UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ISADORA PEDRINI POSSEBON

Classificação de tráfego de rede utilizando técnicas de meta-learning

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho
Co-orientador: Msc. Anderson Santos da Silva

Porto Alegre
2018

“Enquanto não alcances
Não descanses,
De nenhum fruto queiras só metade.”
— MIGUEL TORGA

AGRADECIMENTOS

Este trabalho é fruto de um período de imensurável prazer e realizações. Por isso, agradeço aos meus pais - por todo amor, sacrifício e constante incentivo aos estudos. Aos meus irmãos, pela paciência e carinho diário. À minha grande família, pela paciência e palavras de apoio. Ao meu namorado e sua família, pela atenção e incontáveis conselhos.

Este trabalho também é resultado de todos os professores que passaram pelo meu caminho, todos os colegas de sala de aula e de trabalho. Por isso, agradeço a todos eles. Em especial, agradeço ao meu orientador e ao meu co-orientador, pelos sábios conselhos e pela inspiração recorrente.

Por fim, agradeço a todas as oportunidades que tive - à Universidade Federal do Rio Grande do Sul, ao Instituto de Informática, e à Rijksuniversiteit Groningen.

RESUMO

Classificação de tráfego de rede tem atraído bastante atenção e esforços de pesquisas, pois uma grande quantidade de áreas é beneficiada por esses estudos - desde segurança e gerenciamento de redes, qualidade de serviço (Quality of Service - QoS), sistemas de cobrança de tráfego de rede até detecção de intrusões (Intrusion Detection Systems - IDS). Inspirado em pesquisas anteriores, este trabalho apresenta uma análise comparativa entre técnicas de *meta-learning* e classificadores individuais para classificação de tráfego de rede. As técnicas utilizadas foram as mais comumente encontradas na literatura e disponibilizadas pela biblioteca *scikit-learn*, para a linguagem Python. Além disso, uma técnica adicional foi implementada (Stacking), devido aos bons resultados deste método apresentados na literatura.

Com isso em mente, propõe-se uma análise experimental de diferentes técnicas de *meta-learning* - também conhecidas como *ensemble learners* - em comparação com os seus classificadores base quando aplicados individualmente. Para este fim, propõe-se, ainda, uma arquitetura de sistema. Os experimentos foram realizados com dois conjuntos de dados distintos: dados gerados artificialmente e dados reais, disponibilizados publicamente.

Palavras-chave: Meta-learning. tráfego de rede. classificação. stacking. scikit-learn. ensemble learners.

Network traffic classification using meta-learning techniques

ABSTRACT

Network traffic classification has attracted attention and research efforts because a large number of areas benefit from such studies - from network security and management, quality of service (QoS), network traffic collection systems to intrusion detection (IDS).
Inspired by previous research, this work presents a comparative analysis between meta-learning approaches and individual classifiers to classify network traffic data. The most common techniques found in the literature are available from the *scikit-learn* library, for Python. In addition, an extra technique was implemented (*Stacking*), due to good results of this method disclosed in the literature.
With this in mind, we propose an experimental analysis of different meta-learning techniques - also known as *ensemble learners* - in comparison with their own base classifiers when used individually. To this end, we also propose a system architecture. The experiments were performed with two distinct data sets: artificially generated data and real data, publicly available.

Keywords: meta-learning, network traffic, classification, stacking, scikit-learn, ensemble learners.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANFIS	Adaptive Neural Fuzzy Inference System
DT	Decision Tree
DoS	Denial of Service
DDoS	Distributed Denial of Service
IDS	Intrusion Detection System
KNN	K-Nearest Neighbors
LGP	Linear Genetic Programming
QoS	Quality of Service
RF	Random Forest
SOM	Self-Organizing Maps
SVM	Suport Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

Este trabalho apresenta um estudo comparativo entre diferentes técnicas de *meta-learning* utilizando algoritmos de aprendizagem de máquina, quando aplicados para detecção de anomalias em tráfego de rede. Nas próximas subseções serão apresentados aspectos deste trabalho, tais como sua motivação, objetivos, contribuições e organização do documento que o descreve.

1.1 Motivação

O principal desafio na automação de detecção e classificação de anomalias é o fato de que anomalias podem abranger uma vasta gama de eventos: desde abuso de rede (como ataques DoS - *Denial of Service*, *port scans* e *worms*), falha de equipamentos até comportamento imprevisível do usuário (mudanças de demanda, *flash crowds*) (LAKHINA; CROVELLA; DIOT, 2005).

Com isso em mente, um desafio a se considerar é a dificuldade em definir permanentemente e precisamente um conjunto de anomalias possíveis para um dado domínio - novas características continuam a aparecer ao longo do tempo, por isso, sistemas de detecção de anomalias devem evitar restringir-se a um conjunto pré-definido de anomalias.

Neste contexto, utiliza-se classificadores de dados baseados em aprendizagem de máquina para identificar os mais variados tipos de anomalias. Trabalhos como (NGUYEN; ARMITAGE, 2008) e (MOORE; ZUEV, 2005) já conseguem detectar anomalias utilizando classificadores base individualmente. No entanto, sabe-se de algumas limitações do uso de classificadores individuais, o trabalho proposto em (SUTHAHARAN, 2014), por exemplo, discute algumas dessas limitações para classificação de tráfego de rede.

Sistemas que utilizam classificadores de dados baseados em aprendizagem de máquina podem aprender conceitos sem informação a priori sobre o domínio estudado. No entanto, se quisermos que nossos classificadores de tráfego consigam adquirir muitas habilidades e se adaptar a muitos ambientes, não podemos treinar cada habilidade em cada cenário a partir do zero. Em vez disso, precisamos que esses sistemas aprendam novas tarefas mais rapidamente, reutilizando suas experiências anteriores, ao invés de considerar cada tarefa isoladamente. Essa abordagem de aprender a aprender (meta-aprendizagem) é um passo fundamental para sistemas versáteis que podem aprender continuamente uma grande variedade de tarefas.

A área de *meta-learning* também é conhecida pelo termo *ensemble learning* (montagem de classificadores) e tem uma vasta gama de trabalhos que buscam encontrar os melhores métodos para construir esses classificadores (DIETTERICH, 2000). Este tópico tornou-se especialmente atraente baseado na premissa de que meta-classificadores são muitas vezes mais precisos do que os classificadores individuais que os compõem (DŽEROSKI; ŽENKO, 2004). Trabalhos como (REDDY; HOTA, 2013) e (WANG; GUAN; QIN, 2017) também exploram técnicas de *meta-learning* no contexto de classificação de tráfego de rede.

1.2 Objetivos

O principal objetivo deste trabalho é realizar um estudo comparativo entre diferentes técnicas de *meta-learning* e classificadores individuais, no âmbito de tráfego de rede. Assim, podemos determinar qual é a melhor técnica para se utilizar neste contexto.

Para isso, selecionamos quatro técnicas de *meta-learning* comumente apresentadas na literatura, definimos os classificadores base a serem utilizados, comparamos o desempenho dessas técnicas entre si e, ainda, com a execução dos mesmos classificadores base quando utilizados individualmente.

A avaliação do desempenho dessas técnicas é baseada nas métricas de acurácia, precisão e *recall* