penalidades por uma classificação incorreta crescem exponencialmente com a magnitude da saída da função preditora (Hastie et al., 2008).

As vantagens em utilizar este algoritmo é devido ao *AdaBoost* apresentar parâmetros que podem ser utilizados para analisar dados de grandes dimensões, outra propriedade é o baixo custo computacional, dado que corresponde a um programa de complexidade linear e evita o uso de componentes computacionais pesados (Freund e Schapire , 1999).

Este algoritmo foi um dos escolhidos para ser utilizado na seleção das *features* com o RFE devido a boa performance encontrada na literatura.

2.3.2.2 *Random Forest*

O algoritmo do *Random Forest* proposto primeiramente por Breiman (2001), utiliza o método *Bagging*, descrito anteriormente. Este pode ser definido simplificadamente como uma coleção de árvores de decisão, onde cada árvore tenta estimar uma classificação sendo considerado como um voto, onde será escolhida a classificação com mais votos.

A idéia principal da *Random Forest* é realizar um sorteio aleatório dos preditores no conjunto de dados de treinamento com o objetivo de reduzir a correlação entre as árvores agregadas sem aumentar o valor da variância. Este crescimento da árvore através da seleção randômica das variáveis de entrada é que permite melhorar a redução da variância. Para além desta idéia, existe uma facilidade de paralelizar o processo devido à independência em estimar cada árvore (Friedman et al., 2001).

A *Random Forest* começa com *B* conjuntos de dados de tamanho *n* por amostragem obtidos com reposição do conjunto de treinamento e em seguida, para cada conjunto, estima uma árvore de decisão (Friedman et al., 2001).

O funcionamento do algoritmo se baseia na construção da árvore T_b, onde uma amostra randômica do conjunto de *p* variáveis, com tamanho *m* e sem reposição é escolhida e no subconjunto será realizada a escolha da melhor variável e do valor do ponto de corte em relação a variável de interesse. Em seguida a partir de dois nós filhos é dividido o nó e criado o ensemble das árvores $\{T_b\}_1^B$, onde b varia de 1 a B.

Para os problemas de classificação, a resposta final do algoritmo pode ser dada pela equação abaixo:

(8) $$\hat{C}_b(x) = mais\ votado\{\hat{C}_b(x)\}_1^B$$

Onde $\hat{C}_b(x)$ é predição da *b*-ésima árvore *random forest*, ou seja, a classe preditora de uma nova observação será a que tiver mais votos entre as B árvores *random forest* (Hastie et al., 2008).

Em comparação as D*ecision Tree*, a utilização da *Random Forest* se deve ao fato de que ao trabalhar com um conjunto de dados que possuam muitas *features*, uma árvore de decisão com muitas *features* tem uma grande profundidade e pode responder bem a dados conhecidos, mas perde poder de generalização a dados desconhecidos, ou seja, pode ocorrer *overfitting*. Para evitar este fenômeno, a utilização da *Random Forest* é vantajosa pois em vez de procurar o melhor recurso ao dividir um nó, este procura o melhor recurso em um subconjunto aleatório de recursos e constrói árvores menores a partir de tais subconjuntos. Isso resulta em uma maior diversidade de árvores, que troca um viés mais alto por uma variância menor, o que geralmente produz melhores modelos globais (Géron, 2017). È um algoritmo que tem a capacidade de lidar com dados em grandes volumes e com muitas dimensões.

Este algoritmo foi um dos escolhidos para ser utilizado na seleção das *features* com o RFE devido a boa performance encontrada na literatura.

2.3.3 SVM (*Support Vector Machine*)

O algoritmo SVM foi primeiramente elaborado por Cortes e Vapnik (1995) concebendo um novo método de *machine learning* para problemas de classificação de duas classes, onde vetores de entrada são mapeados em um espaço de *features* de alta dimensão utilizando uma função. Nesse espaço, uma superfície de decisão linear é criada com determinadas propriedades que garantem uma alta capacidade de generalização do algoritmo. Esta superfície é um hiperplano de decisões, definindo o objetivo do SVM, que é encontrar um hiperplano em um espaço dimensional N, sendo N o número de *features*, que classifique distintamente os pontos de dados.

Ao procurar um hiperplano também se procura encontrar um plano com a margem máxima, ou seja, a distância máxima entre os pontos de dados das duas classes. É na maximização da distância da margem que se busca garantir que o modelo terá boa capacidade de generalização (Vapnik et al.,1995).

Em outras palavras, dado dados de treinamento rotulados, o algoritmo gera um hiperplano ideal que categoriza novos exemplos. No espaço bidimensional, este hiperplano é uma linha que divide um plano em duas partes, onde em cada classe se encontra um hiperplano de cada lado (Alpaydin, 2010). O SVM coloca o maior número de pontos da mesma classe do mesmo lado, enquanto maximiza a distância de cada classe a esse hiperplano. A distância de uma classe a um hiperplano é a menor distância entre ele e os pontos dessa classe e é chamada de margem de separação. O

hiperplano gerado pelo SVM é determinado por um subconjunto dos pontos das duas classes, chamado de vetores de suporte, que define a máxima separação entre duas classes.

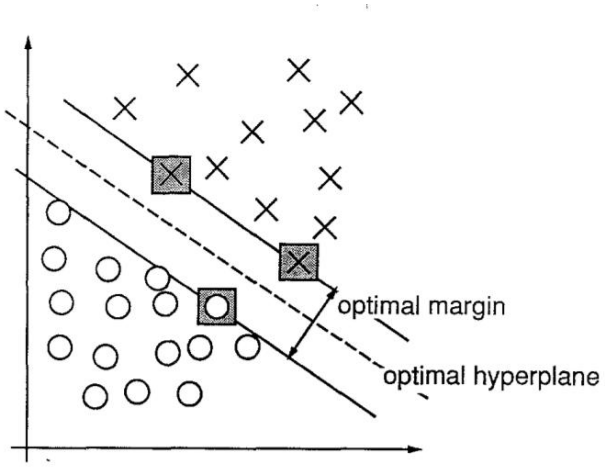

Figura 2: Exemplo de classificação em um espaço bidimensional. Os vetores de suporte estão marcados com um quadrado cinza (Vapnik et al., 1995).

O SVM apresentando por Cortes e Vapnik (1995) se baseia em duas teorias, a teoria de aprendizado estatístico e na teoria de otimização matemática.

A teoria de aprendizado estatístico (Vapnik et al.,1995) estabele condições matemáticas que permitem a escolha de um classificador a partir do conjunto de dados usados para treinamento. As condições matemáticas são importantes pois tem como objetivo obter um modelo com bom desempenho seja nos dados de treinamento, seja para novos dados (Lorena et al., 2007).

Sendo o SVM um classificador linear (Vapnik et al.,1995), este pode ser representado pela função linear descrita na equação abaixo:

(9) $$f(z) = sign(w \cdot z + b)$$

Onde z é o vetor no espaço das *features*, *b* é o *bias* e w é a combinação linear de vetores de suporte e representa o peso para o hiperplano ótimo no espaço das *features*.

(10) $$sign(g) = +1\ para\ g \geq 0\ ;\ sign(g) = -1\ para\ g < 0$$

Onde o valor resultante da função linear demonstra que o hiperplano criado divide a classe de resposta em resposta positiva e negativa. Tanto o *b* quanto o w são ajustados durante o processo de treinamento do SVM.

Um conjunto de pontos de treinamento rotulados $(y_1,x_1),\ldots,(y_l,x_l)$, com y ∈ {-1,1} é dito sendo linearmente separável se existe um vetor w e um escalar *b* tal que as restrições dadas abaixo são válidas para todos os elementos do conjunto de treino.

(11) $$y_i(w \cdot x_i + b) \geq 1, \qquad i = 1, \ldots, l$$

Onde y_i representa o valor alvo e x_i representa os dados de entrada. Estas restrições são importantes para garantir que não exista dados de treinamento entre as margens de separação das classes. Isso implica em um modelo extremamente rígido em relação às margens (Vapnik et al.,1995). Obter o hiperplano separador otimizado é maximizar a margem de separação dos dados, isso fornece ao modelo a capacidade de generalizar bem com novos dados, ou seja, encontrar o hiperplano com a maior margem de separação possível em relação aos vetores de suporte. Para isso é utilizada a teoria de otimização matemática.

O procedimento de otimização matemática que maximiza essa margem é na maior parte dos casos resolvido pela utilização do gradiente descendente, mas também podem ser empregadas técnicas como Inversão de matriz, Método de Newton, Otimização mínima sequencial, entre outras.

Quando os dados são linearmente separáveis, é utilizado o SVM linear. Na maior parte das aplicações do SVM em problemas reais, os dados não são linearmente separáveis, não sendo possível utilizar um hiperplano que separe perfeitamente os pontos. Neste caso, quando não for possível separar as classes linearmente, podem ser utilizadas variáveis de folga ou então pode ser utilizado o SVM não linear (Vapnik et al.,1995; Souza, 2005).

As variáveis de folga são representadas por ξ_i, onde i = 1,...,n, transformando a função dada em (11) para:

(12) $$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, l$$

Essas variáveis de folga permitem que erros de classificação ocorram ou que algum dado permaneça entre os hiperplanos (Vapnik et al.,1995; Lorena et al., 2007).

Uma questão importante quando se utiliza o SVM com os dados não linearmente separáveis é como se pode transformar os dados de forma que o separador possa ser desenhado como um hiperplano. Para isso pode ser utilizado o SVM não linear detalhado abaixo.

A transformação dos dados através do teorema de Cover esclarecido em Haykin (1999), permite a separação destes através da criação de um hiperplano e para isso se utiliza uma função que permite aumentar a dimensão dos dados em uma alta dimensão e criar o hiperplano separador. Essa

transformação dos dados em uma alta dimensão dos dados é chamada de "*Kernelling*" de forma que os dados linearmente não separáveis são transformados em linearmente separáveis. A função matemática usada para a transformação dos dados é conhecida por função kernel e é apresentada abaixo:

(13) $$K(x, x') = \phi(x) \cdot \phi(x')$$

Onde *K* é a função kernel, *x* e *x'* são pontos dos dados de entrada e ϕ representa o mapeamento dos dados de entrada para um novo espaço de maior dimensão. A função kernel recebe os pontos dos dados de entrada e realiza o cálculo do produto escalar dos vetores para transformar em um novo espaço de *features* usualmente de alta dimensão (Scholkopf et al., 2002).

Uma das possibilidades de se verificar que o kernel dentro da otimização matemática é válido é se este atende às condições necessárias contidas no teorema de Mercer (Mercer, 1909). Este teorema é válido se a função kernel produzir matrizes positivas semi-definidas de *K* (Lorena et al., 2007).

As principais funções kernel são linear, polinomial, *radial basis function* (RBF) e rede neural sigmóide de duas camadas e estão apresentadas abaixo (Scholkopf et al., 1999a):

Linear:

(14) $$K(x, x') = (x \cdot x')$$

Polinomial:

(15) $$K(x, x') = (x \cdot x')^d, \ d \in \mathbb{N}$$

Onde d indica o grau da função polinomial.

Radial basis function (RBF) ou Gaussiano:

(16) $$K(x, x') = exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right), \ \sigma > 0$$

Onde σ indica a função de base radial guassiana.

Rede Neural Sigmóide:

(17) $$K(x, x') = S[u(x, x') + c]$$

Onde S(v) é uma função sigmóide e u e c são escalares e devem ser adaptados para atender as condições do teorema de Mercer.

A função kernel linear faz parte do uso de SVM linear para dados que são linearmente separáveis ou quando o número de *features* for maior que o número de amostras dos dados, as demais funções acima fazem parte do SVM não linear, quando os dados são não linearmente separáveis. Como não existe uma forma fácil de conhecer qual função kernel não linear performa melhor, de forma geral são utilizadas as funções descritas acima e comparados os resultados dos modelos.

2.3.3.1 SVM *one-class*

A detetcção de *outliers* ou novidades em um conjunto de dados pode ser realizada utilizando o algoritmo SVM *one-class*, sendo este uma adaptação introduzida por Scholkopf (Scholkopf et al., 1999b) do SVM explicitado acima para a classificação de uma classe única.

Como pode ser observado em Manevitz e Yousef (2001), após a transformação da feature através da função kernel, a origem foi tratada como único membro da segunda classe. Em seguida foi utilizado parâmetros de relaxamento para separar a imagem de uma classe da origem e as técnicas do SVM de classe binária foram então utilizadas. Ou seja, a estratégia adotada por Scholkopf foi de mapear os dados no espaço das *features* correspondente ao kernel e separá-los da origem com o máximo de margem.

Dado um novo ponto x, o valor da função *f(x)* é determinada pela avaliação de qual lado do hiperplano o ponto é atribuído, no espaço das *features*. A abordagem feita por Manevitz e Yousef (2001) para melhor entedimento é demonstrada abaixo:

“ Suponha que um conjunto de dados tenha uma distribuição de probabilidade *P* no espaço das *features*. Encontre um subconjunto “simples” *S* do espaço das *features* de forma que a probabilidade de um ponto de teste de *P* ficar fora de *S* seja limitada por algum valor especificado a priori. Supondo que existe um conjunto de dados extraído de uma distribuição de probabilidade subjacente *P*, é necessário estimar um subconjunto “simples” *S* do espaço de entrada, de forma que a probabilidade de um ponto de teste de *P* ficar fora de *S* seja limitada por alguma especificação a priori $\nu \in (0, 1)$.“

Continuando a abordagem acima, a solução proposta por Scholkopf foi desenvolver um algoritmo que estima a função *f* e esta retorna o valor +1 em uma região capturando a maior parte dos dados sendo uma região de alta densidade de pontos, e -1 em uma região muito menor com baixa densidade dos pontos, sendo esta a região dos *outliers*.

O algoritmo pode ser sintetizado como um mapeamento dos dados no espaço das *features* denominado por *H* utilizando uma função kernel escolhida e então realiza-se a tentativa de separar os vetores mapeados da origem com a margem máxima, conforme pode ser visto na Figura 3.

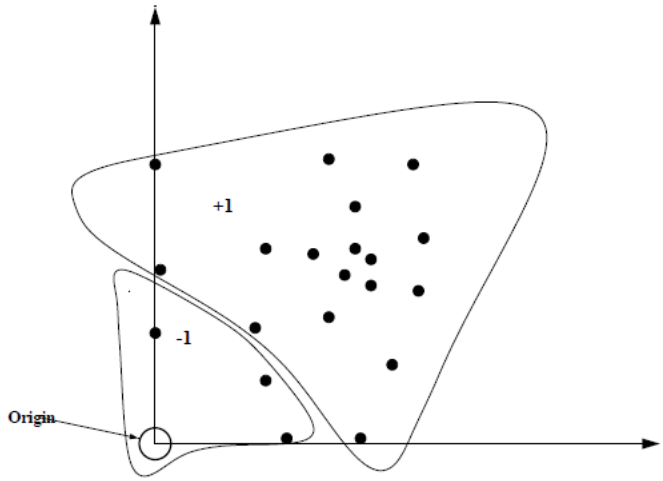

Figura 3: Classificador SVM *one-class*. Separação dos vetores na região positiva e na região dos *outliers* (Manevitz e Yousef, 2001).

Considerando a função de separação *f* dada por:

(18) $$f(x) = +1 \; para \; x \; \epsilon \; S \; ; \; f(x) = -1 \; para \; x \; \epsilon \; \bar{S}$$

Onde *S* e $\bar{S}$ representam o subconjunto e seu complemento respectivamente.

Dado o conjunto de dados de treinamento $x_1, \ldots, x_l$ que pertence a uma classe X, onde X é um conjunto compacto de R^N, sendo $\Phi : X \rightarrow H$ uma função de kernel a qual transforma os dados de treinamento para outro espaço, a separação do conjunto de dados da origem é resolvida através da solução do seguinte problema quadrático:

(19) $$min \frac{1}{2} \| w \|^2 + \frac{1}{vl} \sum_{i=1}^{l} \xi_i - \rho$$

Sujeito a seguinte condição:

(20) $$\big(w \cdot \phi(x_i)\big) \geq \rho - \xi_i, \quad i = 1, \ldots, l, \qquad \xi_i \geq 0$$

E a função decisão é dada por:

(21) $$f(x) = sign\big(w \cdot \phi(x)\big) - \rho$$

Desde que as variáveis de folga ξ_i são penalizadas na função objetivo, então w e ρ solucionam o problema em (20). A resposta da função é positiva para a maior parte dos exemplos contidos no conjunto dos dados de treinamento, e os *outliers* são detectados pela função pelo resultado ser negativo (Scholkopf et al.,1999b).

No pacote *scikit-learn* é possível utilizar este modelo para detecção de anomalia considerando que um conjunto de dados de *n* observações da mesma distribuição descrita por *p* características, ao ser adicionada mais uma observação a esse conjunto de dados e esta for diferente, então é identificado como anômala.

O treinamento do algoritmo é realizado através da seleção dos dados do conjunto de treino que são considerados a maioria, ou seja, não há *outliers* ou ruídos. Deve ser selecionada a função kernel a ser utilizada bem como os parâmetros.

As vantagens de se usar o SVM são a alta acurácia em grande dimensão de dados e a eficiência na memória por utilizar um subconjunto de pontos de treinamento na função decisão, os vetores de suporte. A desvantagem do uso do SVM é a possibilidade de *overfitting* se o numero de *features* for muito maior que o numero de amostras. É utilizado em reconhecimento de imagens, categoria de texto, detector de spams, análise de sentimento, regressão, detecção de *outliers* e clusterização.

O SVM além de outros classificadores como por exemplo K-NN, *Decision Tree*, difere por não estimar diretamente um valor de probabilidade, mas sim a classe de resposta de interesse para uma nova observação.

Como o algoritmo SVM em geral fornece bons resultados de generalização, possui robutez em grandes dimensões e pode ser utilizado em problemas de classificação de classe única e também binária será utilizado neste trabalho para seleção das *features* por tambem ser citado na literatura devido a boa performance e para a detecção de anomalias, pois sendo aprendizado não supervisionado é treinado apenas com as informações das contas normais para que na utilização dos dados de teste as contas anômalas sejam identificadas como ruídos ou *outliers*, sendo consideradas como uma anomalia.

2.3.4 *Word embedding*

O conjunto de técnicas *Word Embedding* aprende uma representação vetorial das palavras de forma que palavras que são semanticamente relacionadas estão próximas entre si no espaço vetorial. Teve início em 1960 com o desenvolvimento de modelos de espaço vetorial para recuperação de informação. Este tipo de técnica tem como objetivo reduzir a alta dimensionalidade das

representações de palavras em contextos "aprendendo uma representação distribuída para palavras" (Bengio et al., 2003).

Para encontrar uma representação numérica de variáveis categóricas que são relevantes para uma tarefa específica, pode-se incorporar suas codificações *one-hot* em um espaço vetorial de menor dimensão. Incorporadores são vetores distribuídos, densos, contínuos, de tamanho fixo que representam numericamente uma variável categórica (Bengio et al., 2003).

O modelo proposto por Bengio (Bengio et al., 2003) teve como objetivo a modelagem de variáveis contínuas para ultrapassar a limitação do *curse of dimensionality*, utilizando para isso a representação das palavras distribuídas em vetores com a dimensão do espaço dos incorporadores, reduzindo a dimensão. Adicionalmente a isto, é utilizada uma função que permite calcular a probabilidade de palavras sequenciais. Este modelo funciona com os vetores de cada categoria sendo treinados durante a fase de treinamento da rede neural, permitindo explorar quais palavras são semelhantes umas às outras em um espaço multidimensional.

Portanto, no final da fase de treinamento, tem-se um vetor que representa cada categoria. Assim, utilizar embedding permite ter uma matriz embedding para manter o tamanho de cada vetor muito menor, existindo um vetor exclusivo para cada categoria da variável categórica.

A utilização de incorporadores embedding em dados com várias palavras e textos é preferível quando comparada aos vetores *one-hot encoded*, utilizados para transformar pequenas palavras em variáveis contínuas, estes últimos são de alta dimensão o que pode causar baixa eficiência computacional.

Dentro deste conjunto, existem diferentes tipos de modelos. Existem modelos que utilizam os documentos como contexto por exemplo, como a Análise Semântica Latente (ASL). O modelo do Word2Vec utiliza as palavras como contexto, sendo explicitado abaixo.

2.3.4.1 *Word2Vec*

Criado em 2013 por Mikolov (Mikolov et al., 2013), é uma forma especialmente eficiente de treinar *Word Embeddings* e é utilizada para computar representação vetorial das palavras. A abordagem utiliza duas arquiteturas diferentes para criar um modelo de rede neural de maior performance, seja no custo computacional seja na acurácia que preserve a regularidade linear entre as palavras.

O trabalho desenvolvido focou em representações distribuídas de palavras aprendidas por redes neurais e apresentou desempenho melhor que o ASL em preservar a regularidade linear entre as

palavras, para isto utilizaram um corpus de alta dimensão. Ao propor comparar diferentes arquiteturas de modelo, foi definido o método de cálculo da complexidade computacional, sendo este função do número de parâmetros que precisam ser acessados para treinar completamente o modelo (Mikolov et al., 2013).

As abordagens anteriores ao *Word2Vec* eram lentas devido a camada não linear na rede neural, a utilização da função softmax calculando no treinamento a probabilidade de cada outra palavra do vocabulário. Sendo uma rede neural linear com apenas uma camada, é uma rede simples e rápida no treinamento, que compensa a simplicidade do modelo no treinamento da rede ao utilizar muitos exemplos. Abaixo são apresentadas as arquiteturas propostas pelos autores.

2.3.4.1.1 Modelo Continuo *Bag-of-words (CBOW)*

Neste modelo, as palavras mais próximas, ou seja, o contexto é utilizado para realizar a predição da palavra atual (Mikolov et al., 2013). É uma rede neural alterada com a remoção da camada escondida e a mudança na camada de projeção, sendo à saída desta última camada a média dos vetores da camada de entrada, como pode ser visto na Figura 4.

A complexidade computacional deste modelo pode ser calculada pela equação abaixo:

(22) $$Q = N \times D + D \times \log_2(V)$$

Onde N representa o tamanho da camada de projeção, D a dimensão dos vetores e V é o tamanho do vocabulário.

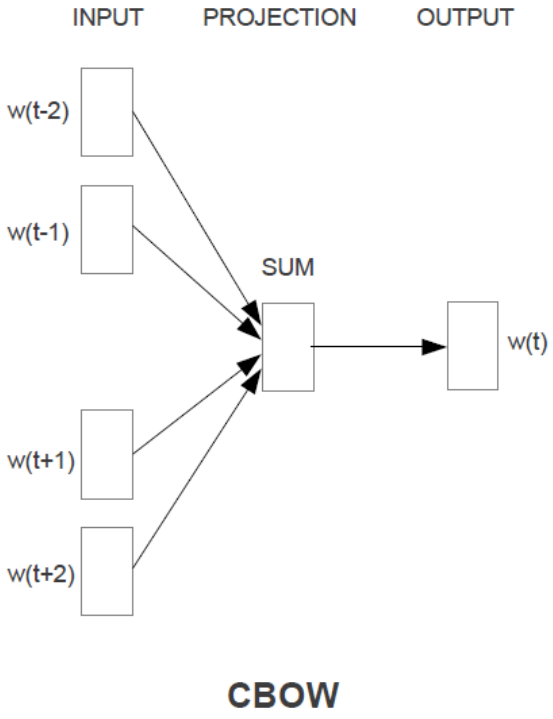

Figura 4: A arquitetura CBOW prevê a palavra atual com base nas palavras mais próximas (Mikolov et al., 2013).

2.3.4.1.2 Modelo Continuo *Skip-gram*

A segunda arquitetura é similar a CBOW, mas em vez de prever a palavra atual com base no contexto, ele tenta maximizar a classificação de uma palavra com base em outra palavra na mesma frase. Ou seja, inversamente ao CBOW, ele utiliza uma palavra para realizar a predição das palavras mais próximas no contexto conforme Figura 5.

A complexidade computacional deste modelo pode ser calculada pela equação abaixo:

(23) $$Q = C \times (D + D \times \log_2(V))$$

Onde C é a máxima distância das palavras, D a dimensão dos vetores e V é o tamanho do vocabulário.

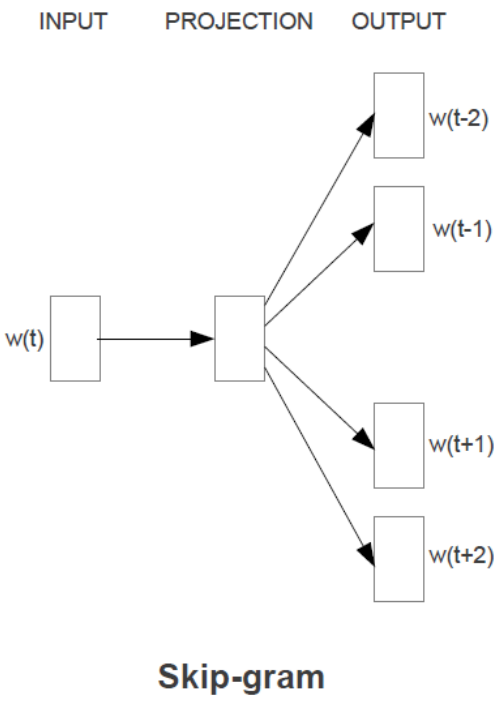

Figura 5: A arquitetura Skip-gram prevê as palavras mais próximas com base na palavra atual (Mikolov et al., 2013).

Comparando as duas arquiteturas, estas estão relacionadas ao tamanho dos dados a serem utilizados. O CBOW apresentará uma melhor representação distribuída para as palavras mais frequentes em um conjunto de exemplos de grande dimensão, ao contrário da Skip-gram que irá performar melhor em uma menor dimensão do conjunto de exemplos e em apresentar palavras raras (Mikolov et al., 2013).

2.3.5 *Autoencoders*

O *Autoencoder* é uma rede neural artificial não supervisionada que aprende como compactar e codificar eficientemente os dados, e aprende como reconstruir os dados da representação codificada reduzida para uma representação o mais próximo possível da entrada original (Kramer, 1991). Ou seja, *Autoencoders* são redes não supervisionadas que aprendem a comprimir as entradas e é considerada uma técnica de redução de dimensionalidade. Este mesmo algoritmo também pode ser utilizado para detecção de *outliers*.

A arquitetura da rede que é totalmente conectada é construída através da utilização de uma camada de entrada e saída com a mesma dimensão *m* e com a utilização de uma camada escondida com dimensão *n* menor que a dimensão das camadas de entrada e saída (Haykin, 2009). A rede é

constituída de dois componentes, uma função encoder $h = f(x)$ e uma função decoder $r = g(h)$ como pode ser vista na Figura 6 abaixo.

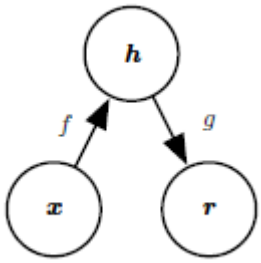

Figura 6: Estrutura geral de um Autoencoder (Goodfellow et al, 2016).

O encoder é a parte da rede que compacta a entrada em uma representação de espaço latente. O decoder visa reconstruir a entrada da representação do espaço latente. O autoencoder como um todo pode ser descrito pela função $g(f(x)) = r$ onde se deseja r o mais próximo da entrada original x.

O codificador dado por f tenta mapear a entrada $x \in R^{dx1}$ para a representação da camada escondida, dada por $z \in R^{rx1}$, onde r é o número de neurônios da camada escondida (Kan et al., 2014). Geralmente f consiste de uma transformação linear e de sucessivas transformações não lineares dada por:

(24) $$z = f(x) = s(Wx + b)$$

Onde $W \in R^{rxd}$ é uma matriz de pesos, $b \in R^{rx1}$ é o bias e s é a função de ativação geralmente não linear como uma sigmóide ou tangente.

O decodificador dado por g tenta mapear a representação escondida z de maneira a reconstruir a entrada x, dada abaixo por:

(25) $$x = g(z) = s(\widehat{W}z + \hat{b})$$

Onde $\widehat{W} \in R^{rxd}$ é uma matriz de pesos, $\hat{b} \in R^{dx1}$ é o bias e s é a função de ativação geralmente a mesma usada no encoder.

A quantidade de perda de informações entre o output do decoder e o input do encoder é uma função de perda (*loss function*) (Baldi, 2012) sendo empregada usualmente o erro quadrático médio e pode ser dada pela equação abaixo:

(26) $$\left[W^*, b^*, \widehat{W}^*, \hat{b}^*\right] = \arg min_{W,b,\widehat{W},\hat{b}} \sum_{i=1}^{N} \| x_i - g(f(x_i)) \|_2^2$$

Onde a otimização dos parâmetros W, b, $\widehat{W}$, $\hat{b}$ minimiza a função de perda e onde x_i representa o *iésimo* exemplo de N dos exemplos de treinamento (Kan et al., 2014).

Devido a não linearidade da função de ativação, a equação (26) é de díficil resolução e para isso é geralmente utilizado o algoritmo do gradiente descendente.

A função de perda mede a captação dos dados de entrada que sejam suficientes para que ocorra a reconstrução dos dados na saída. Para que a rede não realize a memorização dos dados ou o ajuste excessivo (*overfitting*) dos dados de treinamento são utilizadas formas de regularização.

Se a camada escondida tiver menor número de neurônios que a camada de entrada, a camada escondida poderá conseguir extrair as informações essenciais dos dados. É essa diminuição no número de neurônios que força a camada escondida aprender a maioria dos padrões dos dados e ignorar os ruídos, ou seja, o número de dimensões da camada escondida será menor que a de entrada ou saída (Goodfellow et al, 2016).

A quantidade de camadas escondidas pode ser aumentada para tentar melhorar o aprendizado da representação mais importante dos dados, mas com a utilização de apenas uma camada escondida com dimensão menor que as demais camadas da rede, sendo esta conhecida por *bottleneck*. As demais camadas escondidas devem ter dimensão superior a esta última citada e inferior as camadas de entrada e saída.

Existem algumas arquiteturas padrões de autoencoders, sendo as mais comumentes utilizadas *Undercomplete*, *Sparse*, *Denoising* e *Contractive*. A utilização de cada uma dessas arquiteturas depende da natureza do conjunto de dados e o objetivo a ser alcançado na utilização desses dados.

2.3.5.1 *Undercomplete Autoencoder*

A arquitetura deste autoencoder é considerada como a mais simples, devido a restrição do número de neurônios na camada escondida, a informação a ser passada pela rede é limitada. Para o ajuste dos pesos na rede é utilizada a função de perda, realizando assim o aprendizado das informações mais importantes dos dados, ou seja, as informações latentes dos dados de entrada e como reconstruir os dados para uma representação mais fidedigna da entrada original. Esta arquitetura é capaz de aprender relacionamentos não lineares, sendo uma generalização mais poderosa que o PCA (Goodfellow et al, 2016).

Para este modelo, não há o termo de regularização, a rede é treinada com a informação da função de perda. Para evitar a memorização dos dados é necessário garantir que o número de neurônios é muito menor nas camadas escondidas que nas camadas de entrada e saída.

2.3.5.2 *Sparse Autoencoder*

Para esta arquitetura de rede não é realizada uma redução do número de neurônios na camada escondida, mas sim a ativação de um menor número de neurônios através da penalização das ativações na camada escondida. Ou seja, esta técnica utiliza a regularização na ativação de determinados neurônios e não através do ajuste nos pesos da rede. Isso implica que os neurônios que são ativados são dependentes dos dados de entrada, assim dados de entrada diferentes resultarão em ativação de neurônios diferentes através da rede. Com essa definição de ativar neurônios de forma seletiva, a rede extrai as informações mais importantes dos dados de entrada sem implicar no problema do overfitting, construindo esparsidade na camada escondida.

De forma a garantir a esparsidade na rede, a ativação na camada escondida é medida para cada conjunto de dados de treinamento e é adicionado peso a função de perda caso exista ativação excessiva nos neurônios da camada escondida. Existem várias formas de regularização que podem ser adotadas, sendo comumente denotada pela equação abaixo:

(27) $$L\left(x, g\left(f(x)\right)\right) + \Omega(h)$$

Onde *L* representa a função de perda, $h = f(x)$ a saída do encoder, $g(h)$ é a saída do decoder e $\Omega(h)$ a penalização dada pela esparsidade na camada escondida. A esparsidade pode ser definida como:

(28) $$\Omega(h) = \lambda \sum_i |h_i|$$

Onde λ é o hiperparâmetro que fornece o peso aos neurônios da camada escondida (Goodfellow et al, 2016).

2.3.5.3 *Denoising Autoencoder*

Este modelo de rede neural tem como objetivo conseguir reconstruir dados de entrada a partir de dados de entrada corrompidos pela adição de ruídos, conseguindo reproduzir na saída os dados sem a presença do ruído. A incorporação de ruídos na camada de entrada força o autoencoder a aprender as *features* mais robustas e distingui-las dos ruídos adicionados (Vincent et al, 2008). Esta arquitetura proporciona uma rede capaz de generalizar bem para novos dados, pois os dados de saída não são idênticos aos dados de entrada, impedindo a memorização dos dados de treinamento pela rede, mas não realiza redução de dimensionalidade.

Com esta abordagem, não é necessário adicionar um peso a função de perda, o treinamento consiste em minimizar a função de perda para os dados de entrada corrompidos pelo ruído adicionado, sendo representado pela equação abaixo:

(29) $$L\left(x, g\left(f\left(\widetilde{x}\right)\right)\right)$$

Onde $g(f(\tilde{x}))$ é a saída do decoder e $f(\tilde{x})$ é a saída do encoder a partir dos dados de entrada com ruídos.

A arquitetura da rede é representada pela Figura 7 abaixo:

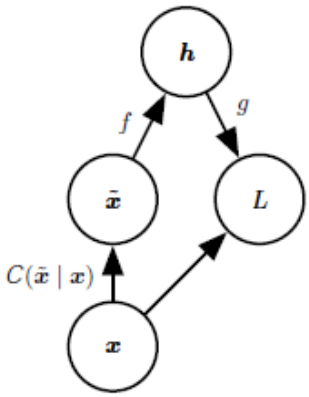

Figura 7: Estrutura do Denoising Autoencoder (Goodfellow et al, 2016).

O ruído representado por $(C(\tilde{x}|x)$ é comumente adicionado aos dados de entrada através da utilização de um ruído Gaussiano ou mesmo de uma fração do conjunto dos dados de entrada randomicamente retirados através de dropout (Géron, 2017).

2.3.5.4 *Contractive Autoencoder*

Considerada outra arquitetura de rede, o Contractive Autoencoder utiliza um termo de regularização que força o autoencoder a aprender as *features* mais robustas através de mudanças pequenas nos dados de entrada. Sendo muito parecido com o Sparse Autoencoder, onde um termo de penalização é adicionado a função de perda como demonstrado na equação 27, o termo de penalização para esta rede é diferente a da rede Sparse e corresponde a norma de Frobenius (soma do quadrado dos elementos) de uma matriz Jacobiana de derivadas parciais associadas a função encoder e é apresentado pela equação abaixo:

(30) $$\Omega(h) = \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

Onde λ é o hiperparâmetro que controla a regularização.

Esta arquitetura introduz um regularizador explícito encorajando as derivadas de *f* a serem tão pequenas quanto possível (Goodfellow et al, 2016), sendo treinado para conseguir resistir as pertubações infinitesimais na entrada e realizar a função de extração de *features*. Ou seja, o modelo é forçado a contrair uma vizinhaça de entradas em uma vizinhaça menor de saídas.

2.3.5.5 Utilização dos *Autoencoders*

Autoencoders podem ser usados para redução de dimensionalidade, sendo esse um dos principais usos de Autoencoders. Durante o processo de redução de dimensionalidade, os *outliers* ou ruídos também podem ser identificados, então essa identificação é um subproduto da redução de dimensionalidade. Com restrições de dimensionalidade e esparsidade apropriadas, os *autoencoders* podem aprender projeções de dados mais interessantes que o PCA ou outras técnicas básicas.

A utilização de Autoencoders em substituição ao PCA é devido as técnicas de autoencoder poderem executar transformações não lineares com função de ativação não linear em várias camadas escondidas dessa rede. É mais eficiente treinar várias camadas com um autoencoder no lugar de treinar uma enorme transformação com o PCA. As técnicas de autoencoders demonstram mérito quando os dados são de natureza complexa e não linear.

Os autoencoders também podem ser utilizados para detecção de anomalias (Sakurada and Yairi 2014; Zhou and Paffenroth 2017). Como a *loss function* (equação 26) de um autoencoder mede o erro de reconstrução, podemos extrair a informação para identificar novas observações de um conjunto de dados que apresentam taxas de erro maiores.

Essas novas observações possuem atributos que diferem significativamente dos outros recursos do conjunto de dados, podendo assim ser considerados como recursos anômalos ou *outliers*. Ou seja, um autoencoder treinado em X ganha a capacidade de reconstruir instâncias invisíveis a partir dos mesmos dados distribuição como X. Se uma nova instância não pertencer ao conceitos aprendidos com X, então esperamos a reconstrução ter um erro alto em relação ao valor limite definido como ponto de corte, identificando essa nova instância como uma anomalia. Assim se a rede aprende o conceito de dados normais, o modelo fica pronto para detectar dados anômalos que não foram visualizados nos dados de treinamento.

Esta abordagem acima possui algumas limitações como a possível baixa eficácia da rede em um conjunto de dados de treinamento pequeno devido ao *overfitting* causado pelo grande número de parâmetros, embora a rede tenha boa performance para grandes conjuntos de dados de treinamento. Outra possível limitação é a convergência apontar para ótimos locais. O aumento no conjunto de

dados de treinamento pode reduzir o *overfitting* mas acarreta em um maior desafio computacional (Chen et al, 2017).

2.3.5.6 *Autoencoder Ensembles*

Com o objetivo de contornar as limitações mencionadas acima para detecção de anomalias podem ser utilizados os Autoencoder Ensembles. Esta rede neural tem sido a melhor técnica em comparação a outros métodos para este uso específico (Chen et al, 2017).

Métodos de aprendizagem ensemble são algoritmos que combinam predições de diferentes detectores com objetivo de criar resultados mais robustos. A utilização de uma combinação de múltiplos detectores pode não performar melhor do que o melhor detector presente nesta combinação, então para garantir que o método ensemble funcione com boa performance, os componentes individuais ensemble devem capturar diferentes partes de padrões subjacentes nos dados.

Uma das possíveis abordagens, diferente da tradicional em que a arquitetura da rede é totalmente conectada, é a utilização de vários autoencoders conectados randomicamente com diferentes estruturas e densidades de conexão, reduzindo assim a complexidade computacional.

Como a utilização de vários autoencoders com a mesma arquitetura de rede totalmente conectada não produz resultados diversificados, é necessário variar a arquitetura destes múltiplos e completamente independentes autoencoders com o uso de camadas da rede randomicamente conectadas. Através de conexões randomicamente selecionadas para obter uma redução da variância, o resultado é combinado através do erro de reconstrução que gera uma pontuação de saída computada pela média sobre diferentes componentes ensemble. As camadas são conectadas randomicamente conforme pode ser visualizada na Figura 8 abaixo.

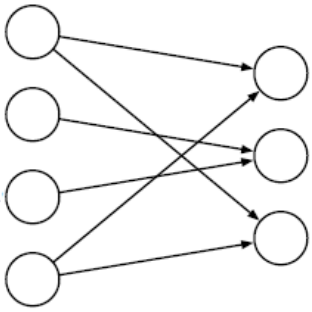

Figura 8: Camadas de rede autoencoder conectadas randomicamente (Chen et al, 2017).

A arquitetura proposta por Chen (Chen et al., 2017) chamada de *RandNet (Randomized Neural Network for Outlier Detection)* utiliza o conceito mencionado acima, tendo encontrado resultados de melhor performance comparados com outras técnicas de detecção de anomalias.

Outro abordagem possível para os Autoencoder Ensembles é um ensemble de pequenos Autoencoders, treinados para reconstruir padrões cujo desempenho melhora incrementalmente com o tempo. A arquitetura funciona com o mapeamento das *features* de uma instância em uma primeira camada da rede, em seguida cada Autoencoder tenta reconstruir as *features* desta instância e é computado o erro de reconstrução utilizando a raiz quadrática do erro médio. O erro de reconstrução então é enviado para a saída do Autoencoder que atua como um mecanismo de votação não linear para o conjunto de autoencoders. Esta rede neural foi chamada de *Kitsune* (Mirsky et al, 2018).

Por fim, uma abordagem mais recente do uso de Autoencoder Ensembles foi dada por Sarvari (Sarvari et al., 2019), onde o algoritmo usa Autoencoders totalmente conectados com o treinamento em sequência. Os dados amostrados para treinar um determinado componente dependem do erro de reconstrução obtidos pelo anterior. Assim quanto mais elevado o valor do erro de resconstrução menor será o peso atribuído ao ponto de dados correspondente. Desta forma, a amostragem ponderada força o Autoencoder a progressivamente aprender sobre o padrão “normal” e distinguir os casos anômalos através de altos valores de erro de reconstrução. Esta abordagem foi denominada *BAE (Boosting-based Autoencoder Ensemble).*

O processo de treinamento de Autoencoders será detalhado no Capítulo 4 desta dissertação, juntamente com as devidas escolhas necessárias em cada etapa para o treinamento desta rede neural.

2.3.6 *Data Preprocessing*

2.3.6.1 *Feature Engineering*

A técnica de Feature Engineering consiste em realizar a extração de *features* de dados brutos e a transformação destas *features* em um formato que seja adequado para o modelo de *machine learning*. É considerada uma etapa crítica no *machine learning* pipeline, pois a seleção de *features* relevantes pode facilmente reduzir a dificuldade da modelagem, permitindo que o modelo aumente a qualidade dos resultados (Zheng & Casari, 2018). Ou seja, a seleção de *features* que melhor representem o problema alvo para o modelo preditivo pode resultar em uma melhoria na acurácia do modelo para dados desconhecidos, não sendo necessário utilizar todas as *features* para melhorar os resultados.

Em um conjunto de dados de treinamento, quanto maior o número de *features*, maior deverá ser o número de exemplos para que o modelo de *machine learning* possa ter um bom desempenho. Quando em um conjunto de dados não é possível reduzir o número de *features*, é observado o fenônemo da maldição da dimensionalidade (Bellman, 1966).

Existem diversas técnicas de Feature Engineering como *filtering, binning, scaling, interaction features, phrase detection, n-grams, bag-of-words, hashing, PCA, one-hot encoding, SIFT, HOG*, entre muitas outras. A utilização de cada uma destas técnicas está associada à abordagem necessária ao conjunto de dados e a utilização dos modelos, bem como a necessidade de aplicação de cada uma destas técnicas.

Em seguida a utilização de técnicas de Feature Engineering, caso estas sejam aplicáveis ao conjunto de dados, é realizada a etapa de seleção das *features* presentes neste conjunto. Estas etapas podem ser independentes do algoritmo de aprendizado (Rawat et al, 2019).

2.3.6.2 *Recursive Feature Elimination (RFE)*

Uma das possíveis técnicas de seleção de *features* é a *Recursive Feature Elimination (RFE).* A RFE funciona removendo recursivamente atributos e construindo um modelo nos atributos que permanecem. Este recurso usa o resultado do modelo para identificar quais atributos (e combinação de atributos) contribuem mais para prever o atributo alvo (Guyon et al, 2002).

Segundo Guyon et al (2002), o procedimento iterativo chamado de RFE é realizado através das seguintes etapas:

i. Treinamento do classificador (otimização dos pesos em relação a função de custo);
ii. Cálculo do critério de classificação para todas as *features*;
iii. Remoção das *features* com a menor pontuação no critério de classificação.

Este procedimento iterativo cria uma instância de eliminação de feature reversa. Para que ocorra uma otimização computacional deste procedimento é mais eficiente remover várias *features* a cada iteração dessa instância, mas isso implica uma possível degradação do desempenho da classificação e que produz uma classificação de *features* em um subconjunto

Se apenas uma feature for removida em cada instância, ocorrerá então uma classificação de *features* em um conjunto e não em um subconjunto. Um fato importante a ser considerado é que as *features* com melhor classificação não são necessariamente as que são individualmente mais relevantes. Ainda é importante salientar que o RFE não tem efeito sobre os métodos de correlação,

pois o critério de classificação é calculado com informações sobre uma única feature (Guyon et al, 2002).

3 Trabalho realizado

Este capítulo apresenta o trabalho realizado com as alternativas que foram consideradas, tendo em consideração o objetivo deste trabalho que foi desenvolver um modelo de detecção de contas anômalas a partir de contas presentes no Twitter e aplicar este modelo ao período relativo das eleições presidenciais ocorridas no Brasil em 2018 e das eleições para assembléia legislativa ocorrida em Portugal no ano de 2019.

Foi realizada a identificação de contas consideradas anômalas e a partir destas contas anômalas foi realizada clusterização do conteúdo presente nos *tweets* publicados por estas contas para verificar a ocorrência de grupos semanticamente parecidos e do contéudo publicado relacionado à discursos políticos.

3.1 Ferramentas utilizadas

Os datasets utilizados para o treinamento e validação dos modelos cresci-2017 e verified-2019, que possuem dados rotulados manualmente com contas de usuários reais e contas falsas ou automatizadas foram obtidos através do repositório online acessível em https://*bot*ometer.osome.iu.edu/*bot*-repository/datasets.html.

O uso destes conjuntos de dados rotulados permite, além do já explicitado em itens anteriores, o treino e a validação a partir de diferentes fontes de dados considerando que são consistentes, possibilitar um bom poder de generalização dos modelos (Yang et al., 2020).

O conjunto de dados utilizados para os testes dos detectores foi retirado através da API desenvolvedora do Twitter no período relativo as eleições presidenciais ocorridas no Brasil no ano de 2018 e nas eleições da assembléia legislativa para Portugal em 2019. Para a coleta dos dados na API foi utilizada a ferramenta s*earchtweets for Python.* Para garantir escalabilidade e resiliência foi utilizado o serviço Colaboratory da Google (Google Colab) com a coleta sendo realizada em streams devido ao elevado tamanho dos dados, para evitar a perda da conexão e da requisição a API do Twitter.

Para o o pré-processamento dos dados foram utilizadas as bibliotecas *Numpy, Seaborn* e *Pandas* e *Matplolib* para a visualização de gráficos. São utilizados os classificadores já implementados na biblioteca scikit-learn *to* Python para os métodos Regressão Logística, *AdaBoost, Random Forest, Support Vector Machine* (SVM) e SVM *one-class*.

O scikit-learn possui uma aplicação da *Recursive Feature Elimination* (RFE) que permite uma prática utilização desta técnica chamada *Recursive Feature Elimination with cross-validadtion* (RFECV).

Para a utilização dos Autoencoders foi empregada a API Keras da plataforma open source Tensorflow 2.0.

O uso do Word2Vec foi provido através da Gensim, uma biblioteca Python open source para processamento de linguagem natural. Também foi utilizada a biblioteca NLTK para o processamento de linguagem.

Para a execução dos algoritmos foi utilizado o Jupyter Lab (*web-based user interface*).

3.2 Pré-processamento dos dados

A etapa de pré-processamento dos dados é de extrema importância no pipeline de *machine learning*. Pré-processar os dados é o processo de preparação, limpeza e estruturação do conjunto de dados a serem usados nas etapas de treinamento, testes e validação dos modelos.

O objetivo de preparar os dados é tratar as *features* que podem ter valores em falta, *outliers*, dados categóricos, valores em escalas contraditórias entre as *features*, etc. Caso os dados não passem pela etapa de pré-processamento os modelos podem performar com resultados insatisfatórios ou mesmo não conseguir gerar resultados.

Foi realizado o pré-processamento nos *datasets* cresci-2017 (Cresci et al., 2017) e verified-2019 (Yang et al, 2020) utilizados para treinamento e validação. Estes conjuntos de dados são disponibilizados pelos autores no *website* chamado "*Bot* Repository" (2020). O conjunto de dados cresci-2017 contém mais de 9.000 contas diferentes rotuladas com as contas reais e anômalas identificadas. O conjunto de dados verified-2019 contém 1.987 contas diferentes verificadas como contas de usuários humanos.

Na análise dos campos do *dataset* cresci-2017 para as contas de usuários reais e anômalas, é possível observar na figura a seguir que existem 43 *features* presentes no conjunto de dados inicial para as contas de usuários.

	id	name	screen_name	statuses_count	followers_count	friends_count	favourites_count	listed_count	url	lang	...
0	1502026416	TASUKU HAYAKAWA	0918Bask	2177	208	332	265	1	NaN	ja	...
1	2492782375	ro_or	1120Roll	2660	330	485	3972	5	NaN	ja	...
2	293212315	bearclaw	14KBBrown	1254	166	177	1185	0	NaN	en	...
3	191839658	pocahontas farida	wadespeters	202968	2248	981	60304	101	http://t.co/rGV0HIJGsu	en	...
4	3020965143	Ms Kathy	191a5bd05da04dc	82	21	79	5	0	NaN	en	...

5 rows × 43 columns

Figura 9: Head do dataset cresci-2017.

No conjunto de dados coletados do Twitter e utilizados para testes, foi criada uma visualização a partir da *feature* ‘created_at’ com o valor do ano, para verificar a data de criação das contas. Esta visualização permite observar que no caso dos dados coletados para o Brasil e Portugal, o maior percentual das contas criadas foi no ano de 2009.

Como o fenômeno de criação de contas anômalas é relativamente recente, podemos observar na figura abaixo para o caso do Brasil que existe um menor número de criação de contas a partir de 2012, mas com um aumento em 2018 onde podem estar presentes contas anômalas.

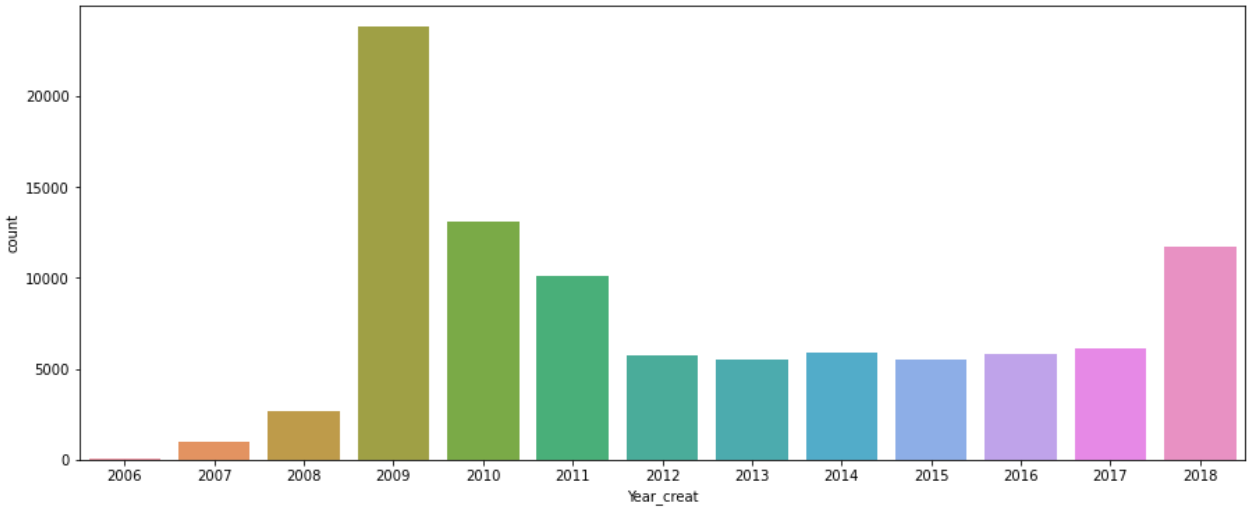

Figura 10: Contas Brasil agrupadas pela data de criação.

Para o caso de Portugal, a figura a seguir apresenta que houve um aumento na criação de contas no ano de 2015 e no ano de 2019.

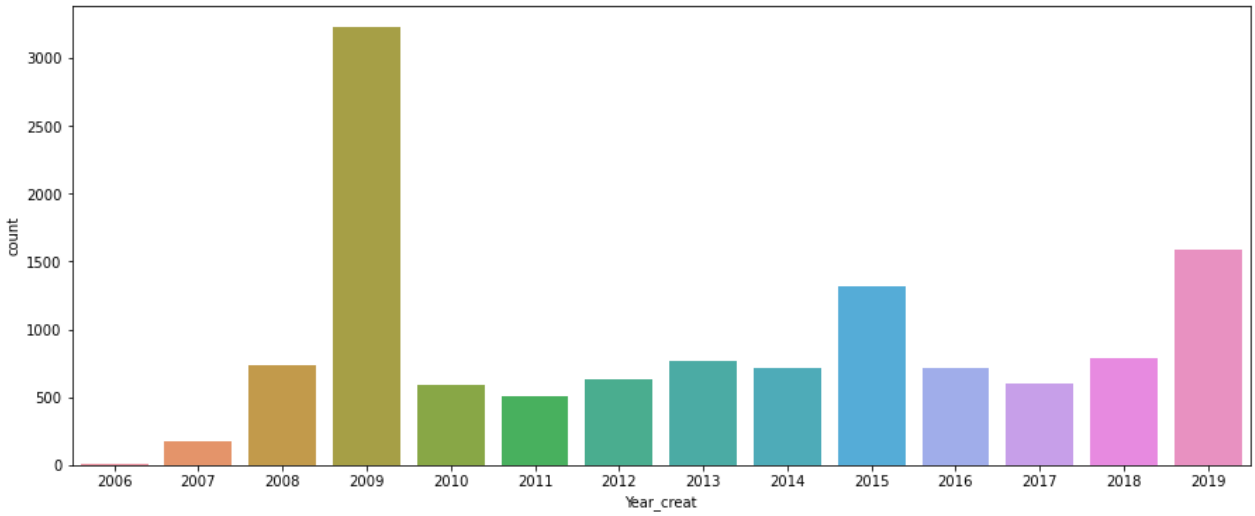

Figura 11: Contas Portugal agrupadas pela data de criação.

3.2.1 *Feature Engineering*

A utilização de técnicas de feature engineering permitem o tratamento dos dados de forma a assegurar que o modelo não será impactado por atributos de diferentes tipos, dados faltantes, categóricos, ou mesmo a não reescala dos dados que pode impactar determinados algoritmos.

O tratamento do conjunto de dados consistiu em selecionar os atributos em comum entre cresci-2017 e verified-2019 concatenando os dois datasets, verificar a presença de valores nulos (NaN), substituir valores NaN com False ou com 0 dependendo do valor da feature e transformar dados numéricos contínuos em formato inteiro.

Outra verificação realizada foi a presença de *outliers* nos pontos de dados rotulados como contas normais, com a substituição destes pela média dos valores da feature correspondente. A retirada de *outliers* é importante pois alguns algoritmos tendem a não performar bem diante de ruídos por serem muito sensíveis a estes, como as redes neurais por exemplo (Chen et al, 2017).

As transformações implementadas para o conjunto de dados cresci-2017 e verified-2019 também foram implementadas para os dados de teste coletados pela API do Twitter, com a exceção do tratamento de *outliers*, visto que não há rótulos (normal ou anômalo) nos dados de teste.

Em seguida para as *features* ‘name’, ‘description’ e ‘screen_name’ dos dados de treinamento e validação, foi obtido o comprimento das palavras para ser utilizado como valor contínuo, pois essas *features* podem ajudar a identificar as contas reais, visto que os usuários reais tendem a ter um comprimento de texto maior no nome e na descrição da conta.

Os dados com valor categórico True e False, que serão utilizados nos modelos, foram transformados para valores contínuos através do encoding dos valores categóricos utilizando para isso a técnica Label Encoder, que transforma os dados False em 0 e True em 1. Os dados categóricos de url foram transformados para o valor 1 quando presentes e 0 quando nulos (NaN).

Foi adicionada a feature ‘Outcome’ considerada como o ‘target’ nos conjunto de dados a ser usado para treinamento e validação (cresci-2017 e verified-2019), assinalada com o valor de 0 para contas normais e 1 para contas anômalas. A Figura abaixo mostra a distribuição das contas rotuladas como normais e anômalas agrupadas pela feature ‘Outcome’.

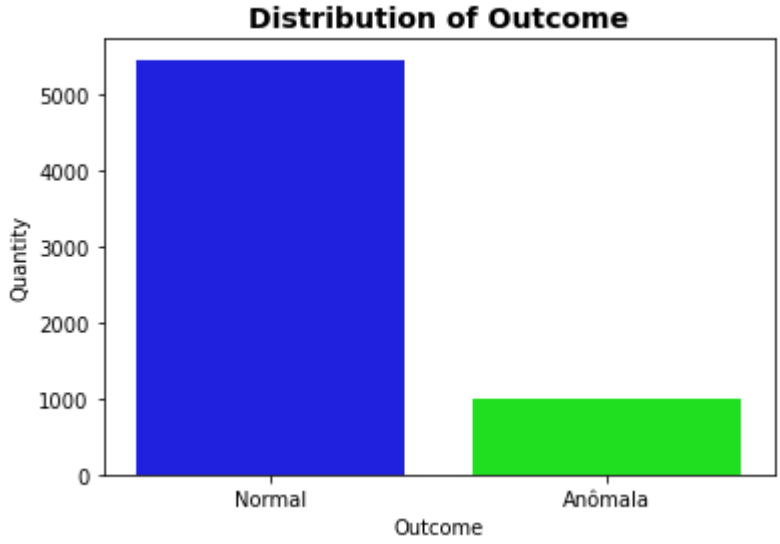

Figura 12: Contas agrupadas pela feature ‘Outcome’.

Alguns algoritmos como SVM, *Logistic Regression*, *Linear Regression*, *K-nearest neighbors* são afetados pela escala dos dados de entrada. Outros algoritmos como *Random Forest*, *Decision Tree*, *AdaBoost* não precisam de dados normalizados pois são classificadores lineares, onde apenas importa a ordem e não a magnitude dos valores.

O processo conhecido como feature scaling, também comumente chamado de feature normalization, é a mudança na escala dos dados e deve ser realizado para cada feature de forma separada (Zheng & Casari, 2018). A reescala dos dados impede que uma *feature* tenha maior impacto para o modelo por apresentar uma escala maior nos dados que outras *features*.

Na utilização do modelo SVM e SVM *one-class* é fundamental a normalização dos dados para que o algoritmo consiga executar bem o treinamento quando os dados não são separáveis linearmente, sendo necessária à utilização de uma função não linear no kernel, como por exemplo, a

utilização de uma função RBF (radial basis function) que usa a distância não euclidiana em sua implementação e é impactada pela escala dos dados. A reescala dos dados melhora a performance do modelo e reduz a instabilidade numérica.

Nos conjuntos de dados para treinamento, validação e testes utilizados nos modelos para a seleção das *features* (Logistic Regression e SVM) e no SVM *one-class* para detecção de anomalias, foi realizada a reescala de algumas *features* com o auxilio dos métodos Normalizer e StandardScaler presentes no scikit-learn. Esta técnica emprega a normalização por linhas das *features* selecionadas e por *features* de forma independente, respectivamente. Para o Logistic Regression foi utilizado o método StandardScaler e para o SVM e SVM *one-class* foi empregado o método Normalizer.

Nos conjuntos de dados para treinamento, validação e testes utilizados para o Autoencoder Ensemble foi realizada a reescala dos dados utilizando o método PowerTransformer do scikit-learn, que transforma os valores em uma distribuição mais normal. Esta é uma etapa obrigatória na preparação dos dados para utilização em redes neurais, pois valores muito assimétricos podem saturar algumas funções de ativação e tornar difícil o trabalho do gradiente descendente.

A reescala dos dados é realizada apenas após a divisão dos dados de treinamento e validação, porque se a reescala dos dados for procedida antes da divisão, os dados de treinamento e validação poderão acabar sendo escalados em torno de um valor médio o qual não corresponde a média dos dados de treinamento e validação. Isso impacta o resultado do modelo no caso de utilizar os dados de validação para verificar a performance em como o modelo responde para dados desconhecidos, o que contradiz o objetivo da divisão em dados de treinamento e validação.

Após o tratamento dos dados para treinamento e validação, estes foram reduzidos a uma dimensão de (6452, 38) com a exclusão das *features* ‘created_at’, ‘datatime’, ‘name’, screen_name’ e ‘description’, ou seja, o dataset ficou com 6452 casos (5461 contas normais e 991 contas anômalas) e 38 *features* para serem utilizadas na etapa de Feature Selection.

O tratamento dos conjuntos de dados coletados do Twitter reduziu os dados a uma dimensão de (97000, 38) no caso do Brasil e (12350, 38) no caso de Portugal. Para a etapa de testes dos modelos SVM *one-class* e Autoencoder Ensembles foram identificadas as contas duplicadas nos dados de testes sendo reduzidas para 41101 contas Brasil e 2726 contas Portugal. Isso se deve ao fato de que algumas contas publicam mais de 1 *tweet* e a coleta dos dados foi realizada através dos *tweets* publicados.

Foi adicionada uma nova feature a estes datasets chamada de ‘verified_blue’. Esta feature recebeu os valores de certificada (True) e não certificada (False) e é proveniente da feature ‘verified’, que é certificada pelo Twitter com o valor de True quando a conta do usuário é de interesse público e autêntica. Esta nova feature permite verificar se as contas certificadas (673 contas Brasil e 59 contas Portugal) foram detectadas como contas normais pelo detector de anomalias SVM *one-class* e Autoencoder Ensemble.

3.2.2 *Feature Selection*

A seleção das principais *features* dos conjuntos de dados para treinamento e validação foi realizada utilizando a técnica RFE (*Recursive Feature Elimination*), com a aplicação do RFECV (*Recursive Feature Elimination* with Cross-Validation), que realiza a eliminação de recurso recursivo com ajuste automático do número de recursos selecionados com validação cruzada, selecionando as *features* mais importantes.

As *features* mais importantes para identificar contas de usuários “normais” são selecionadas através da *Recursive Feature Elimination* (RFE). O ganho da utilização da RFE neste trabalho é devido existir um grande número de *features* presentes no conjunto de dados, permitindo assim selecionar apenas *features* de alta importância que possibilitam melhor performance dos modelos, reduzir complexidade e overfitting, além de não ser necessária uma seleção manual, automatizando o trabalho.

Primeiramente o algoritmo escolhido é treinado no conjunto inicial de *features* e a importância de cada *feature* é obtida através de um atributo ‘coef_’ ou através de um atributo ‘feature_importances_’. Em seguida, as *features* menos importantes são removidas do conjunto atual de *features*. Esse procedimento é repetido recursivamente no conjunto de dados até que o número de *features* a serem selecionadas seja finalmente alcançada a partir do melhor valor da métrica de avaliação definida a ser obtido pelo modelo.

Antes de iniciar a utilização do RFECV para a seleção das principais *features* é necessário verificar a existência de *features* correlacionadas presentes no conjunto de dados. *Features* altamente correlacionadas fornecem a mesma informação para o modelo e podem aumentar o tempo de processamento do RFECV, por isso é importante a eliminação destas *features*.

Para verificar a existência de *features* correlacionadas foi realizado o cálculo de uma matriz de correlação, com a utilização de um coeficiente de correlação de 90%, ou seja, as *features* que tenham mais de 90% de correlação devem ser excluídas do conjunto de dados.

É possível visualizar a saída do cálculo da matriz de correlação abaixo:

```
{'time_zone', 'verified'}
```

As *features* ‘time_zone’ e ‘verified’ apresentaram correlação, sendo eliminada a feature ‘time_zone’, pois essa feature foi descontinuada pelo Twitter e não está mais presente em nenhuma coleta de dados da API.

A feature ‘id’ foi retirada da etapa de seleção das *features* por ser uma informação individual de cada conta, não acrescentando informação para os modelos, mas não foi retirada do dataset final por ser necessária para identificação da conta na etapa da clusterização.

A técnica RFECV requer a seleção de alguns parâmetros em sua utilização como:

- estimator: é o algoritmo a ser utilizado na instância como estimador
- step: Número de *features* a ser removida a cada iteração
- cv: Número da validação cruzada
- scoring: Métrica de avaliação a ser utilizada

Foram escolhidos os modelos de *machine learning* Random Forest, AdaBoost, Logistic Regression e SVM como estimadores no RFECV por terem sido os algoritmos com os melhores resultados observados nos trabalhos encontrados na literatura citados anteriormente.

Em todos as instâncias para os diferentes estimadores a definição dos parâmetros foi de valor 1 para o step, para a validação cruzada foi utilizado o Stratified Kfold de 10 e como métrica de avaliação foi escolhida a acurácia.

Para a execução do RFECV foi criada uma instância com cada estimador, realizada a separação do target ‘Outcome’ e do conjunto de dados em duas variáveis distintas e ao final da execução destas instâncias foi possível obter o número ótimo de *features*, dado como parâmetro de saída. Este número de *features* mais importantes foi obtido pelo valor mais alto de acurácia alcançado na execução de cada instância do RFECV.

Abaixo é possível visualizar as *features* mais importantes a partir do atributo ‘feature_importances_’ para cada estimador utilizado.

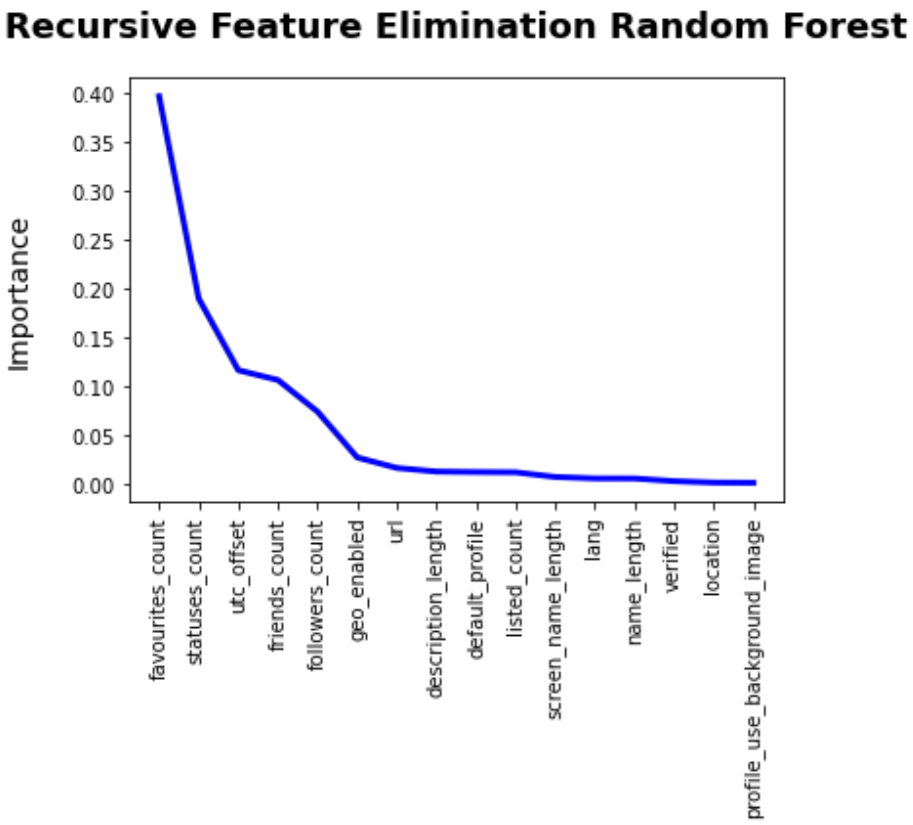

Figura 13: *Features* selecionadas pelo RFECV com Random Forest.

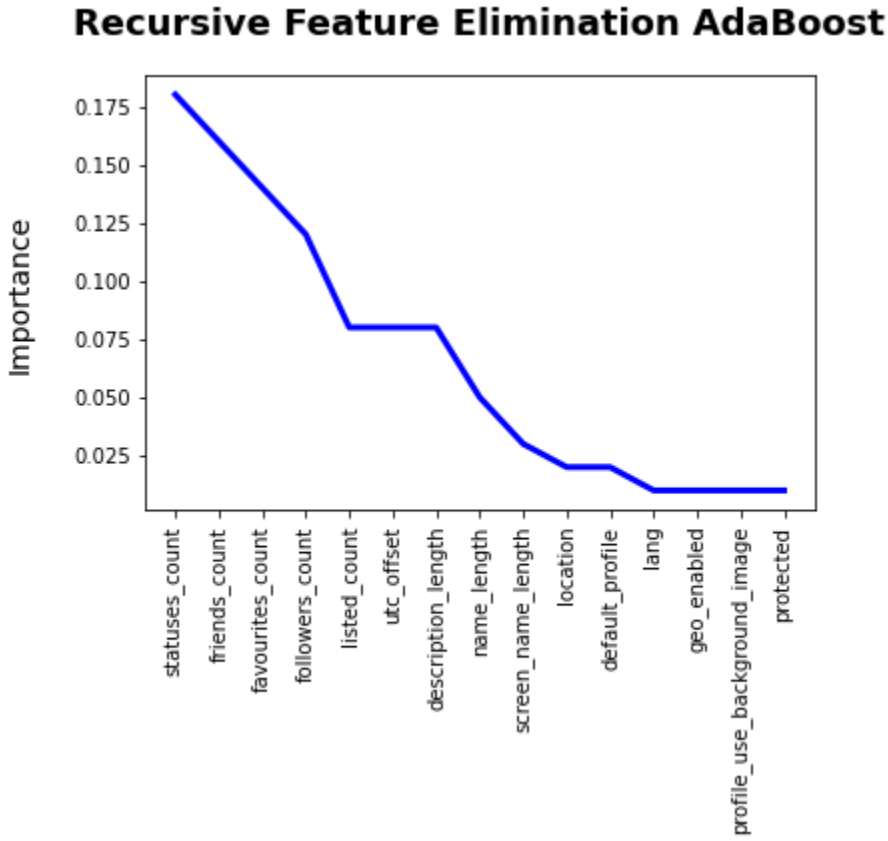

Figura 14: *Features* selecionadas pelo RFECV com AdaBoost.

A utilização de RFECV não tem grande impacto no algoritmo Random Forest, pois a seleção das *features* é realizada pelo próprio algoritmo que procura as melhores *features* entre o subconjunto aleatório das *features* utilizando para isso uma função intrínseca que avalia a importância da feature.

O motivo para utilização desta técnica com o Random Forest, como pode ser visto nos trabalhos de Wang (Wang et al, 2020) e de Patgiri (Patgiri et al, 2018), é devido à dificuldade em encontrar o valor limite das medidas de importância das *features* usadas para determinar as *features* selecionadas na seleção das *features* usando a função intrínseca do Random Forest.

Outra consideração que Wang et al relata é que diferentes amostragens bootstrap também resultam na classificação das medidas de importância das *features* não única em diferentes modelos de Random Forest. Por estes motivos pode ser utilizado o RFECV com o Random Forest.

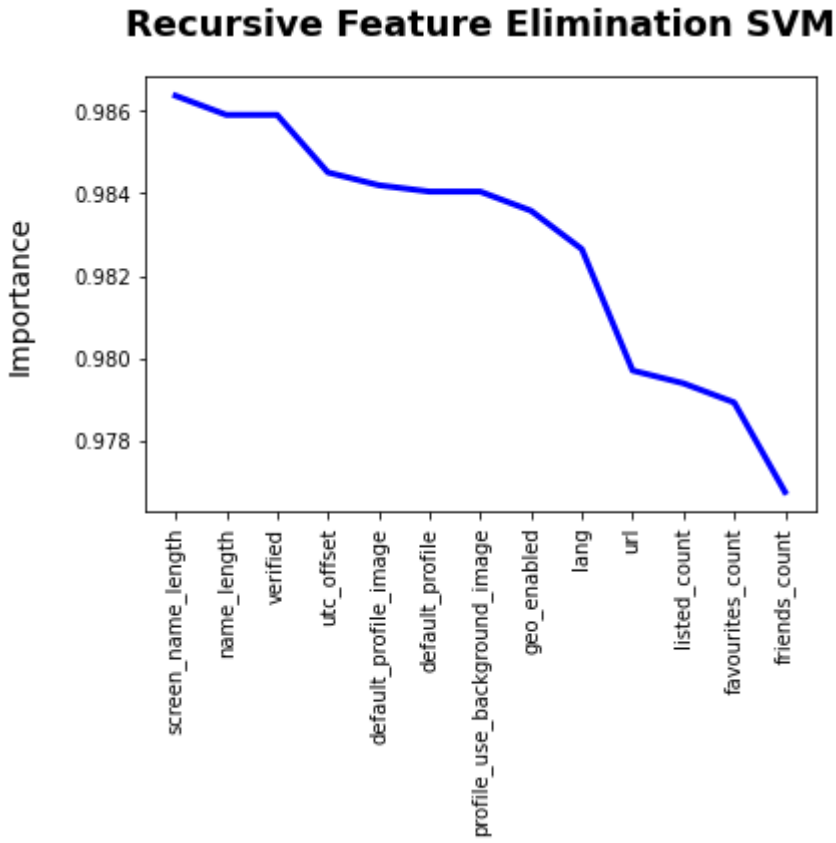

Figura 15: *Features* selecionadas pelo RFECV com SVM.

Para a utilização de RFECV com o SVM é necessário que o kernel definido seja linear ou que na utilização de kernel não linear seja definida a medida de importância das *features* através de uma métrica de aprendizado para o SVM com kernel não linear.

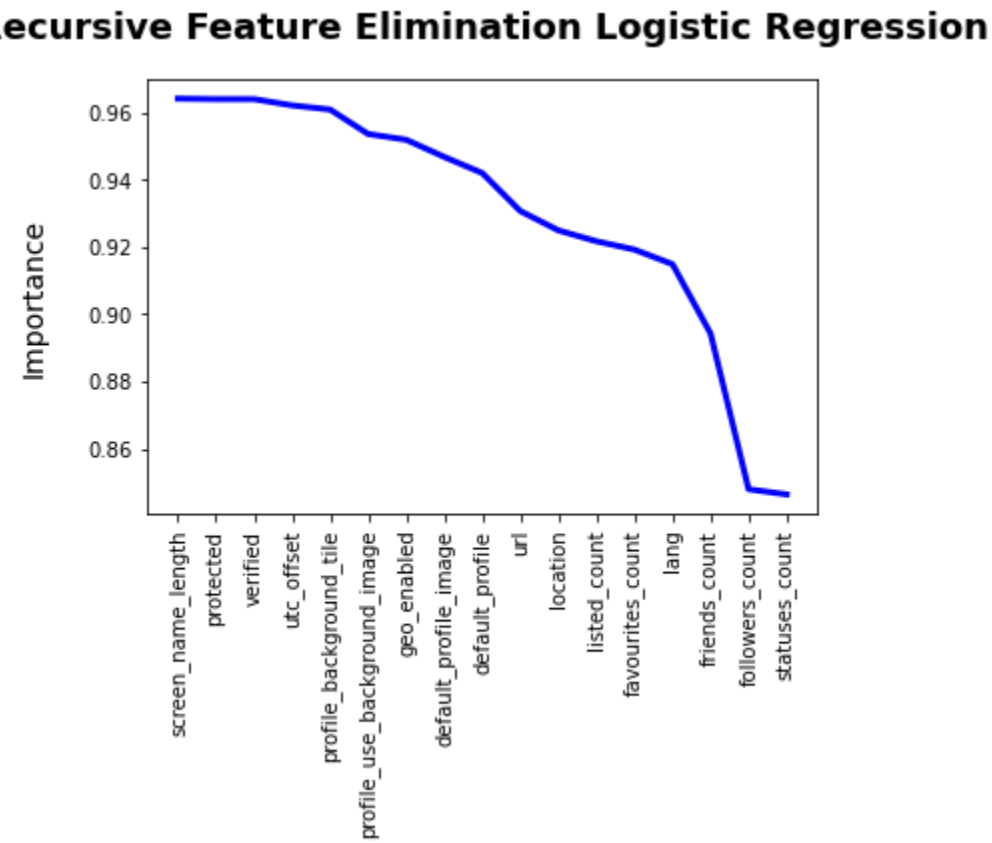

Figura 16: *Features* selecionadas pelo RFECV com Logistic Regression.

Ao final da execução das 4 instâncias foram selecionadas as *features* que obtiveram os maiores valores do atributo ‘feature_importances_’ nos algoritmos (Random Forest, SVM, AdaBoost e Logistic Regression) utilizando o valor de Z-score, que considera a média e o desvio padrão de cada uma das *features*. Na figura 17 é possível visualizar as *features* finais selecionadas através do Z-score.

As *features* 'default_profile_image', 'protected', ‘profile_use_background_image’, ‘geo_enabled’, ‘utc_offset’ e ‘lang’ consideradas importantes pelo RFECV foram eliminadas porquê foram descontinuadas pelo Twitter e retornam valores nulos em solicitações de coleta dos dados através da API. As demais *features* que não foram consideradas importantes pelo RFECV foram eliminadas dos conjuntos de dados para treinamento, validação e testes.

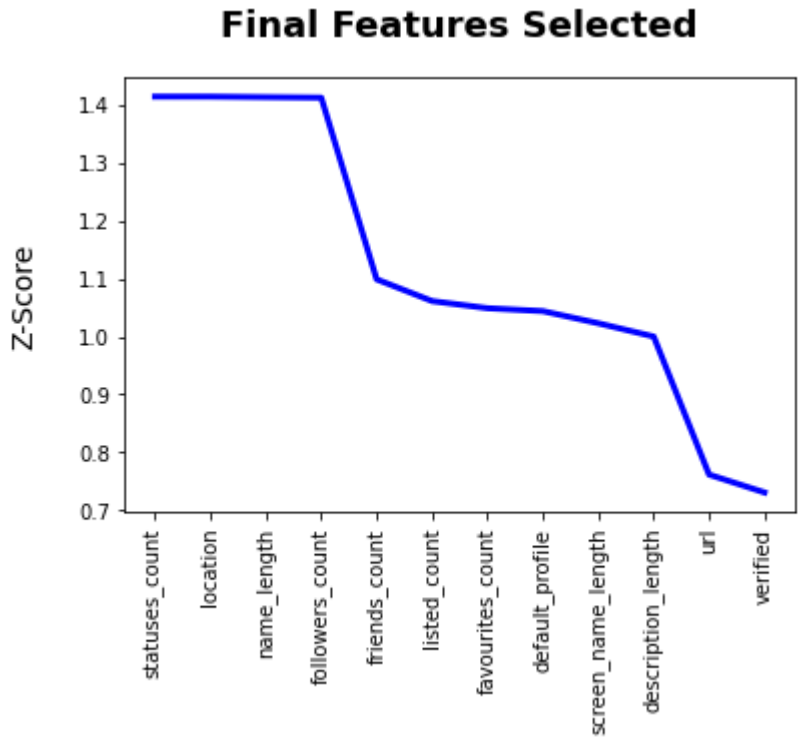

Figura 17: *Features* finais selecionadas utilizando Z-score.

Algumas das *features* mais importantes consideradas já foram mencionadas nos trabalhos referenciados no capítulo 2 como importantes na detecção de contas de usuários reais ou anômalos.

Por exemplo, analisando a importância da feature *favourites_count*, esta significa o número de *tweets* que o usuário marcou como "like". Um número alto deste recurso está relacionado a um comportamento de um usuário real, pois contas falsas e *bots* tendem a não interagir de forma direta com publicações de outros usuários.

A tabela 2 a seguir apresenta as *features* selecionadas e a definição delas a partir do glossário do Twitter (2019e).

Feature	Definição
default_profile	Se igual a True indica que o usuário não alterou o tema ou plano de fundo de seu perfil de usuário
description_length	Comprimento de caracteres que o usuário descreve a conta
favourites_count	O número de *tweets* que este usuário gostou durante o tempo de vida da conta
followers_count	O número de seguidores que esta conta possui atualmente
friends_count	O número de usuários que esta conta está seguindo (também conhecidos como "seguidores")

listed_count	O número de listas públicas das quais este usuário é membro
location	O local definido pelo usuário para o perfil desta conta. Não necessariamente um local, nem analisável por máquina
name_length	Comprimento de caracteres que o usuário utiliza como nome da conta
screen_name_length	Comprimento de caracteres que o usuário utiliza como nome do perfil
statuses_count	O número de *tweets* (incluindo retuítes) emitidos pelo usuário
url	Um URL fornecido pelo usuário em associação com seu perfil
verified	Se igual a True indica que o usuário tem uma conta verificada como autêntica

Tabela 2: *Features* selecionadas e definição.

Ao final desta etapa foi possível visualizar as *features* mais importantes a serem utilizadas no desenvolvimento dos modelos de detecção de anomalias na segunda etapa deste trabalho.

3.3 Métricas de avaliação

As métricas matriz de confusão, Precision, Recall e F1-Score permitem verificar a classificação das contas como normais ou anômalas, a existência de falsos positivos, negativos, a performance dos modelos utilizados e são consideradas métricas de fácil interpretação dos resultados.

O cálculo do Precision é dado através da equação:

(31) $$Precision = \frac{TruePositive}{TruePositive+FalsePositive}$$

Isso significa que mede a porcentagem de acertos do algoritmo em identificar apenas anomalias.

Para o cálculo do Recall este é dado pela equação:

(32) $$Recall = \frac{TruePositive}{TruePositive+FalseNegative}$$

Que mede a porcentagem de acertos em identificar todas as anomalias.

O F1 Score pode então ser definido como:

(33) $$F1\ Score = 2 \times \frac{Precision\ x\ Recall}{Precision+ Recall}$$

Estas definições são importantes pois será considerado o Precision como métrica de avaliação mais importante.

Para o detector de anomalias de contas do Twitter é preferível identificar mais contas normais mesmo que não identifique todas as anomalias, ou seja, evitar ter muitos falsos negativos ao custo da ocorrência de falsos positivos. É melhor identificar as contas normais sendo mesmo contas de usuários reais e nesse caso, deixar de identificar algumas anomalias, detectando mais contas normais e poucas anomalias, pois isso faz com que o detector seja menos questionado em sua capacidade de detectar contas de usuários reais.

Não é necessária a utilização da métrica acurácia nos detectores de anomalia, pois estes são aprendizados não supervisionados e, portanto não possuem o valor alvo na fase de treinamento.

3.4 Detector de anomalias

Para a construção do detector de anomalias foram utilizadas as técnicas SVM *one-class* e Autoencoder Ensembles. Estas técnicas foram escolhidas devido a facilidade na abordagem destes algoritmos em identificar contas anômalas a partir de contas consideradas normais, ou seja, as contas de usuários que não estiveram dentro do conjunto de dados marcados como “normais” são consideradas anômalas.

Como estes algoritmos permitem a identificação de *outliers* ou novidades que possuem atributos nos quais diferem significativamente dos outros recursos do conjunto de dados, foram assim escolhidos para a construção do detector.

Outros motivos para a escolha destas duas técnicas são a boa performance para dados de grande escala, eficiência na memória por utilizar um subconjunto de pontos de treinamento na função decisão (vetores de suporte no SVM *one-class*), redução da complexidade computacional (Autoencoder Ensembles).

A escolha destes algoritmos permite ser ajustada e utilizada para outras eleições por não ser estática, pois os dados de treinamento utilizados para os modelos tem assinalado as contas identificadas de usuários “normais” e estes tendem a não apresentar mudança em sua utilização, ao contrário dos *bots* ou contas falsas.

Estas últimas tendem a serem modificadas para se assimilar ao perfil de comportamento das contas reais. Ou seja, é mais confiável as labels das contas de usuários “normais” dos datasets rotulados. Ainda, o fato de que os *bots* em um determinado dataset são detectados facilmente não implica que classificadores treinados nesse dataset possam detectar *bots* em um outro conjunto de dados (Yang et al., 2020).

3.4.1 SVM *one-class*

O Support Vector Machine *one-class* é um algoritmo de aprendizado não supervisionado que é treinado apenas com os dados das contas normais, aprende as fronteiras destes pontos e é capaz de classificar todos os pontos que se encontram fora dessa fronteira limite, ou seja, os dados anômalos.

Nos datasets de treinamento e validação (cresci-2017 e verified-2019) e testes (dados reais coletados Brasil e Portugal) foi realizada a seleção das *features* mais importantes conforme detalhado na etapa de Feature Selection.

Para a etapa de treinamento e validação foram criadas duas variáveis distintas, uma contendo a feature 'Outcome' e outra com o restante do conjunto dos dados de treinamento e validação.

Em seguida para a seleção do conjunto de dados de treinamento e validação foi utilizada validação cruzada. Esta técnica permite avaliar modelos de *machine learning* treinando vários modelos em subconjuntos dos dados de entrada disponíveis e realizando a avaliação no subconjunto separado e complementar aos dados para que cada ponto de dados possa ser validado. É uma forma de evitar a ocorrência de overfitting. Para isto foi utilizado Stratified Kfold, com valor de k=5, que retorna dados estratificados preservando o percentual de amostras de cada classe.

Após a separação do conjunto de treinamento e validação, os dados foram normalizados com a utilização do método Normalizer do sklearn, como mencionado anteriormente.

Os dados da feature 'Outcome' foram transformados no valor de 0 para 1 no caso das contas normais e de 1 para -1 no caso das contas anômalas, pois o SVM *one-class* faz a predição considerando os valores de 1 para inliers e -1 para *outliers*.

O modelo SVM *one-class* possui alguns parâmetros que são necessários para a execução do algoritmo, que podem impactar na perfomance do modelo. São descritos abaixo os principais parâmetros:

- kernel: Define o tipo de kernel a ser utilizado. Como mencionado no Estado da Arte, transforma os dados através de uma função não linear para projetar os dados em uma dimensão superior. O padrão é a 'rbf' (função de base radial).
- nu: Um limite superior da fração de erros de treinamento e um limite inferior da fração de vetores de suporte e deve estar entre 0 e 1, ou seja, é a proporção de *outliers* esperada nos dados. É considerado um fator importante para o algoritmo, pois muitos modelos de

machine learning não supervisionados necessitam que seja indicado o número de *outliers* ou membros de classe esperados.

- gamma: Coeficiente kernel para kernel ‘rbf, ‘poly’ e ‘sigmoid’. Controla a influência de amostras de treinamento individuais, ou seja, atua como um hiperparâmetro de regularização. Um valor alto de gamma pode implicar em superajuste aos dados de treinamento e perda de capacidade de generalização para novos dados, mas a redução para valores baixos pode acarretar em perda de entendimento dos dados e baixa performance do modelo. É necessária alguma experimentação para encontrar o melhor valor.

A proporção dos dados anômalos no dataset de treinamento e validação é de 15% (991 contas anômalas para 6452 total de contas). O conjunto de dados para o treinamento do SVM *one-class* utilizaram apenas os dados sem a feature ‘Outcome’, pois o treinamento é não supervisionado e o modelo apenas aprenderá as informações sobre os dados das contas normais.

Foi criada uma instância para a execução do modelo com os parâmetros definidos de ‘rbf’ para o kernel, a proporção de *outliers* (15%) para o nu e para o gamma foi utilizado o valor ‘scale’. O valor ‘scale’ é o padrão para o modelo e é definido pela equação:

(34) $$'scale' = \frac{1}{(número\ de\ features \times variância(X))}$$

Onde X é o conjunto de dados sem a presença da *feature* ‘Outcome’.

Após o treinamento do algoritmo, foi realizada a predição no conjunto de validação para avaliar a performance do modelo com um conjunto de dados desconhecidos. As métricas de avaliação do modelo utilizadas foram o F1 Score, Precision, Recall e matriz de confusão. É possível visualizar os resultados na figura 18 e 19 a seguir:

```
F1 score : 0.681
Precision : 0.761
Recall : 0.897
```

Figura 18: Resultados SVM *one-class*

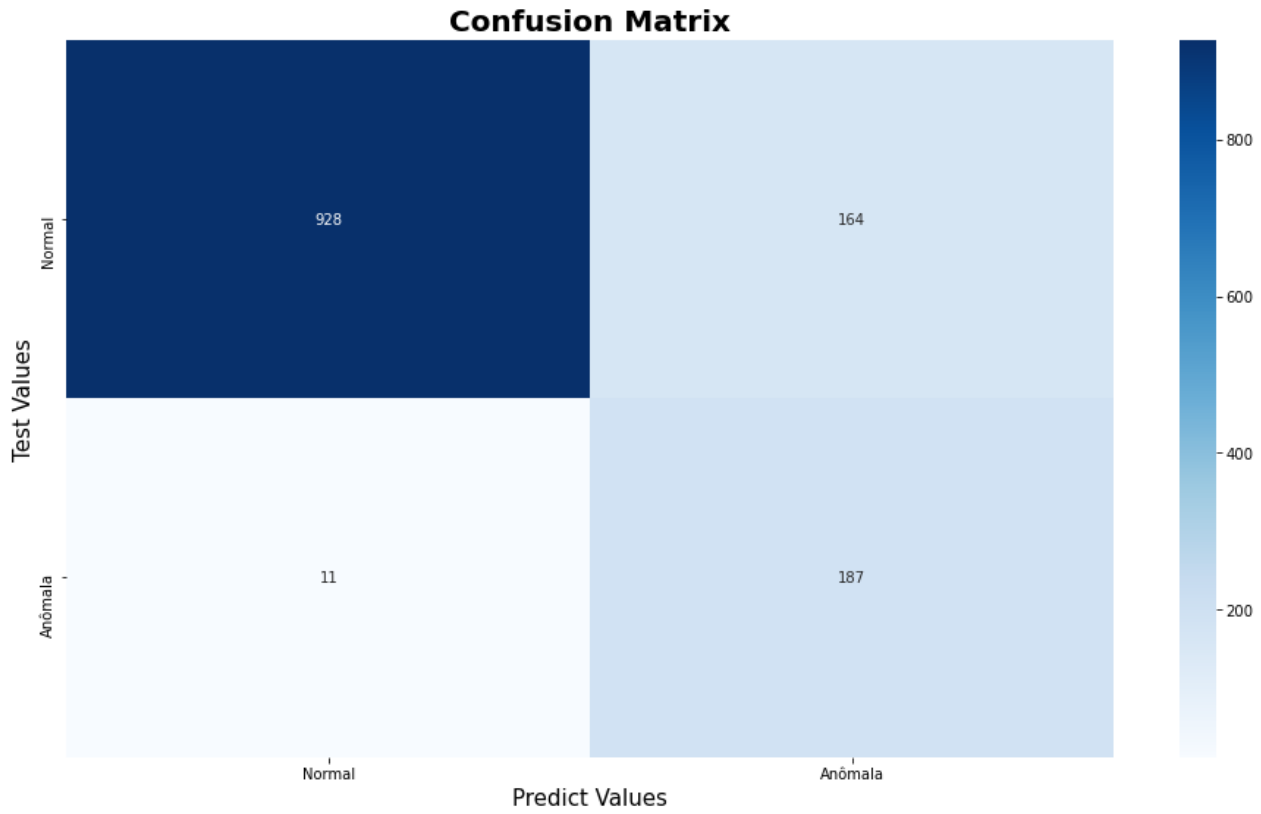

Figura 19: Matriz de confusão SVM *one-class.*

A otimização dos parâmetros, também conhecida como *Fine-tune Hyperparameters* (Géron, 2017), pode ser executada de forma manual ou através de métodos como o Grid Search, Randomized Search, Ensemble methods, etc. A otimização dos hiperparâmetros tem como objetivo encontrar a melhor combinação dos hiperparâmetros presentes nos modelos de *machine learning* que podem impactar diretamente a performance do algoritmo. Não foi possível fazer a otimização dos hiperparâmetros do SVM *one-class* utilizando a técnica Grid Search ou Randomized Search do scikit-learn, pois estas técnicas necessitam que seja definido o parâmetro de regularização do estimador (parâmetro C existente no SVM) e o algoritmo SVM *one-class* não possui este parâmetro.

Como para o SVM *one-class* não existe um método de otimização desenvolvido dentro de pacotes como o scikit-learn, é necessário o desenvolvimento de uma função que execute todas as combinações possíveis dos valores atribuídos aos hiperparâmetros, com a utilização de métricas de avaliação para escolha destes hiperparâmetros.

Os valores dos hiperparâmetros foram escolhidos através da seleção dos melhores resultados de F1 Score, Precision e Recall com a utilização do método ParameterGrid do scikit-learn, que cria um

subset dos hiperparâmetros, e o desenvolvimento de uma função para executar todas as combinações possíveis dos 3 hiperparâmetros (kernel, nu e gamma).

A combinação dos valores para os 3 hiperparâmetros gerou um subset de 252 combinações possíveis de hiperparâmetros sendo aplicados na função desenvolvida que executou 252 instâncias do algoritmo SVM *one-class*. As combinações dos hiperparâmetros com os melhores resultados de F1 Score, Precision e Recall ordenados pelo valor de Precision podem ser visualizadas na Tabela 3:

	param	F1 Score	Precision	Recall
211	{'gamma': 'scale', 'kernel': 'poly', 'nu': 0.01}	0.76506	0.943171	0.817502
223	{'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.01}	0.721003	0.939706	0.787657
212	{'gamma': 'scale', 'kernel': 'poly', 'nu': 0.0173}	0.781341	0.934121	0.833347
224	{'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.0173}	0.73494	0.922352	0.802586
12	{'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.01}	0.407843	0.896935	0.629024
24	{'gamma': 0.01, 'kernel': 'sigmoid', 'nu': 0.01}	0.407843	0.896935	0.629024
48	{'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.01}	0.404669	0.881377	0.628108
60	{'gamma': 0.001, 'kernel': 'sigmoid', 'nu': 0.01}	0.404669	0.881377	0.628108
235	{'gamma': 'scale', 'kernel': 'sigmoid', 'nu': 0.01}	0.301255	0.874173	0.58862
49	{'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.0173}	0.415094	0.851985	0.633394

Tabela 3: Otimização de Hiperparâmetros SVM *one-class.*

A escolha da melhor combinação dos hiperparâmetros para o SVM *one-class* é apresentada na primeira linha (indice 211) da tabela 3. Na figura 20 a seguir é possível ver os resultados na matriz de confusão para o SVM *one-class* com os hiperparâmetros otimizados selecionados.

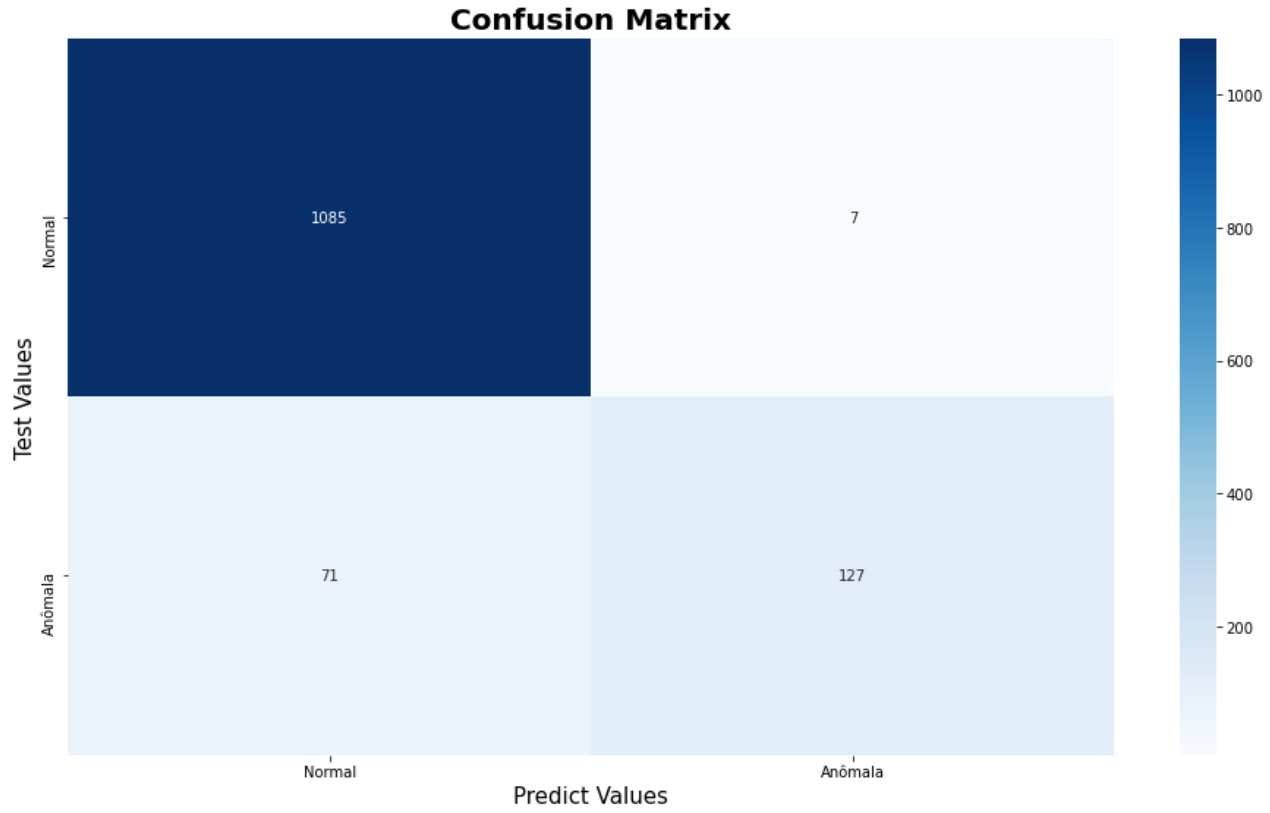

Figura 20: Matriz de confusão SVM *one-class* otimizada.

O relatório da classificação apresenta os valores de Precision, Recall e F1 Score para cada classe prevista no conjunto dos dados de validação. Os resultados são apresentados na Figura 21 a seguir. A predição do modelo SVM *one-class* produz valores de -1 para as contas classificadas como anômalas e 1 para as contas classificadas como normais pelo preditor.

	precision	recall	f1-score	support
-1	0.95	0.64	0.77	198
1	0.94	0.99	0.97	1092
accuracy			0.94	1290
macro avg	0.94	0.82	0.87	1290
weighted avg	0.94	0.94	0.93	1290

Figura 21: Relatório de classificação SVM *one-class* otimizada.

Em outra visualização, é possível observar as contas que foram detectadas corretamente como normais e anômalas pelo preditor a partir do conjunto de dados de validação. Do total de 1290 contas presentes no dataset de validação, 1212 foram preditas corretamente enquanto 78 não foram detectadas de forma correta pelo SVM *one-class*.

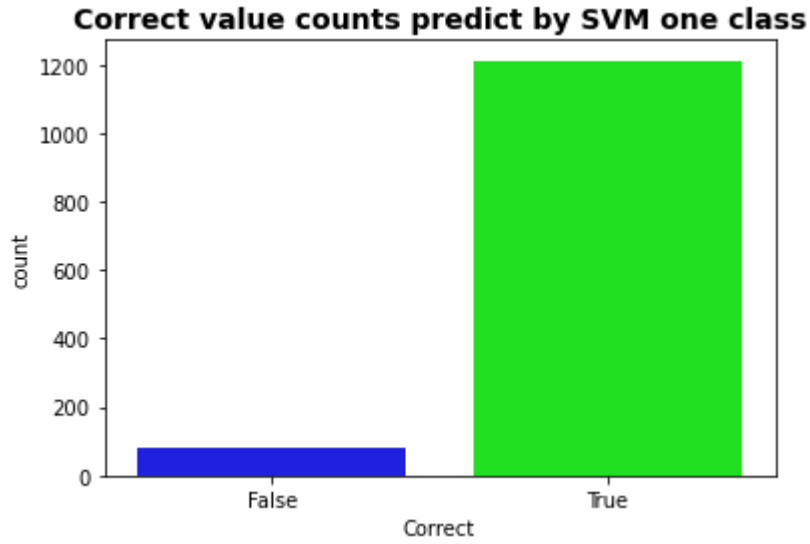

Figura 22: Contas corretas preditas pelo SVM *one-class.*

Em seguida, foi realizada a predição das contas normais e anômalas no dataset real coletado do Twitter, considerado o dataset de testes.

Com a seleção das *features* mais importantes realizada nos conjunto de dados de teste, os dados foram reescalados utilizando o método Normalizer do sklearn. Para executar o detector SVM *one-class*, os dados referentes as eleições do Brasil e em Portugal foram instanciados separadamente no método predict.

As figuras 23 e 24 permitem visualizar as contas do dataset do Twiter para cada país que foram identificadas como normal (valor 0) e anômalas (valor1).

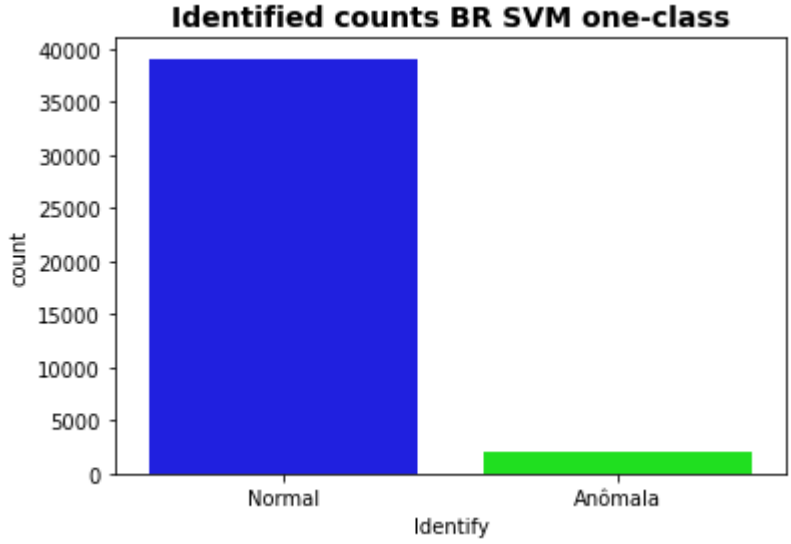

Figura 23: Contas identificadas Brasil pelo SVM *one-class.*

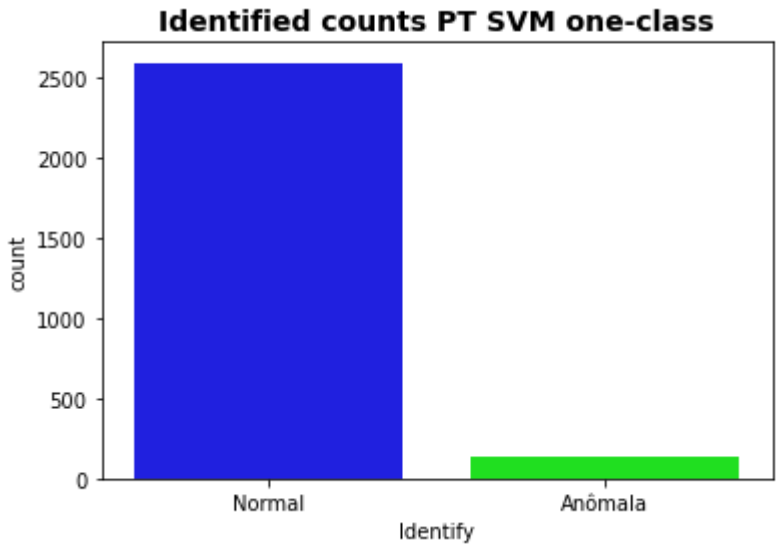

Figura 24: Contas identificadas Portugal pelo SVM *one-class.*

Com as contas classificadas como anômalas, foi salvo o dataset real para as contas coletadas durante as eleições no Brasil e em Portugal contendo os *tweets* referentes a estas contas para serem utilizados na etapa de criação de clusters com o Word2Vec e o Autoencoder.

3.4.2 Autoencoder Ensembles

Os Autoencoders são redes neurais não supervisionadas treinados para ter como saída os dados de entrada utilizando para isso camadas escondidas que aprendem as informações mais importantes dos dados de entrada. Utilizado como detector de anomalias neste trabalho, o treinamento ocorre apenas com os dados das contas normais. Com a rede treinada apenas com os dados normais, as contas anômalas são identificadas a partir do alto valor do erro de reconstrução, por não conseguir reproduzir a mesma representação dos dados.

Igualmente ao realizado no SVM *one-class,* foram selecionadas as 12 *features* nos conjuntos de dados cresci-2017 e verified-2019 para o treinamento e validação da rede, e criadas duas variáveis distintas, uma contendo a feature ‘Outcome’ e outra com o restante do conjunto dos dados de treinamento e validação.

Para a divisão dos dados de treinamento e teste foi utilizada a técnica Train Test Split presente no pacote do scikit-learn. Foi selecionado aleatoriamente 80% dos dados para treinamento e separados os 20% restantes para a validação do modelo. Após a separação do conjunto de treinamento e validação, os dados foram reescalados com a utilização do método PowerTransfomer, como mencionado anteriormente.

O conjunto de dados para o treinamento dos Autoencoders utilizaram apenas os dados sem a feature ‘Outcome’, pois o treinamento é não supervisionado e o modelo apenas aprenderá as informações sobre os dados das contas normais.

Os Autoencoders possuem muitos parâmetros para configuração e que podem acarretar em tipos diferentes de Autoencoders dependendo do valor dos parâmetros escolhidos. A seguir são detalhados os parâmetros utilizados na rede neural para este trabalho.

- units: número de neurônios utilizados nas camadas de entrada, escondida e de saída, pode mudar o tipo de rede.
- epochs: número de épocas do treinamento da rede. Quando definido com apenas 1 época, todo o conjunto de dados de treinamento é passado pela rede apenas uma única vez.
- batch_size: número total de dados de treinamento utilizados em uma iteração. Deve ser definido em função dos resultados do modelo, se for definido altos valores a rede pode ter dificuldade em generalizar para novos dados.
- activation: função de ativação a ser utilizada em cada camada da rede. É possível utilizar as funções linear, sigmóide, tangente hiperbólica, softmax, ReLU, LeakyReLU, Parametric ReLU, Randomized LeakyReLU, entre outros. As camadas escondidas devem ter funções não lineares, pois são essas camadas que performam a não linearidade da rede, sem essas funções todas as camadas seriam unidas em uma única transformação linear. A camada de saída deve ter funções como linear, sigmóide ou softmax para casos de regressão, classificação de duas classes ou classificação multiclasse respectivamente.
- activity_regularizer: regularizador a ser adicionado a função de perda para evitar overfitting. Podem ser utilizados os argumentos ‘L1’ ou o ‘L2’ para diminuir os pesos (w). O regularizador ‘L1’ age penalizando os pesos por um valor constante, minimizando os pesos não nulos. O argumento ‘L2’ penaliza pesos com grandes magnitudes minimizando os pesos em um valor proporcional a w, pois grandes pesos são um dos maiores motivos de overfitting. A diferença entre os dois argumentos é quando um determinado peso possui um valor alto, ‘L2’ reduz o peso muito mais que ‘L1’. No caso do peso ter um valor pequeno, ‘L1’ irá reduzir o peso mais do que ‘L2’.
- earlyStopping: usa o conjunto de validação para checar a ocorrência de overfitting e interrompe o treinamento do modelo antes disso ocorrer. É informado um valor de ‘steps’ que verifica em intervalos regulares deste ‘step’ se o modelo durante o treinamento superajustou aos dados de treinamento, caso isso ocorra, o treinamento é interrompido e informado em qual época de treinamento foi parado o treinamento.

- optimizer: algoritmo de otimização responsável pela velocidade de convergência da rede. O tipo de algoritmo depende do tipo de problema e do tempo disponível para utilização. Para redes neurais é indicado utilizar otimizadores mais rápidos do que o regular gradiente descendente, como Adam, Adagrad, Momentum, Nesterov ou RMSProp. Dentre estes, o Adam é considerado o mais rápido e geralmente o escolhido.
- loss: utilizado para calcular a função de perda (custo) do modelo. Algumas funções calculam o erro entre o valor predito e o valor dos dados de treinamento, como o erro quadrático médio, erro absoluto médio, entre outras.
- metrics: métrica de avaliação utilizada durante o treinamento da rede para avaliação dos resultados. Podem ser utilizados os argumentos ‘accuracy’, 'cosine_similarity', Precision, Recall, entre outros.

Para a construção do Autoencoder Ensembles foram utilizados Autoencoders treinados em sequência com os dados de treinamento amostrados a partir do valor do erro de reconstrução do Autoencoder anterior, ou seja, são utilizados os mesmos dados de treinamento mas com distribuição de probabilidade em selecionar dados alterada em função do valor do erro de reconstrução do Autoencoder anterior.

Desta forma, quando o dado tem um alto valor de reconstrução recebe um peso baixo, fazendo com que o próximo Autoencoder aprenda sobre os dados com maior peso a partir de nova amostragem dos dados. Isso faz com que ao final do treinamento apenas os pontos de dados que apresentaram informação sobre as contas normais tenham tido mais importância e assim contribuindo para o Autoencoder identificar as contas anômalas nos dados de teste.

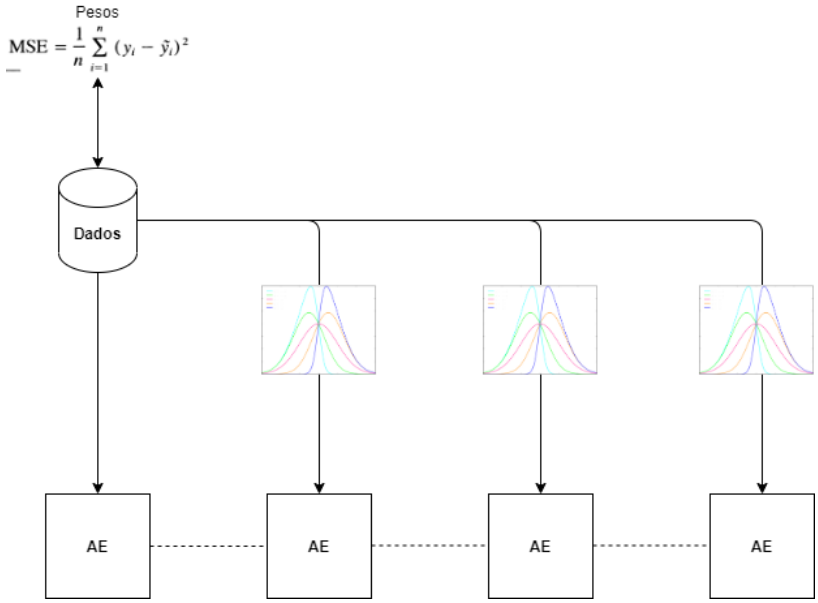

Figura 25: Autoencoder Ensembles desenvolvido.

As etapas descritas acima podem ser consideradas como uma aplicação dos métodos Boosting para a criação de Autoencoder Ensembles, pois conforme Aggarwal (2017), no contexto do problema de classificação, métodos Boosting podem ser considerados exemplos de ensembles sequenciais.

A arquitetura da rede é composta pela camada de entrada com a dimensão dos dados de treinamento (12 neurônios), 5 camadas escondidas com 16, 8, 4, 8, 16 neurônios e a camada de saída com 12 neurônios. Esta arquitetura foi definida após o ajuste em vários números de camadas escondidas, de unidades de neurônios e de funções de ativação, sendo a que performou os melhores resultados. Como foram utilizadas formas de regularização, pode-se definir mais neurônios na camada escondida que na camada de entrada. Os parâmetros utilizados na rede foram:

- epochs: 600
- batch_size: 64
- activation: tanh (primeira camada de escondida), relu (restantes camadas escondidas)
- activity_regularizer: ‘L1’(com taxa de aprendizado de 10e-5)
- earlyStopping: 50 ‘steps’
- optimizer: Adam
- loss: erro quadrático médio
- metrics: ‘accuracy’

O treinamento dos Autoencoder Ensembles convergiu com 219 épocas sendo interrompido pelo earlyStopping. Foram realizados 3 treinamentos considerando o método descrito anteriormente. Após o treinamento da rede, foi realizada a predição no conjunto de validação para avaliar a performance dos Autoencoders. As métricas de avaliação do modelo utilizadas foram o F1 Score, Precision, Recall e matriz de confusão. É possível visualizar os resultados na figura 26 e 27 a seguir:

	precision	recall	f1-score	support
0	0.90	0.98	0.94	1076
1	0.79	0.43	0.56	215
accuracy			0.89	1291
macro avg	0.85	0.71	0.75	1291
weighted avg	0.88	0.89	0.87	1291

Figura 26: Relatório da classificação Autoencoder Ensembles.

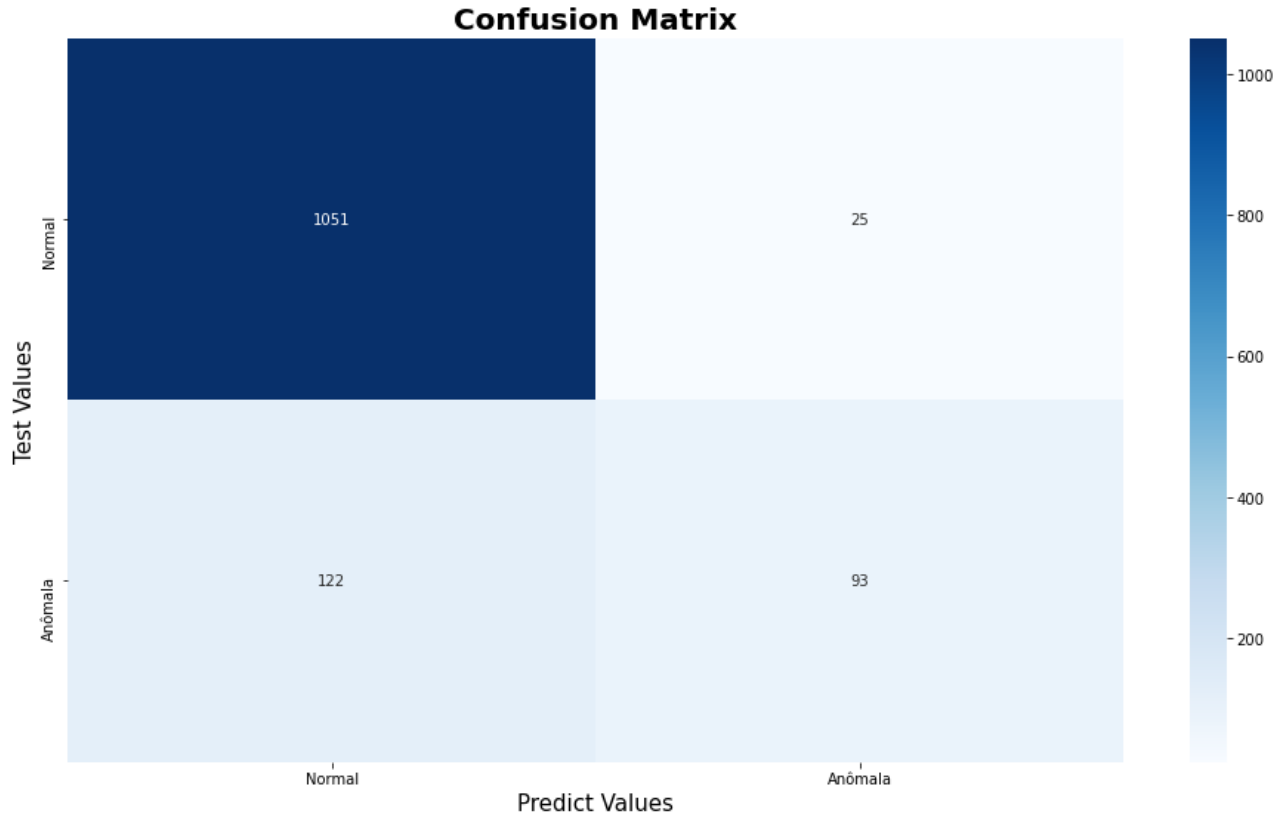

Figura 27: Matriz de confusão Autoencoder Ensembles.

Para o conjunto de treinamento e validação é possível visualizar a perda em função das épocas de treinamento na figura 28, onde a perda do conjunto de validação é muito maior que no conjunto de treinamento, indicando a presença das contas anômalas neste conjunto.

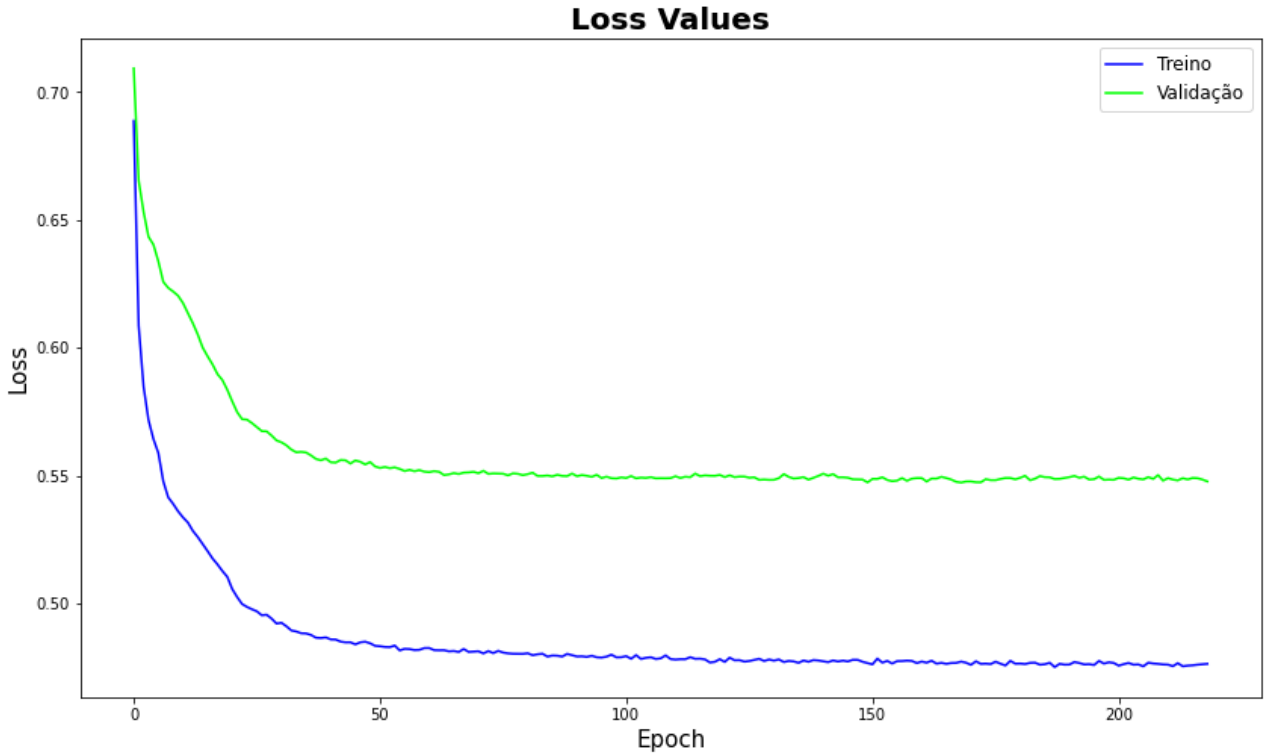

Figura 28: Perda x Época de treinamento.

Outra visualização importante dada pela figura 29 é o erro de reconstrução apresentado para as contas normais e as contas anômalas no conjunto de validação, é possível verificar que o Autoencoder aprendeu a identificar as contas anômalas pois estas tem um erro de reconstrução maior que as contas normais.

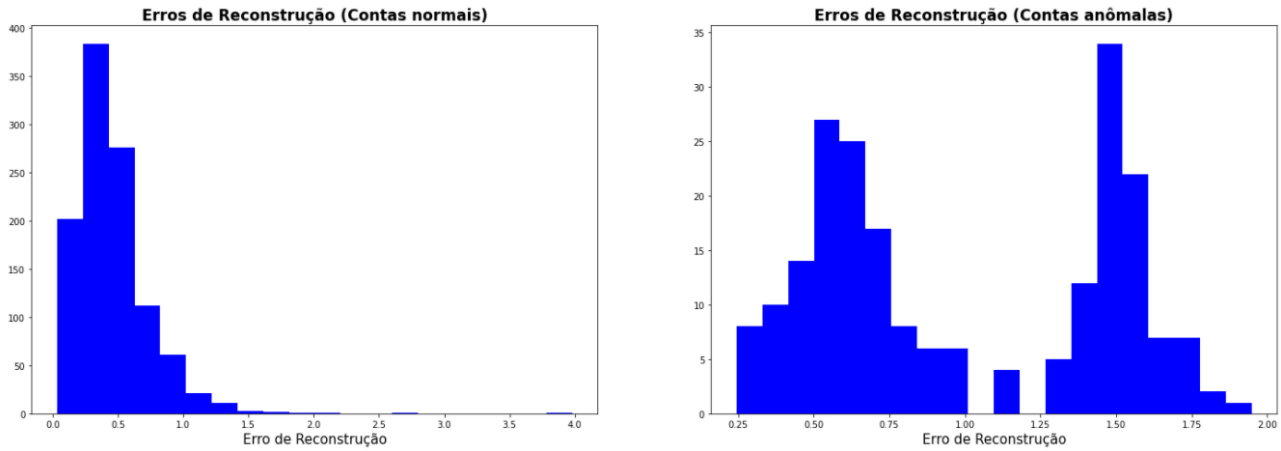

Figura 29: Erro de reconstrução das contas normais e anômalas.

Utilizando o valor de erro de reconstrução considerado como o ponto de corte na separação das contas normais e anômalas, é possível verificar a separação destas contas no conjunto de validação dos dados na figura 30, sendo visível que muitas contas anômalas foram identificadas como contas normais, resultado da escolha de um Precision maior que o Recall.

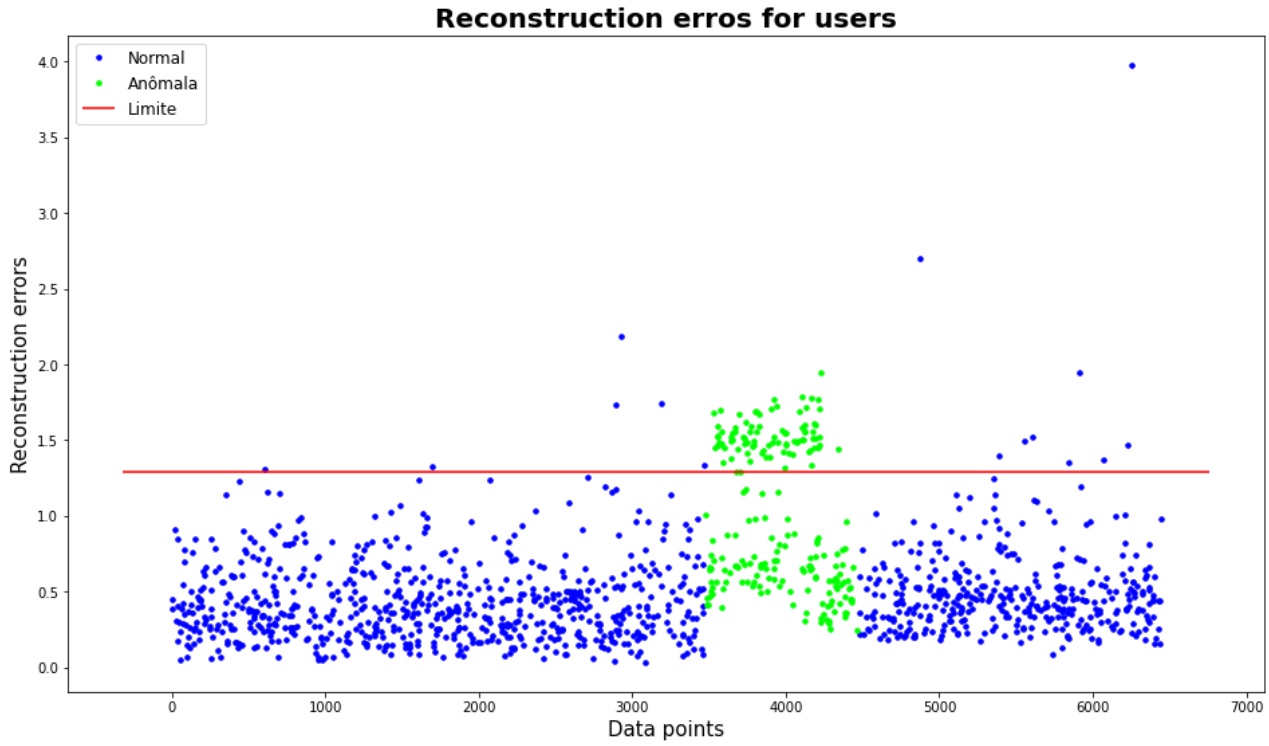

Figura 30: Separação das contas a partir do valor do erro de reconstrução.

Em outra visualização, é possível observar as contas que foram detectadas corretamente como normais e anômalas pelo preditor a partir do conjunto de dados de validação. Do total de 1291 contas presentes no dataset de validação, 1145 foram preditas corretamente enquanto 146 não foram detectadas de forma correta pelo Autoencoder.

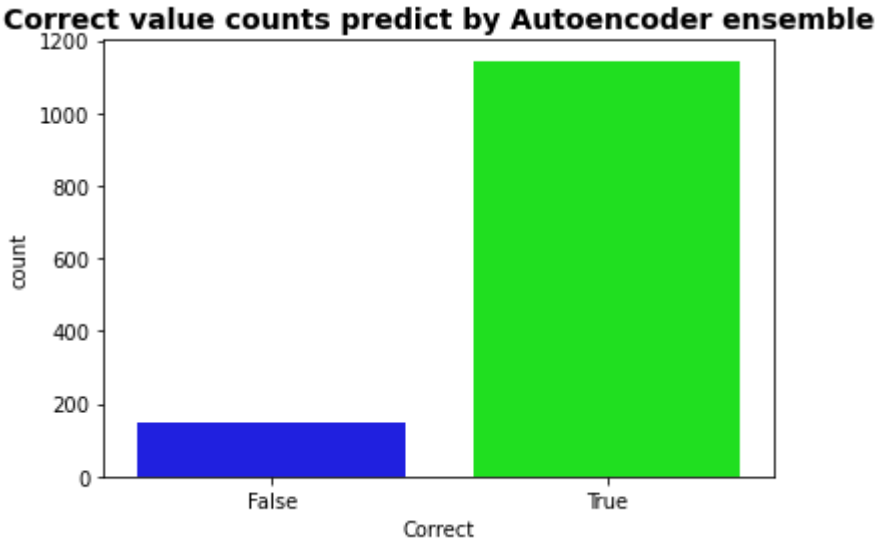

Figura 31: Contas corretas preditas pelo Autoencoder.

Realizar o *Fine-tune Hyperparameters* (Géron, 2017) em redes neurais é considerado uma tarefa difícil devido a elevada quantidade de hiperparâmetros existentes na rede que acarreta a necessidade de tempo disponível.

Para tentar melhorar a perfomance da rede, foi utilizada a técnica Dropout. Especificamente, o Dropout treina o conjunto que consiste em todas as sub-redes que podem ser formadas pela remoção de unidades sem saída de uma rede de base subjacente (Goodfellow et al, 2016). O Dropout pode ser considerado como método de treinamento de um ensemble de modelos em paralelo.

Considerando o Dropout como um dos parâmetros da rede neural, abaixo é detalhado seu funcionamento.

- dropout: outra técnica de regularização para evitar overfitting, o dropout "desliga" randomicamente neurônios da camada escondida a cada iteração do treinamento com uma probabilidade 'p'. Em termos práticos, em cada época do treinamento, o dropout fixa alguns pesos em 0, inativando estes neurônios. Dessa forma, a rede é forçada a redistribuir os cálculos e impede de criar dependência dos recursos de um subconjunto. O valor do dropout é definido através do argumento 'rate'.

Para a utilização do Dropout nos Autoencoder Ensembles foram utilizados diversos valores na faixa de 0.2 a 0.5 nas iterações da rede neural sendo o melhor resultado obtido com o valor de 0.28.

A utilização do Dropout melhorou a Precision para as contas anômalas mas diminuiu o valor do Recall para estas mesmas contas e o Precision para as contas normais, conforme apresentado na figura 32.

	precision	recall	f1-score	support
0	0.89	0.99	0.94	1076
1	0.86	0.41	0.56	215
accuracy			0.89	1291
macro avg	0.87	0.70	0.75	1291
weighted avg	0.89	0.89	0.87	1291

Figura 32: Relatório da classificação Autoencoder Ensembles com Dropout.

Conforme apresentado anteriormente, o objetivo é ter um valor de Precision mais alto que Recall, para garantir que todas as contas que sejam identificadas como normais sejam mesmo

normais, mesmo que algumas contas anômalas também sejam identificdas como normais. Desta forma foi utilizada a rede neural considerando a aplicação do Dropout.

Em seguida, foi realizada a predição das contas normais e anômalas no conjunto de dados de testes coletado do Twitter. Após a seleção das *features* mais importantes realizada nestes dados, foi aplicada a reescala utilizando o método PowerTransforme do sklearn. Para executar o detector Autoencoder Ensembles, os dados referentes as eleições do Brasil e em Portugal foram instanciados separadamente no método predict.

As figuras 33 e 34 permitem visualizar as contas do dataset do Twiter para cada país que foram identificadas como normal (valor 0) e anômalas (valor1).

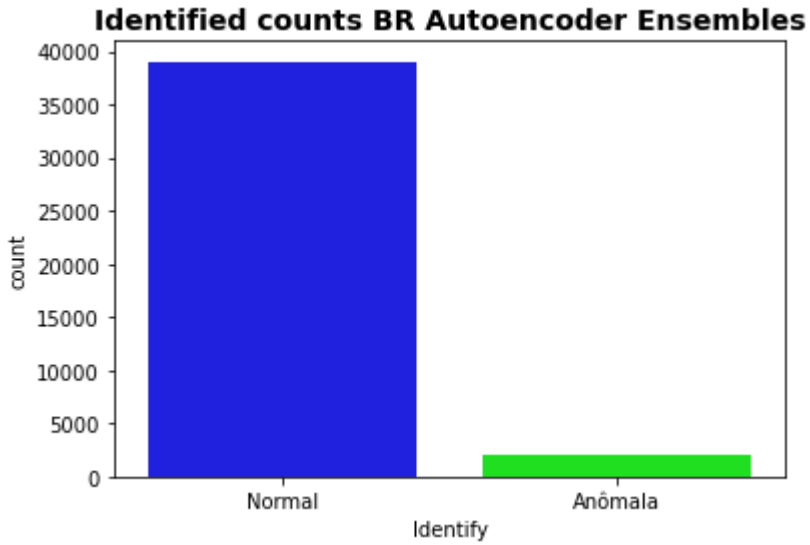

Figura 33: Contas identificadas Brasil pelo Autoencoder Ensembles.

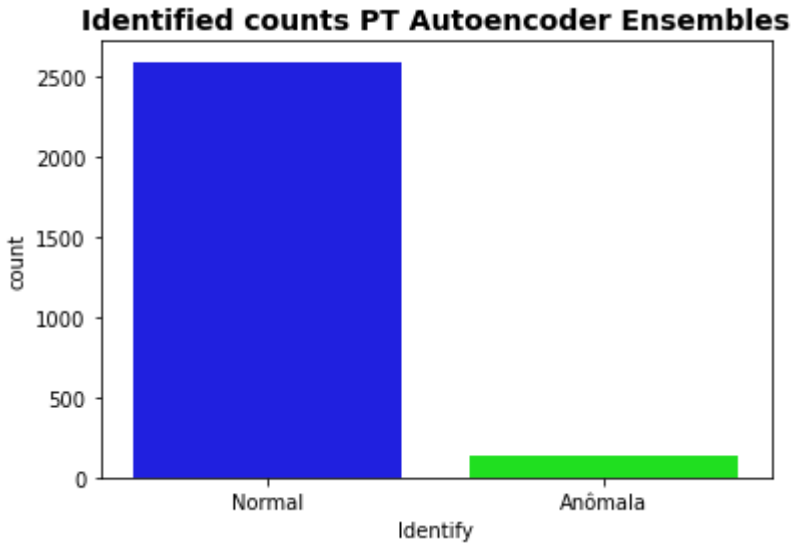

Figura 34: Contas identificadas Portugal pelo Autoencoder Ensembles.

Com as contas classificadas como anômalas, foi salvo o dataset real para as contas coletadas durante as eleições no Brasil e em Portugal contendo os *tweets* referentes a estas contas para serem utilizados na etapa de criação de clusters com o Word2Vec e o Autoencoder.

3.5 Clusterização

A utilização de embedding permitiu a visualização do conteúdo publicado pelas contas identificadas como anômalas e possibilitou a utilização de Autoencoders para redução de dimensionalidade a fim de verificar possível homogeneidade no conteúdo das publicações dos *tweets* e descobrir se existiam grupos semanticamente parecidos.

3.5.1 Word Embedding

A técnica *Word Embedding (Word2Vec*) utilizou o conteúdo dos *tweets* publicados pelas contas anômalas sendo representados em vetores contínuos de tamanho fixo, ou seja, representou os *tweets* das contas detectadas como anômalas com o numero de *features* reduzidos em um vetor.

O Word Embedding foi escolhido devido à utilização de incorporadores embedding em dados com várias palavras e textos apresentar alta eficiência computacional e o Word2Vec ser uma forma especialmente eficiente de treinar *Word Embeddings*.

Foi selecionado o conjunto de *tweets* das contas anômalas detectadas pelo SVM *one-class* para o Brasil e o conjunto de *tweets* das contas anômalas detectadas pelo Autoencoder Ensembles para Portugal.

Após análise dos dados salvos no algoritmo SVM *one-class* e do Autoencoder Ensembles, foi possível inferir que as contas detectadas pelo SVM *one-class* tiveram melhores resultados para as contas do Brasil, pois detectaram 4205 *tweets* de contas anômalas não verificadas pelo Twitter contra 3903 *tweets* pelo Autoencoder Ensembles nos dados Brasil. Para o caso de Portugal o Autoencoder Ensembles teve melhor resultado com 530 *tweets* de contas anômalas não verificadas pelo Twitter contra 304 *tweets* do SVM *one-class*.

A quantidade de *tweets* é diferente da quantidade de contas anômalas detectadas, pois uma conta pode ter publicado mais de 1 *tweet* e por isso existe uma diferença entre a quantidade de contas detectadas anômalas e a quantidade de *tweets* a serem analisados nesta etapa.

A dimensão dos dados para o Brasil é de 4205 *tweets* e para Portugal de 530 *tweets*. Primeiro foi necessário realizar o pré-processamento dos dados para converter os *tweets* em sentenças individuais para cada conjunto de dados separadamente. O tratamento dos dados é listado a seguir:

1. Utilizado regex (Módulo Regular Expressions do Python) para remoção dos @usuários;
2. Removidos números, símbolos e caracteres especiais com utilização de caracteres latinos do regex:
3. Removidas as urls presentes;
4. Removidas as ‘stop words’;
5. Realizado o tokenizer (texto separado em sentenças);
6. Realizado stemming (reduzir a palavra para sua raiz)

Após a limpeza dos dados, foi utilizada a técnica WordCloud que gera uma visualização onde as palavras mais frequentes aparecem em um tamanho maior que as palavras menos frequentes, conforme as figuras 35 e 36 a seguir.

Figura 35: WordCloud para as contas anômalas Brasil.

Figura 36: WordCloud para as contas anômalas Portugal.

Também foi criada uma visualização com as 20 ‘hashtags’ mais utilizadas no conjunto de dados apresentada nas figuras 37 e 38.

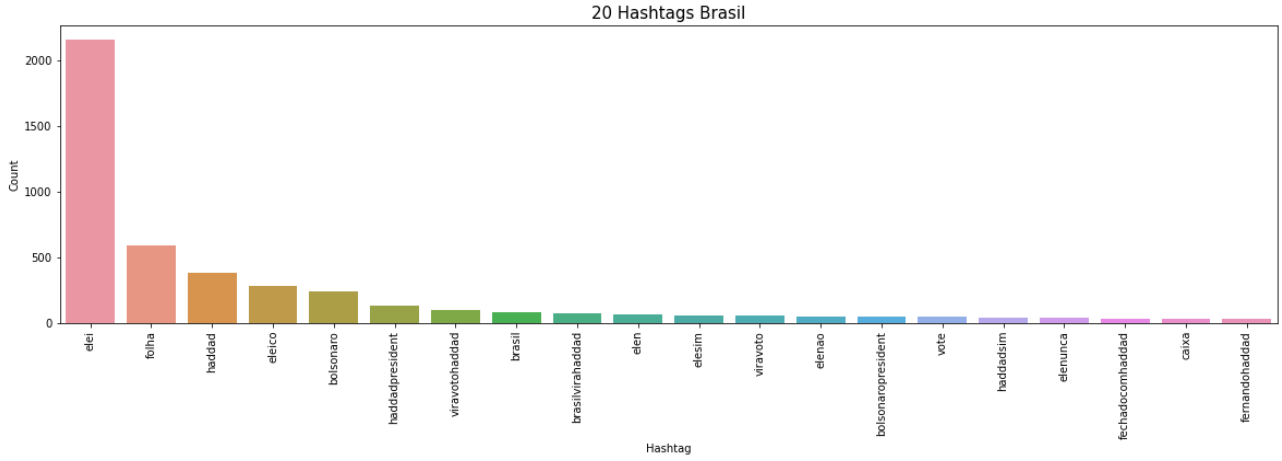

Figura 37: 20 ‘hashtags’ mais utilizadas Brasil.

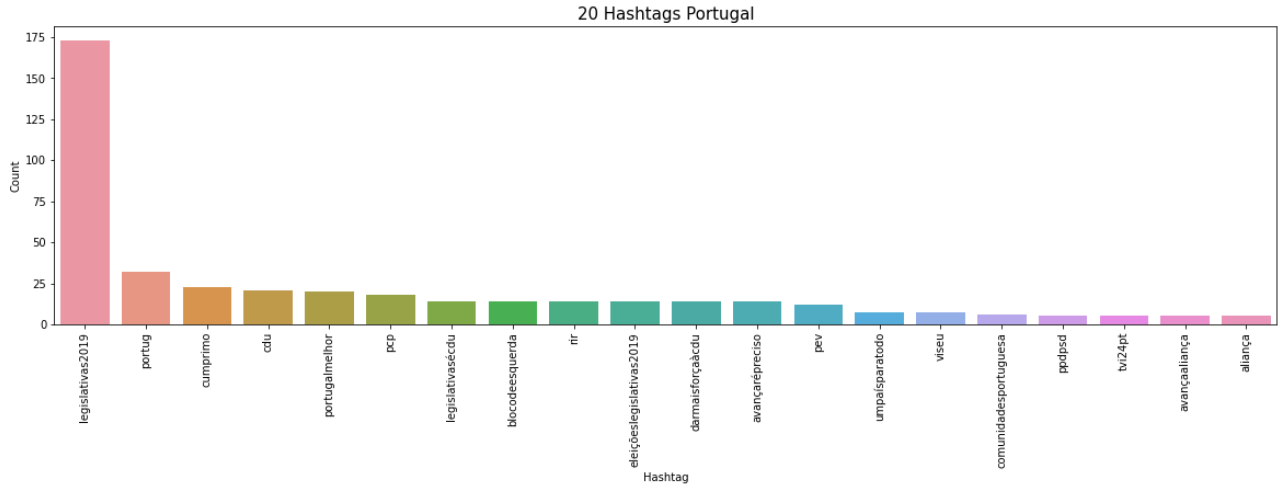

Figura 38: 20 ‘hashtags’ mais utilizadas Portugal.

O Word2Vec é uma rede neural linear com apenas uma camada, sendo simples e rápida no treinamento. Os parâmetros escolhidos para utilização desta rede neural foram:

- size: número de dimensões do embedding, valores maiores requerem mais dados de treinamento, mas podem levar a modelos mais precisos. Foi utilizado 200
- window: Distância máxima entre a palavra atual e a prevista em uma frase. Utilizado o valor de 5
- min_count: número de ocorrência para a palavra ser considerada, ignora todas as palavras com frequência total inferior a esta. Valor escolhido foi 10
- sg: treinamento do algoritmo, 1 para skip-gram model (dataset de menor dimensão), 0 para cbow (dataset de maior dimensão). Escolhido skip-gram pois os dois conjuntos de dados são de baixa dimensão
- negative: se > 0, a amostragem negativa será usada, o inteiro para negativo especifica quantas "palavras de ruído" devem ser desenhadas (geralmente entre 5-20). Se definido como 0, nenhuma amostra negativa é usada. Setado o valor de 5
- workers: número de partições durante o treinamento. Utilizada 32 partições
- seed: seed para o gerador de número aleatório, utilizado 34

O treinamento desta rede neural foi rápido e produziu como saída words embedding, em que cada conjunto de dados gerou um vocabulário de 691 palavras (Brasil) e 71 (Portugal) em um espaço dimensional de 200 dimensões.

É possível visualizar a similaridade entre as palavras através do vocabulário criado, como apresentado nas figuras 39 e 40 para as palavras ‘bolsonaro’ e ‘costa’. O Word2Vec aprende

vetores para cada palavra única e utiliza similaridade de cosseno para descobrir os vetores (palavras) mais semelhantes.

```
model.wv.most_similar('bolsonaro')
```

```
[('arma', 0.302173912525177),
 ('fernando', 0.29236918687820435),
 ('preconceito', 0.25258612632751465),
 ('rei', 0.2436305582523346),
 ('discurso', 0.24152937531471252),
 ('tortura', 0.23846624791622162),
 ('mero', 0.23580671846866608),
 ('twitter', 0.22948986291885376),
 ('nando', 0.2263772338628769),
 ('#cbnaselei', 0.22460561990737915)]
```

Figura 39: Similaridade para a palavra ‘bolsonaro’.

```
model.wv.most_similar(positive="costa")
```

```
[('quem', 0.9995837211608887),
 ('antónio', 0.9995335340499878),
 ('debat', 0.9995319247245789),
 ('sousa', 0.9994972348213196),
 ('pessoa', 0.9994877576828003),
 ('para', 0.9994866847991943),
 ('jerónimo', 0.9994858503341675),
 ('grand', 0.9994733333587646),
 ('partido', 0.9994670152664185),
 ('pelo', 0.9994632005691528)]
```

Figura 40: Similaridade para a palavra ‘costa’.

Foi criada uma função para representação vetorial do *tweet*, visto que o Word2Vec gera vetores de palavras e é necessário representar o *tweet* completo para ser utilizado no Autoencoder, não apenas as palavras. A função em Python gerou um vetor para cada *tweet* a partir da posição média dos vetores das palavras contidas neste *tweet*, com o comprimento do vetor sendo o mesmo da dimensão dos dados já definidos no Word2Vec.

A representação do *tweet* em um vetor a ser utilizado no Autoencoder para reduzir a dimensão do mesmo tem como objetivo tentar responder ao questionamento se existem tipos diferentes de *tweets* publicados pelas contas anômalas. Assumindo que as palavras contidas em um *tweet* pertencem ao mesmo tópico, a média dos vetores embedding pode informar se existe diferença entre os tipos de *tweet*.

3.5.2 Autoencoder – redução da dimensionalidade

Os Autoencoders podem ser utilizados na criação de clusters devido a sua capacidade em reduzir dimensão de dados não lineares. Como explicado anteriormente, esta rede neural possui grande capacidade em agrupar dados, executar transformações não lineares com função de ativação não linear em várias camadas escondidas da rede e demonstrar mérito quando os dados são de natureza complexa e não linear.

A rede neural para os dados Brasil foi elaborada com uma arquitetura de 5 camadas escondidas de 100, 50, 2 (*bot*tleneck), 50, 100 neurônios e a camada de entrada e saída com 200 neurônios. Os parâmetros utilizados na rede foram:

- epochs: 3000
- batch_size: 64
- activation: linear (camadas escondidas) e sigmóide (camada escondida *bot*tleneck)
- optimizer: sgd momentum
- loss: erro quadrático médio

A rede neural para os dados Portugal utilizou os mesmos parâmetros citados acima mas com uma arquitetura de 1 camada escondida de 2 neurônios. Foi realizada esta escolha devido ao conjunto de dados ser menor.

Nesta rede neural não é utilizada a etapa de validação porquê o autoencoder foi treinado apenas para reduzir os conjuntos de dados com objetivo de representar os dados e criar cluster, e como não será utilizado este Autoencoder em outro conjunto de dados, apenas neste, não é necessária a etapa de validação.

O treinamento da rede foi realizado em 3000 épocas (Brasil) e 3000 épocas (Portugal). Para a redução de dimensionalidade com Autoencoder é utilizada apenas as camadas ‘encoded’, pois o objetivo é reduzir a dimensão dos dados de entrada para a dimensão definida na camada escondida *bottleneck*. Após o treinamento foi realizada a predição dos dados a partir do modelo com as dimensões dos dados de entrada e da camada *bottleneck*, que reduziu os dados de uma dimensão de 200 para 2 dimensões, permitindo criar uma visualização em clusters das contas Brasil e Portugal.

4 Discussão dos Resultados

Nesta seção são discutidos os resultados obtidos a partir de uma análise crítica dos mesmos, comparando os métodos entre si e com algumas informações presentes na literatura dos últimos trabalhos relacionados com esta dissertação. As contas anômalas detectadas são verificadas através dos detectores de *bots* disponíveis atualmente na internet.

4.1 Seleção das *Features*

As *features* selecionadas a partir da utilização da técnica RFECV neste trabalho podem ser comparadas com as *features* selecionadas pelo trabalho de Yang (Yang et al , 2020).

Em 'Scalable and Generalizable Social *Bot* Detection through Data Selection', Yang realiza a seleção das principais *features* a partir de uma coleção de datasets conhecidos e também rotulados manualmente, utilizando classificadores Random Forests e validação cruzada em um framework para detectar *bots* em real time presentes no Twitter. As *features* selecionados por Yang et al são apresentadas na figura 41.

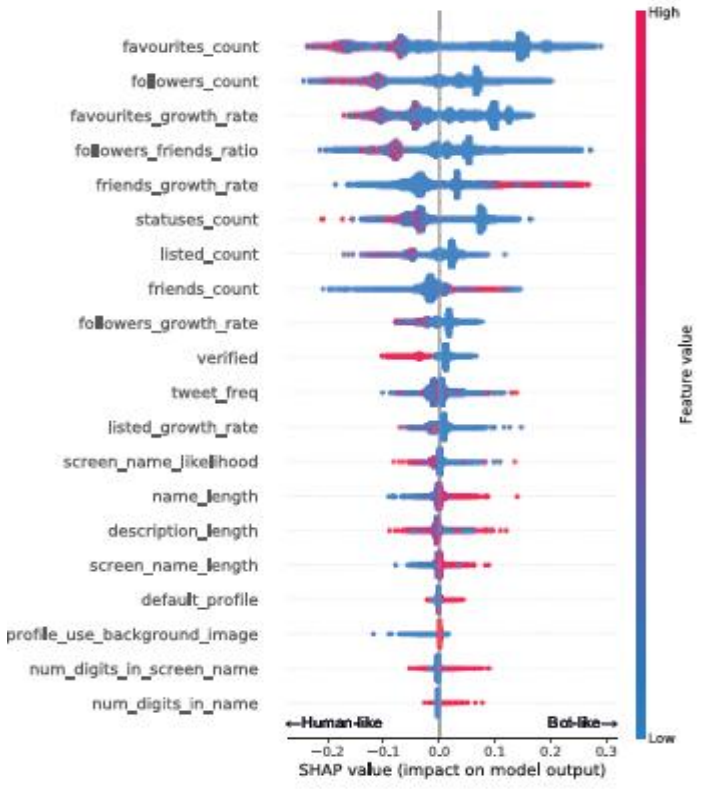

Figura 41: *Features* selecionadas por Yang (Yang et al, 2020).

A partir da figura 42 e comparando com a tabela 4 a seguir é possível observar que a maior parte das *features* selecionadas pelo RFECV estão presentes na seleção proposta por Yang (Yang et al, 2020), validando a técnica utilizada nesta dissertação como uma solução para escolha das principais *features* de forma não manual.

Esta abordagem permite que a seleção de *features* possa ser realizada de forma automatizada e garantindo que as principais *features* possam ser selecionadas e assim utilizadas no modelo de detecção de anomalias ou mesmo em outros trabalhos que utilizem técnicas de machine learning e que necessitem de redução e seleção das *features* mais importantes.

Feature selecionadas pelo RFECV
default_profile
description_length
favourites_count
followers_count
location
statuses_count
friends_count
name_length
url
listed_count
screen_name_length
verified

Tabela 4: *Feature* selecionadas pelo RFECV nesta dissertação.

4.2 Detector de anomalias

Os resultados obtidos pelos detectores de anomalia SVM *one-class* e Autoencoder Ensembles apresentaram globalmente boa performance.

A escolha da métrica Precision como *ground truth* para otimização dos modelos permitiu a detecção de mais contas normais em detrimento de menor identificação das contas anômalas, pois foi considerado mais importante garantir que as contas normais não sejam erroneamente classificadas como anômalas que o contrário, ou seja, que não ocorram falsos negativos.

Os resultados do detector de anomalias SVM *one-class* para os dados Brasil e Portugal são apresentados na tabela 5 abaixo, incluindo as contas anômalas não certificadas e certificadas pelo Twitter.

As contas anômalas certificadas (7 Brasil e 2 Portugal), que no total de contas certificadas computam 673 Brasil e 59 Portugal, foram detectadas de forma errônea pelo SVM *one-class*, foram excluídas do dataset utilizado na etapa de clusterização.

Dataset	Total contas	Normal	Anômala	Anômala não certificada	Anômala certificada
BR	41101	39051	2050	2043	7
PT	2726	2588	138	136	2

Tabela 5: Contas identificadas no dataset real SVM *one-class.*

Para o detector de anomalias Autoencoder Ensembles, os resultados para os dados Brasil e Portugal são apresentados na tabela 6 abaixo, incluindo as contas anômalas não certificadas e certificadas pelo Twitter.

As contas anômalas certificadas (8 Brasil e 3 Portugal), que no total de contas certificadas computam 673 Brasil e 59 Portugal, foram detectadas de forma errônea pelo Autoencoder Ensembles, foram excluídas do dataset utilizado na etapa de clusterização.

Dataset	Total contas	Normal	Anômala	Anômala não certificada	Anômala certificada
BR	41101	39034	2067	2059	8
PT	2726	2594	132	129	3

Tabela 6: Contas identificadas no dataset real Autoencoder Ensembles.

Após análise das contas detectadas pelos algoritmos SVM *one-class* e pelo Autoencoder Ensembles a partir dos resultados apresentados nas tabelas 5 e 6, podemos verificar que os dois detectores apresentaram resultados muito próximos na detecção de contas anômalas certificadas e não certificadas.

Estes resultados permitem inferir que tanto a utilização de um modelo de machine learning como o SVM *one-class* quanto a utilização de uma rede neural como o Autoencoder Ensembles permitem obter resultados com uma boa performance, sendo a utilização de aprendizagem não supervisionada

uma boa escolha neste tipo de problema ao não ser necessária a utilização de dados rotulados no treinamento dos algoritmos.

Após a retirada das contas certificadas identificadas como anômalas, foram selecionadas as contas anômalas detectadas pelo SVM *one-class* para os dados Brasil e as contas anômalas identificadas pelo Autoencoder Ensembles para os dados Portugal para a etapa de clusterização.

Essa escolha foi devido a quantidade de *tweets* associado a cada conta anômala detectada, pois o SVM *one-class* detectou contas anômalas com mais *tweets* publicados para os dados Brasil e o Autoencoder Ensembles para os dados Portugal, considerando que contas anômalas tendem a publicar *tweets* em quantidade muito superior a de um usuário real no Twitter.

Comparando o SVM *one-class* com o Autoencoder Ensembles nos resultados dos dados de validação, o desempenho dos dois modelos foi muito próximo conforme a tabela 7.

matriz de classificação SVM one-class True Positives: 1085 False Positives: 71 True Negatives: 127 False Negatives: 7	matriz de classificação Autoencoder Ensembles True Positives: 1061 False Positives: 126 True Negatives: 89 False Negatives: 15

Tabela 7: Matriz de classificação dados de validação SVM *one-class* e Autoencoder Ensembles.

Considerando a distribuição dos dados normais e anômalos no conjunto de validação dos dois modelos, é possível observar que o Autoencoder Ensembles classificou mais Falsos positivos, confirmando que este teve os valores de Precision menor que o do SVM *one-class*.

Uma possibilidade de aumentar os valores de Precision do Autoencoder Ensembles é realizar otimização de alguns parâmetros como por exemplo o número de camadas escondidas ou de neurônios na camada escondida ou ainda o valor de inicialização dos pesos.

As contas anômalas detectadas pelo SVM *one-class* e pelo Autoencoder Ensembles foram verificadas através de alguns detectores mencionados em Trabalhos on-line para avaliar se as contas detectadas pelos modelos desenvolvidos nesta dissertação também são identificadas por estas ferramentas.

Foi realizada a ordenação das contas anômalas a partir dos maiores valores de erro de reconstrução para as contas Portugal detectadas pelo Autoencoder Ensembles e selecionadas algumas contas Brasil detectadas pelo SVM *one-class*.

A partir do id das contas (número de identificação da conta no Twitter) foi pesquisada a classificação destas contas nas ferramentas on-line '*Bot*ometer'e 'Pega *Bot*' conforme apresentado nas figuras a seguir.

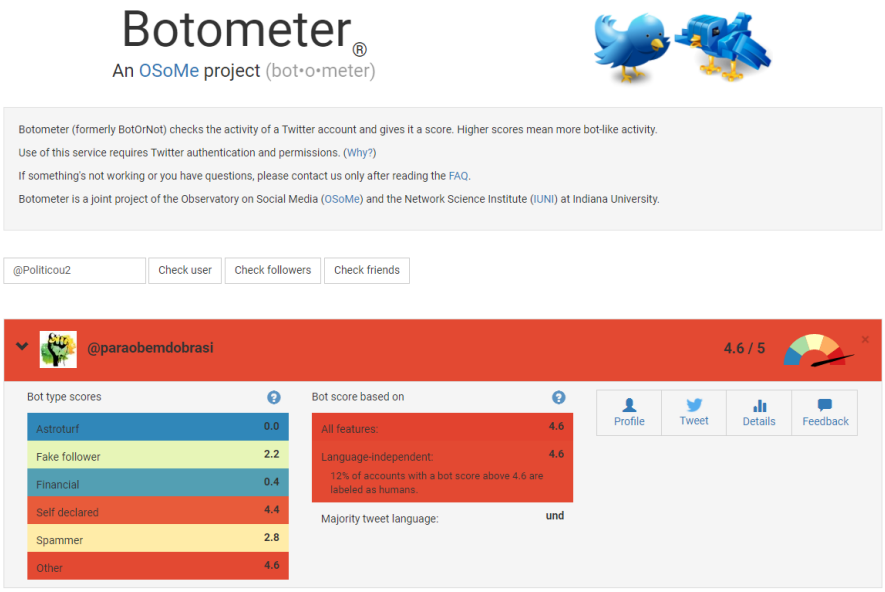

Figura 42: Conta Brasil também identificada como anômala no *Bot*ometer.

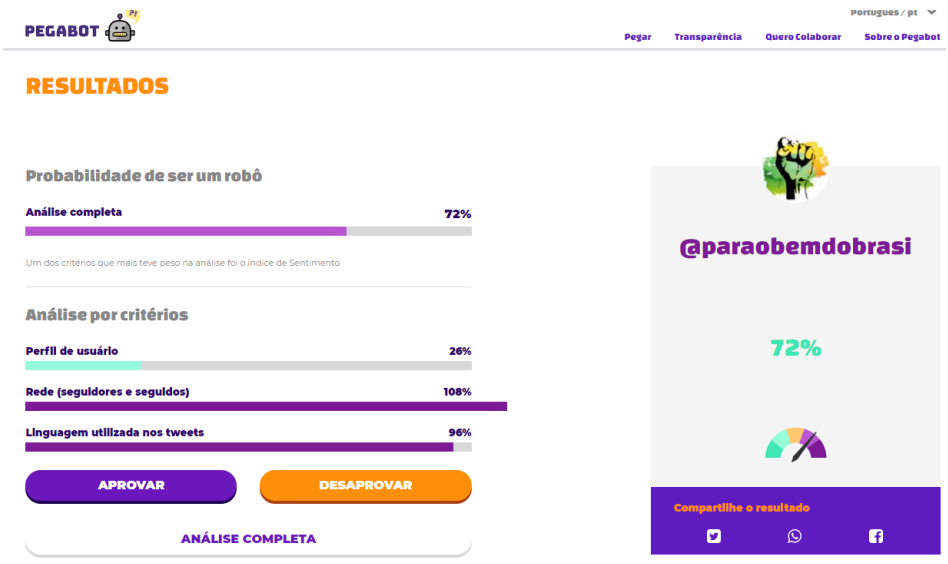

Figura 43: Conta Brasil também identificada como anômala no Pega *Bot.*

Outras contas que estão inativas foram identificadas como anômalas no SVM *one-class* e no *Bot*ometer como pode ser visto nas figuras 44 e 45. No Pega *bot* a conta tem uma probabilidade média de ser um *bot* conforme figura 46.

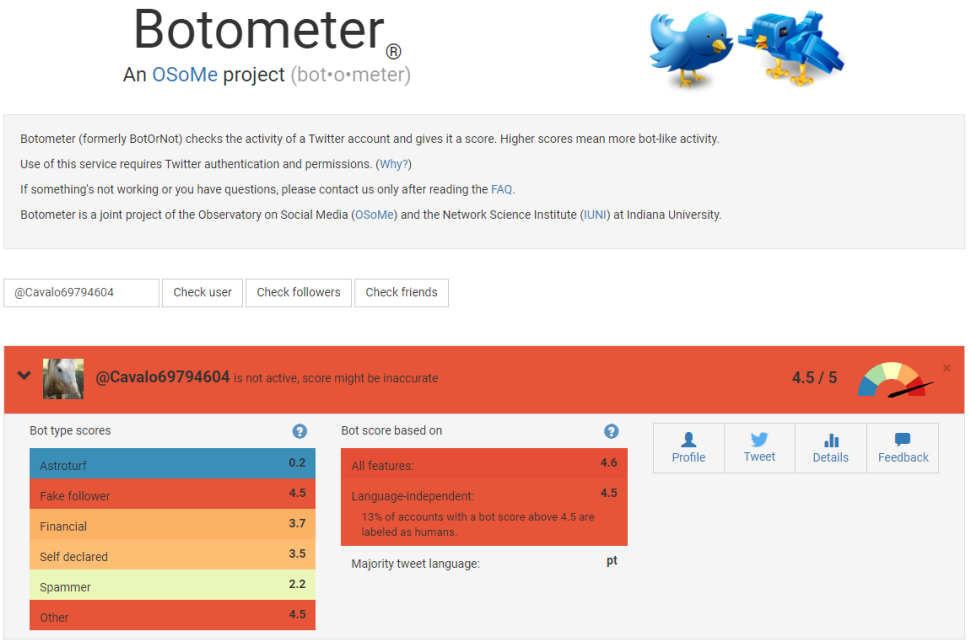

Figura 44: Conta Brasil também identificada como anômala no *Bot*ometer.

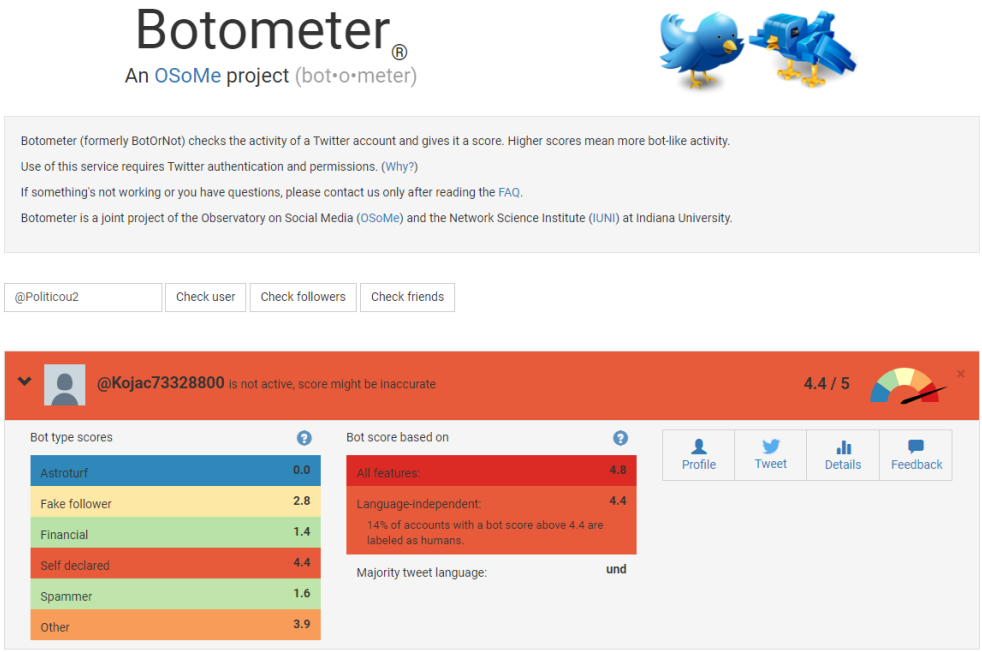

Figura 45: Conta Brasil também identificada como anômala no *Bot*ometer.

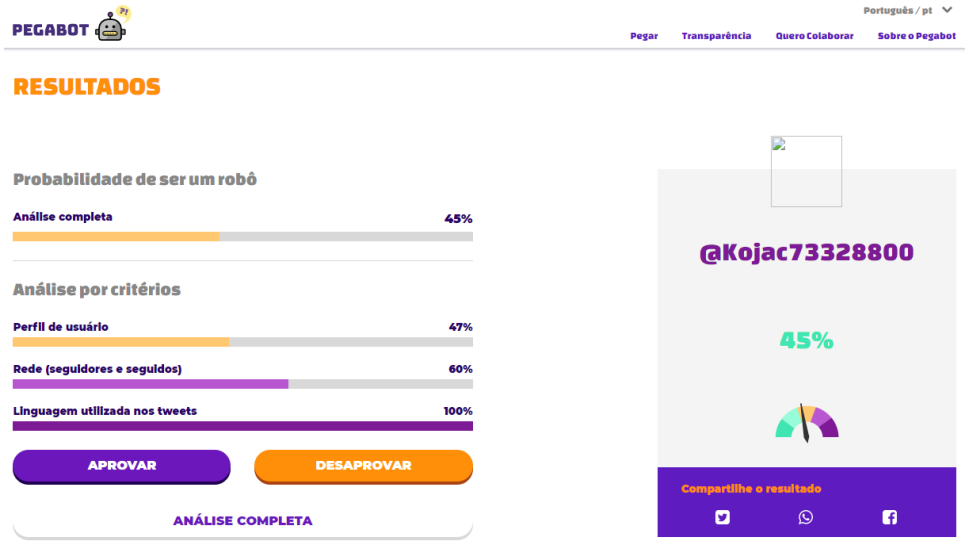

Figura 46: Conta Brasil com probabilidade média de ser anômala no Pega *Bot*.

A mesma verificação foi realizada para as contas detectadas pelo Autoencoder Ensembles dos dados Portugal e também foram identificadas como anômalas pelos detectores on-line, conforme figuras 47 a 49 a seguir.

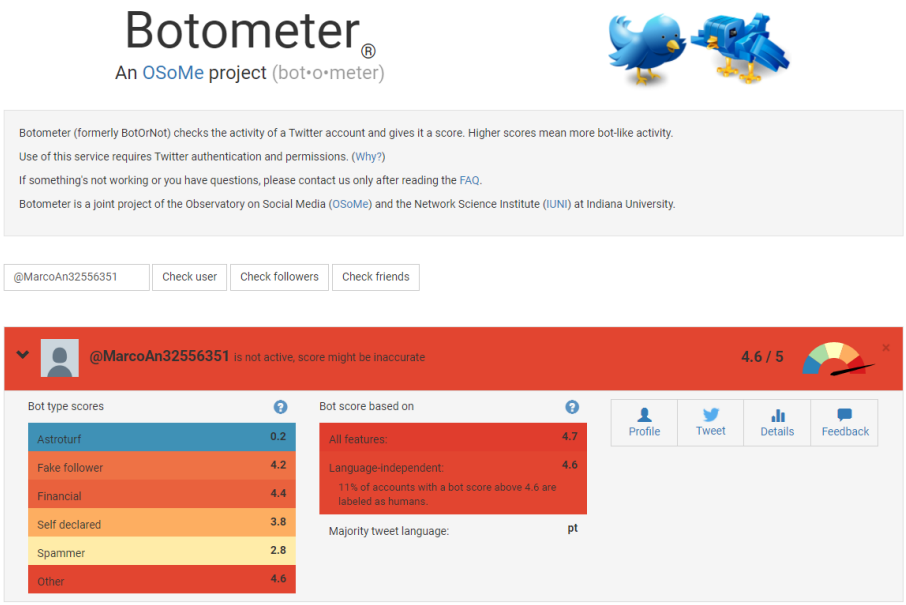

Figura 47: Conta Portugal também identificada como anômala no *Bot*ometer.

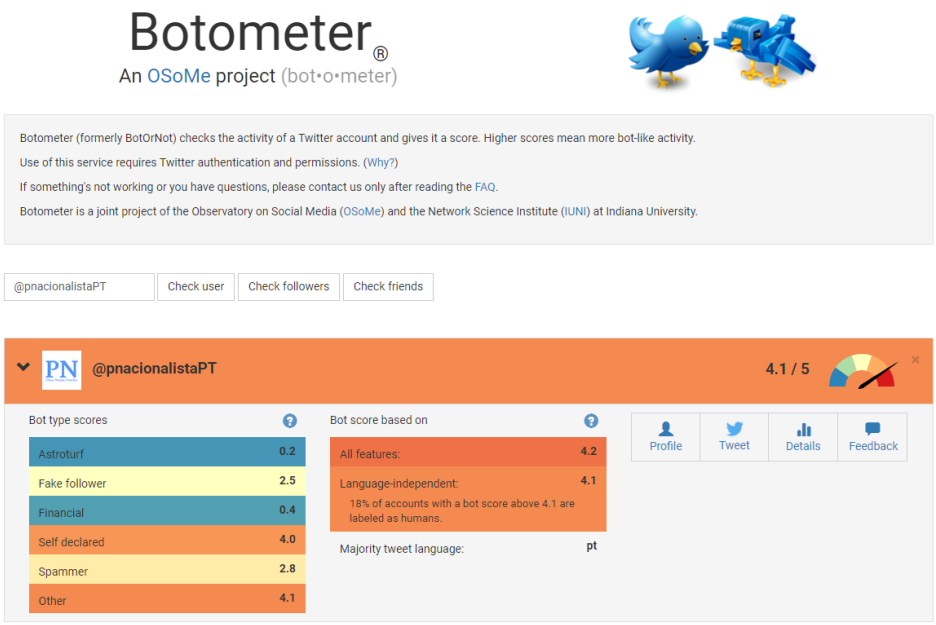

Figura 48: Conta Portugal também identificada como anômala no *Bot*ometer.

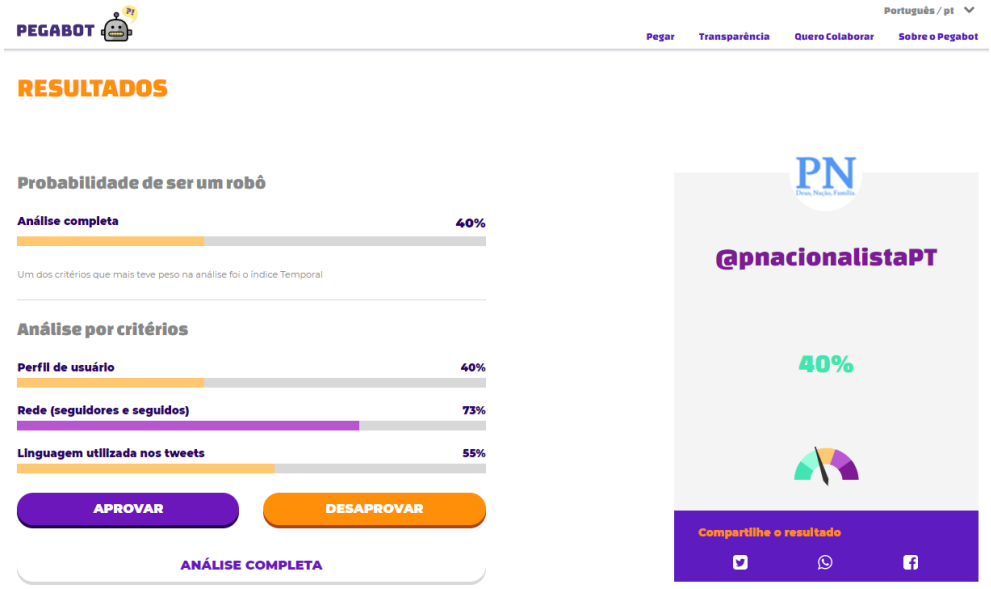

Figura 49: Conta Portugal também identificada como anômala no Pega *Bot*.

Todas as contas anômalas na amostra foram sinalizadas com alta pontuação de atividade de um *bot* ou probabilidade de ser *bot*, permitindo inferir que os algoritmos implementados neste trabalho possuem boa capacidade na detectores de contas anômalas, validando o trabalho desenvolvido.

4.3 Clusterização

Com o uso do WordCloud pode-se verificar as palavras mais frequentes presentes nos *tweets* das contas anômalas e do contéudo publicado relacionado à discursos políticos.

A partir da representação vetorial criada pelo Word2Vec e pela função desenvolvida em Python foi possível criar vetores para cada *tweet* das contas Brasil e Portugal, sendo esta etapa fundamental para representar os *tweets* em clusters utilizando o Autoencoder como redutor de dimensionalidade.

Com a utilização da rede neural Autoencoder foi possível gerar uma visualização de clusters que permitiram verificar a ocorrência de diferentes tipos de *tweets* das contas anômalas entre as contas Brasil e Portugal conforme apresentado na figura 50 a seguir.

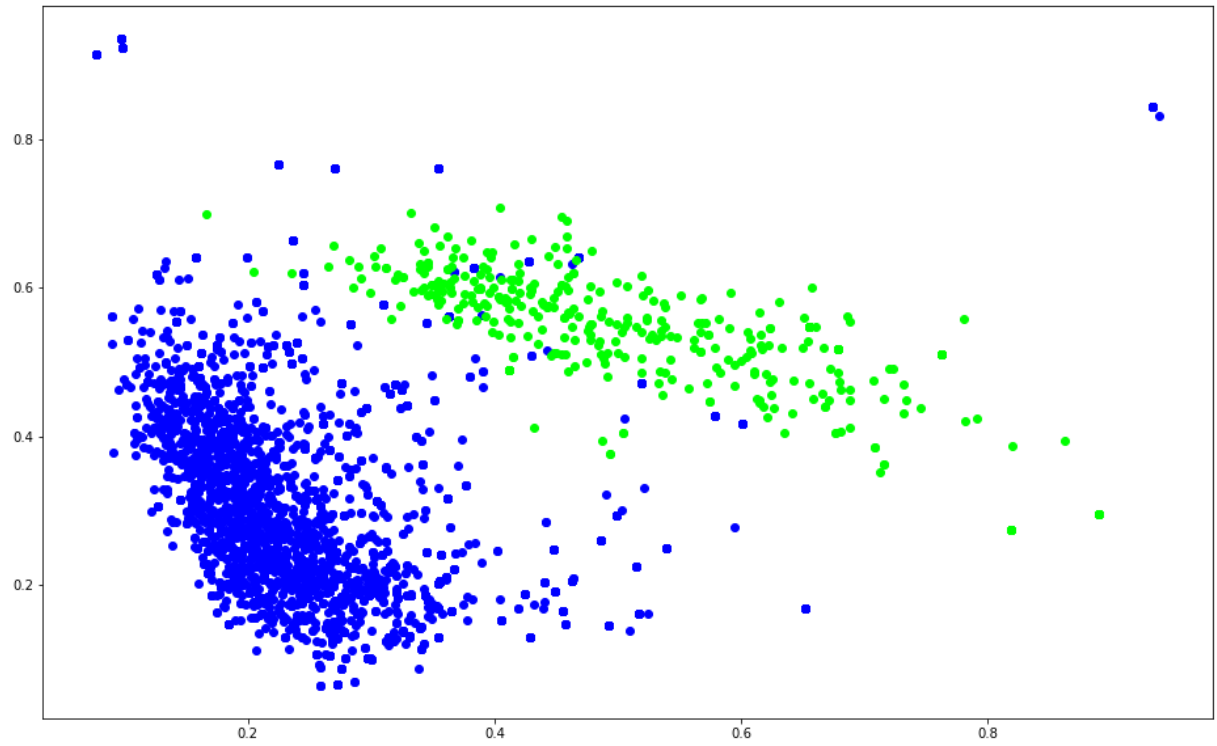

Figura 50: Clusters Brasil (azul) e Portugal (verde) a partir da redução de dimensão com Autoencoder.

É possível verificar que existe diferença nos *tweets* publicados pelas contas anômalas nos dados Brasil em azul, com alguns poucos tipos de *tweets* diferentes, comparativamente para os dados de contas Portugal em verde, apesar da menor quantidade de *tweets*. Essa diferença entre as contas Brasil e Portugal pode ocorrer devido a uma possível diferença de discurso político entre os dois países.

Essa visualização permite ainda inferir que as contas anômalas utilizam palavras que estão muito relacionadas entre si e que publicam conteúdo praticamente homogêneo, característica inerente ao comportamento de *bots*, ou seja, não parece existir vários tipos diferentes de *tweets* publicados pelas contas anômalas. Isso pode ser observado com maior prepoderância nos caso das contas anômalas do Brasil.

Com esta redução da dimensão dos dados a partir da utlização de Autoencoders e a criação de clusters, foi possível elaborar uma ferramenta automatizada que permite numa próxima etapa analisar o conteúdo dos *tweets* das contas anômalas do Brasil e de Portugal e também verificar os tipos diferentes de *tweets* das contas anômalas.

5 Conclusão

A primeira parte deste trabalho conseguiu selecionar as principais *features* de forma não manual através da técnica RFECV a partir de conjunto de dados rotulados e disponibilizados publicamente em repositório on-line. As *features* selecionadas como mais importantes neste trabalho também estão presentes em outros trabalhos mais recentes e que foram apresentados na revisão da literatura, validando a seleção das *features* mais importantes através desta técnica.

Em seguida foram utilizados dois algoritmos independentes para criar um detector de anomalias a partir de contas do Twitter, o SVM *one-class* e o Autoencoder Ensembles. Para o Autoencoder Ensembles foi desenvolvido um método de ensembles a partir da distribuição de probabilidade dos dados de entrada variando em função do valor do erro de reconstrução do Autoencoder anterior com o treinamento dos Autoencoders em sequência.

A utilização de métodos de otimização dos hiperparâmetros como o ParameterGrid e a função desenvolvida em Python permitiu melhorar os valores de Precision do SVM *one-class*. Para o Autoencoder Ensembles foi utilizada a técnica Dropout que também aumentou os valores de Precision do modelo nos dados de validação.

Uma possibilidade futura de trabalho é a otimização com a utilização dos métodos Grid Search ou Randomized nos hiperparâmetros do Autoencoder Ensembles, pois esta rede neural possui muitos hiperparâmetros e realizar *Fine-tune Hyperparameters* para encontrar a melhor combinação destes demanda uma alta disponibilidade de tempo.

Os dois detectores apresentaram boa performance, apenas o Autoencoder Ensembles teve valores de Precision abaixo dos valores alcançados pelo SVM *one-class*, mas que pode ser melhorado através da otimização dos hiperparâmetros como citado acima.

Como forma de validar o detector de anomalias deste trabalho, foram verificadas algumas contas anômalas em detectores on-line e todas estas foram também identificadas com alta probabilidade de serem *bots* ou terem comportamentos anômalos, comprovando que os detectores SVM *one-class* e Autoencoder Ensembles são assertivos em identicar anomalias e comprovando que a utilização de modelos de aprendizado não supervisionado podem ter uma boa performance em problemas de detecção de anomalias.

As vantagens apresentadas neste trabalho são a utilização da técnica RFECV em permitir a utilização destes algoritmos sem a necessidade de listas de palavras-chave, já que as *features* necessárias para o treinamento destes modelos foram selecionadas de forma não manual.

Outra vantagem é a utilização destes modelos de forma mais atemporal, pois o treinamento considerou as informações das contas reais e não de contas anômalas. Esta abordagem é devido ao fato do comportamento de contas reais não variar muito ao longo do tempo, ao contrário das contas anômalas que tendem a imitar o comportamento humano e ser mais difícil sua identificação. Esta última permite que os detectores desenvolvidos sejam reutilizados em outras eleições para qualquer país, independente da língua utilizada neste país, sendo esta uma inovação proposta por este trabalho.

Por fim, os *tweets* das contas anômalas foram representados em um espaço vetorial utilizando Word2Vec e tiveram sua dimensão reduzida a partir da utilização de um Autoencoder constituído de funções lineares e sigmóide para a criação de clusters.

Os clusters criados a partir do Autoencoder permitiram visualizar que existe diferença de discurso político nas contas anômalas durante as eleições no Brasil e em Portugal, indicando que contas anômalas tendem a publicar um conteúdo em *tweets* de forma específica para cada eleição. Para as contas Brasil foi verificado que os *tweets* estão muito próximos, podendo indicar homogeneidade em seu conteúdo, não sendo verificada a ocorrência de vários tipos diferentes de *tweets*.

Por fim, uma outra possibilidade futura deste trabalho é analisar o conteúdo dos *tweets* das contas anômalas do Brasil e de Portugal e também verificar os tipos diferentes de *tweets* das contas anômalas.

6 Referências Bibliográficas

Aaker, J., Chang, V. (2010). Obama and the power of social media and technology, The European Business Review, 16-21, May - June 2010.

Abokhodair, N., Yoo, D., McDonald, D.W. (2015). Dissecting a Social *Bot*net: Growth, Content and Influence in Twitter, Leveraging Language, CSCW 2015, March 14-18, 2015, Vancouver, BC, Canada.

Araújo, V.A. (2011). *Visibilidade em detrimento da interatividade O Twitter nas recentes eleições presidenciais de Portugal e do Brasil.* Tese de Mestrado em Ciências da Comunicação – variante em Comunicação Política. Porto: Universidade do Porto.

Aggarwal, C. C. (2017). Outlier Analysis. Second Edition, Springer International Publishing AG 2017.

Arnaudo, D. (2017). Computational Propaganda in Brazil: *Social bots* during Elections, Computational Propaganda Research Project, Working Paper No.2017.8

Baldi, P. (2012). Autoencoders , Unsupervised Learning , and Deep Architectures. In UTLW'11 Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning workshop (Vol. 27, pp. 37–50).

Bellman, R. (1966). Dynamic programming. Science, 153 (3731), 34–37.

Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C. (2003). A Neural Probabilistic Language Model. Journal of *Machine learning* Research 3, 1137–1155.

Bengio, Y., Schwenk, Holger; Senécal, Jean-Sébastien; Morin, Fréderic; Gauvain, Jean-Luc (2006). A Neural Probabilistic Language Model. Studies in Fuzziness and Soft Computing. 194. pp. 137–186.

Brachten, F., Stieglitz, S., Hofeditz, L., Kloppenborg, K., Reimann, A. (2017). Strategies and Influence of *Social bots* in a 2017 German state election – A case study on Twitter, Australasian Conference on Information Systems 2017, Hobart, Australia.

Breiman, L (1994). *Bagging Predictors,* Technical report N412, Department of Statistics, University of California.

Breiman, L (2001). *Random Forests. Machine learning,* Springer, v. 45, n. 1, p. 5–32.

Caetano, J.A. (2018). *Characterizing politically engaged users during the 2016 us presidential campaign using twitter*. Tese de mestrado em *Program in Informatics*. Brasil: Pontifical Catholic University of Minas Gerais.

Caverlee, J., Lee, K., Eoff, B.D. (2011). Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter, ICWSM.

Chavoshi, N., Hamooni, H., Mueen, A. (2017). On-Demand *Bot* Detection and Archival System, International World Wide Web Conference Committee (IW3C2).

Chen, J., Sathe, S., Aggarwal, C., Turaga, D. (2017). Outlier Detection with Autoencoder Ensembles. Proceedings of the 2017 SIAM International Conference on Data Mining (pp.90-98).

Cortes, C., Vapnik, V. (1995). Support-Vector Networks. *Machine learning*, 20, 273-297.

Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., Tesconi, M. (2017). The paradigm-shift of social spam*bots*: Evidence, theories, and tools for the arms race, In Proceedings of the 26th International Conference on World Wide Web Companion (pp. 963-972), ACM.

Davis, C., Varol, O., Ferrara, E., Flammini, A., & Menczer, F. (2016). *Bot*OrNot: A System to Evaluate *Social bots*, In Proceedings of The 25Th International Conference Companion on World Wide Web - WWW '16 Companion, 274-274.

Efthimion, P.G., Payne, S., Proferes, N. (2018). Supervised *Machine learning Bot* Detection Techniques to Identify Social Twitter *Bots*, SMU Data Science Review: Vol. 1 : No. 2 , Article 5.

European Union (2018). A multi-dimensional approach to disinformation, Report of the independent High level Group on fake news and online disinformation, Directorate-General for Communications Networks, Content and Technology (European Commission).

Alpaydin, E. Introduction to *Machine learning*. The MIT Press, 2nd edition, 2010.

Faceli, K., Lorena, A. C., Gama, J. e Carvalho, A. C. P. L. F. (2011). Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina. LTC. Rio de Janeiro.

Fernquist, J., Kaati, L., Schroeder, R. (2018). Political *Bots* and the Swedish General Election, In 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), 124-129.

Ferrara, E., Varol, O., Davis, C., Menczer, F., & Flammini, A. (2016a). The rise of *social bots*, Communications of the ACM, 59(7), 96–104.

Ferrara, E., Bessi, A. (2016b). *Social bots* distort the 2016 U.S. Presidential election online discussion, First Monday, Volume 21, Number 11 – 7.

Ferrara, E. (2017). Disinformation and social *bot* operations in the run up to the 2017 french presidential election, University of Southern California, Information Sciences Institute.

Finkel, J., Luo, M., Metaxa-Kakavouli, D., Peeples, C., Shenoy, A., Echeverr y, N.T. (2017). Fake news and Misinformation: The roles of the nation's digital newsstands, Facebook, Google, Twitter and Reddit.

Freund e Schapire (1995) Yoav Freund e Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Em Proceedings of the Second European Conference on Computational Learning Theory, EuroCOLT '95, páginas 23–37, London, UK, UK. Springer-Verlag. ISBN 3-540-59119-2.

Freund e Schapire (1999) Yoav Freund e Robert E. Schapire. A Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.

Friedman, J.; Hastie, T.; Tibshirani, R (2001). The elements of statistical learning. [S.l.]: Springer series in statistics New York, v.1.

Géron, Aurélien (2017). Hands-On *Machine learning* with Scikit-Learn and TensorFlow (Concepts, Tools, and Techniques to Build Intelligent Systems), First Edition, O'reilly.

Gonçalves, C. I. S. (2012). A *Influência da Rede Social Facebook na Decisão de Voto.* Tese de Mestrado em Comunicação Estratégica. Lisboa: Faculdade de Ciências Sociais e Humanas. Universidade Nova de Lisboa.

Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep Learning. Cambridge, MA : MIT Press, [2017] | Series: Adaptive computation and *machine learning* series.

Gorwa, R., Guilbeault, D. (2018). Unpacking the Social Media *Bot*: A Typology to Guide Research and Policy, ICA 2018, Prague (CZ).

Graves, L., Cherubini, F. (2016). The Rise of Fact-Checking Sites in Europe, Oxford: Reuters Institute for the Study of Journalism.

Grimme, C., Preuss, M., Adam, L., Trautmann, H. (2017). *Social bots*: Human-Like by Means of Human Control?, Big Data, 5(4), 279-293.

Guyon, I., Weston, J., Barnhill, S., Vapnik, V. (2002). Gene Selection for Cancer Classification using Support Vector Machines. *Machine learning*, 46, 389–422.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation (2nd Edition), Prentice Hall.

Haykin, S. (2009). Neural Networks and Learning Machines (3rd Edition), Prentice Hall.

Han, J. e Kamber, M. (2006). The Data Mining: Concepts and Techniques. Second Edition, Morgan Kaufmann Publishers. San Francisco.

Hastie, T., Tibshirani R., Friedman J. (2008). The elements of Statistical Learning: Data Mining, Inference and Prediction. Second edition, Springer.

Howard, P.N., Kollanyi B. (2016). *Bots*, Strongerin and Brexit: Computational propaganda during the UK-EU referendum, Working Paper 2016.1, Oxford: Computational Propaganda Project, 2016.

Java, A., Song, X., Fini, T., Tseng, B. (2007). Why We Twitter: Understanding Microblogging Usage and Communities, 9th WEBKDD and 1st SNA-KDD Workshop '07.

Kan, M, Shan, S. Chang, H., Chen, X. (2014). Stacked Progressive Auto-Encoders (SPAE) for Face Recognition Across Poses. IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1883-1890, doi: 10.1109/CVPR.2014.243.

Kaplan, A.M., Haenlein, M. (2010). Users of the world, unite! The challenges and opportunities of Social Media, Business Horizons (2010) 53, 59-68.

Krippahl, L. (2017). Aprendizagem Automática (*Machine learning*). Lecture Notes.

Kramer, Mark A. (1991). "Nonlinear principal component analysis using autoassociative neural networks" (PDF). AIChE Journal. 37(2): 233–243.

Lorena, A. C., Carvalho, A. C. P. L. F. (2007). Uma introdução às *Support Vector Machines*. RITA, Volume XIV, Número 2.

Manevitz, L.M., Yousef, M. (2001). *One-class* SVMs for Document Classification. Journal of *Machine learning* Research 2, 139-154.

Martín, B.R. (2019). *La comunicación política de Podemos en las redes sociales Facebook y Twitter: estudio de caso*. Tese de doutorado em *CIENCIAS DE LA INFORMACIÓN*. Espanha: *UNIVERSIDAD COMPLUTENSE DE MADRID, FACULTAD DE CIENCIAS DE LA INFORMACIÓN*.

Mazza, M., Cresci, S., Avvenuti, M., Quattrociocchi, W., Tesconi, M. (2019). RTbust: Exploiting Temporal Patterns for *Bot*net Detection on Twitter. In Proceedings of ACM Web Science Conference 2019 (WebSci 2019). ACM, New York, NY, USA, 10 pages.

Mercer, J. (1909). Functions of positive and negative type, and their connection the theory of integral equations. Mathematical, Physical and Engineering sciences, article XVI.

Mikolov, T., Chen, K., Corrado, G., Dean J. (2013). Efficient Estimation of Word Representations in Vector Space. 1-12.

Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A. (2018). Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. Ben-Gurion University of the Negev. 10.14722/ndss.2018.23211.

Morstatter, F., Pfeffer, J., Liu, H., Carley, K.M. (2013). Is the sample good enough? Comparing data from Twitter's Streaming API with Twitter's Firehose, Proceedings of the Seventh International AAAI Conference on Weblogs and Social Media.

Muntean, A. (2015). *The Impact of Social Media Use of Political Participation*. Tese de Mestrado em *Corporate Communication*. Dinamarca: *Aarhus University*.

Nielsen, R.K., Kalogeropoulos, A., Newman, N., Fletcher, R. (2019). Reuters Institute, Digital News Report 2019, Reuters Institute for the Study of Journalism.

Nulty, P., Theocharis,Y., Popa, S.A., Parnet, O., Benoit, K. (2016). Social media and political communication in the 2014 elections to the European Parliament, Electoral Studies 44, 429-444.

Oentaryo, R., Murdopo, A., Prasetyo, P., Lim, E. (2016). On Profiling *Bots* in Social Media, Social informatics: 8th International Conference, SocInfo 2016, Bellevue, WA, November 11-14: Proceedings, Pages 92-109.

Paletz, D.L., Cook, T.E., Owen, D.M. (2015). American Government and Politics in The Information Age, FlatWorld, version 2.0.

Patgiri, R., Varshney, U., Akutota, T.and Kunde R. (2018). An Investigation on Intrusion Detection System Using *Machine learning*. Department of Computer Science & Engineering, National Institute of Technology Silchar, Assam-788010, India. IEEE Symposium Series on Computational Intelligence SSCI 2018.

Pinto, M.V., Kogan, A., Rondon, T., Aquino, E.L.C., Junior, C.A.M. (2018). Recomendações sistêmicas para combater a desinformação nas eleições do Brasil, IT&E – Instituto Tecnologia e Equidade, São Paulo.

Rahmawati, I. (2014). *Social media, politics, and young adults: The impact of social media use on young adults' political efficacy, political knowledge, and political participation towards 2014 Indonesia General Election*. Tese de Mestrado em Communication Studies. Países Baixos: University of Twente. Faculty of Behavioral Sciences.

Rawat, T., Khemchandani, V. (2019). Feature Engineering (FE) Tools and Techniques for Better Classification Performance. International Journal of Innovations in Engineering and Technology (IJIET).

Rossi, Sippo (2019). *Detecting and analyzing bots on Finnish political twitter*. Tese de Mestrado em *Information and Service Management*. Finlândia: Aalto University School of Business.

Ruediger, M. A. (2018). *Bots* and Brazil's Electoral Legal System in the 2018 Elections. Rio de Janeiro: FGV DAPP, 2018. (Policy Paper Sala de Democracia Digital #Observa2018, 3).

Ruediger, M. A. et al. (2018). Robôs, Redes Sociais e Política no Brasil: Análise de interferências de perfis automatizados nas eleições de 2014. Policy Paper . Rio de Janeiro: FGV DAPP, 2018.

Sakurada, Mayu, and Takehisa Yairi. 2014. "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction." In *Proceedings of the Mlsda 2014 2nd Workshop on Machine learning for Sensory Data Analysis*, 4. ACM.

Sales, C.O.T (2016). *As Novas Tecnologias da Informação e Comunicação ao serviço da Democracia – As eleições legislativas portuguesas em 2015*. Tese de Mestrado em Ciências da Comunicação – Comunicação Estratégica. Lisboa: Faculdade de Ciências Sociais e Humanas. Universidade Nova de Lisboa.

Santos, A. (2017). O impacto do big data e dos algoritmos nas campanhas eleitorais, ITSRio, Rio de Janeiro.

Sanovich, S., Stukal, D., Tucker, J.A. (2017). Turning the Virtual Tables: Government Strategies for Addressing Online Opposition with an Application to Russia, Comparative Politics, 50, 435-482.

Sarvari, H., Domeniconi, C., Prenkaj, B., Stilo, G. (2019). Unsupervised Boosting-based Autoencoder Ensembles for Outlier Detection.

Schäfer, F., Evert, S., & Heinrich, P. (2017). Japan's 2014 General Election: Political *Bots*, Right-Wing Internet Activism, and Prime Minister Shinzō Abe's Hidden Nationalist Agenda. Big data, 5(4), 294-309.

Scholkopf, C. J. C. Burges, and A. J. Smola (1999a). Advances in Kernel Methods. Support Vector Learning. MIT Press, Cambridge, MA.

Scholkopf, J.C. Platt, J.Shawe-Taylor, A.J. Smola, and R.C. Williamson (1999b). Estimating the support of a high-dimensional distribution. Technical report, Microsoft Research, MSR-TR-99-87.

Scholkopf, B., Smola, A.J. (2002). Support Vector Machines and Kernel Algorithms. Encyclopedia of Biostatistics, 5328-5335.

Singh, A. (2018). *Social Media Analytics and the Role of Twitter in the 2014 South African General Election: A Case Study*. Tese de Mestrado em *Master of Science.* África do Sul: University of the Witwatersrand.

Souza, B. F. (2005). Seleção de características em SVMs aplicadas a dados de expresão gênica. Tese de Mestrado em Ciências da computação e Matemática computacional. São Paulo: USP.

Stukal, D., Sanovich, S., Bonneau, R., Tucker, J. A. (2017). Detecting *bots* on Russian political Twitter, Big Data, 5, 310-324.

Stukal, D., Sanovich, S., Bonneau, R., Tucker, J. (2019a). The Use of Twitter *Bots* in Russian Political Communication, PONARS Eurasia Policy Memo No. 564.

Stukal, D., Sanovich, S., Tucker, J.A, Bonneau, R. (2019b). For Whom the *Bot* Tolls: A Neural Networks Approach to Measuring Political Orientation of Twitter *Bots* in Russia, SAGE Open April-June 2019: 1–16.

Suuronen, A. (2018). *Political competition and social media: Can Facebook change the status quo of Finnish politics?*. Tese de Mestrado em *Public Choice*. Finlândia: University of Tampere, Faculty of Social Sciences.

Tan, P.-N., Steinbach, M. e Kumar, V. (2006). Introdution to Data Mining. Pearson Addison-Wesley.

Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., Kappas, A. (2010). Sentiment strength detection in short informal text, Journal of the American Society for Information Science and Technology, volume 61, number 12.

Thieltges, A., Papakyriakopoulos, O., Serrano, J., Hegelich, S. (2018). Effects of *Social bots* in the Iran-Debate on Twitter, Bavarian School of Public Policy, Technical University of Munich.

Tufekci, Z. (2014). Engineering the public: big data, surveillance and computational politics, First Monday, Volume 19, Number 7.

Varol, O., Ferrara, E., Davis, C., Menczer, F., Flammini, A. (2017). Online Human-*Bot* Interactions: Detection, Estimation, and Characterization, International AAAI Conference on Web and Social Media.

Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P. (2008). Extracting and composing robust *features* with denoising autoencoders. Proceedings of the 25th International Conference on *Machine learning*. 1096-1103.

Wang, C.,Pan, Y., Chen, J.,Ouyang, Y., Rao, J., Jiang, Q. (2020). Indicator element selection and geochemical anomaly mapping using *Recursive Feature Elimination* and random forest methods in the Jingdezhen region of Jiangxi Province, South China. Applied Geochemistry 122 (2020) 104760.

Walt, E.V.D, Eloff, J. (2018). Using *Machine learning* to Detect Fake Identities: *Bots* vs Humans, IEEE Access, Volume 6, 2018.

Wijeratne, S., Bhatt, A.S.S., Balasuriya, L., Al-Olimat, H.S., Gaur, M., Yazdavar, A.H., Thirunarayan, H. (2017). Feature Engineering for Twitter-based Applications, Chapter 1, In book: Feature Engineering for *Machine learning* and Data Analytics, Edition: Data Mining and Knowledge Discovery Series, Publisher: Chapman and Hall, Editors: Guozhu Dong, Huan Liu.

Woolley, S.C., Kollanyi, B., Howard, P.N. (2016). *Bots* and Automation over Twitter during the U.S. Election, COMPROP DATA MEMO 2016.4 / 17 NOV 2016.

Yang, K.C., Varol, O., Hui, P.M., Menczer, F. (2020). Scalable and Generalizable Social *Bot* Detection through Data Selection, The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20).

Zheng, A., Casari, A. (2018). Feature Engineering for *Machine learning*. O'Reilly Media, April 2018: First Edition.

Zhou, Chong, and Randy C Paffenroth. 2017. "Anomaly Detection with Robust Deep Autoencoders." In *Proceedings of the 23rd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 665–74. ACM.

Internet

Borges, Liliana (2018). "Rui Rio já "*tweet*a. Como estão os políticos portugueses no Twitter?", em Jornal PÚBLICO. Lisboa: 01-12-2018. Disponível em: https://www.publico.pt/2018/12/01/politica/noticia/rui-rio-ja-*tweet*a-estao-politicos-portugueses-twitter-1853185. Acedido em: 22-05-2019.

Cotrim, Antonio (2018). "Próximas eleições portuguesas podem ser marcadas por 'fake news' diz investigador da Universidade do Minho", em Jornal Observador. Lisboa: 20-11-2018. Disponível em: https://observador.pt/2018/11/20/proximas-eleicoes-portuguesas-podem-ser-marcadas-por-fake-news-diz-investigador-da-universidade-do-minho. Acedido em: 23/05/2019.

Bot Repository (2020). Disponível em: https://*bot*ometer.osome.iu.edu/*bot*-repository/datasets.html

Acedido em: 26-02-2020.

Facebook (2012). "The 2012 Election Day Through the Facebook Lens". Disponível em: https://www.facebook.com/notes/facebook-data-science/the-2012-election-day-through-the-facebook-lens/10151181043778859/. Acedido em: 27-05-2019.

IFCN, International Fact-Checking Network (2019). "signatories". Disponível em: https://www.ifcncodeofprinciples.poynter.org/signatories. Acedido em: 25-06-2019.

Kearney, Michael W. (2019). "*tweetbot*ornot". Disponível em: http://*tweetbot*ornot.mikewk.com/. Acedido em: 10-06-2019.

Quartz (2015). "Millions of Facebook users have no idea they're using the internet". Disponível em: https://qz.com/333313/milliions-of-facebook-users-have-no-idea-theyre-using-the-internet/. Acedido em: 25-06-2019.

Statista (2019). "Number of monthly active Twitter users worldwide from 1st quarter 2010 to 1st quarter 2019 (in millions)" em Statista. Disponível em: https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/. Acedido em: 26-06-2019.

Silverman, C. "Recent research reveals false rumours really do travel faster and further than the truth". Disponível em: https://firstdraftnews.com/recent-research-reveals-false-rumours-really-do-travel-faster-and-further-than-the-truth. Acedido em: 20.06.19.

Twitter (2019a). "Glossary", em Twitter. Disponível em: https://help.twitter.com/en/glossary. Acedido em: 15-06-2019.

Twitter (2019b). "*Tweet* Object", em Twitter. Disponível em: https://developer.twitter.com/en/docs/*tweets*/data-dictionary/overview/intro-to-*tweet*-json.html. Acedido em: 20-06-2019.

Twitter (2019c). "FCTNOVA" em Twitter. Disponível em: https://twitter.com/FCTNOVA. Acedido em: 26-06-2019.

Twitter (2019d). "Twitter Termos de Serviço", em Twitter. Disponível em: https://twitter.com/pt/tos. Acedido em: 26-06-2019.

Twitter (2019e) “Data dictionary” em Twitter. Disponível em: https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/user-object Acedido em 20-06-2019.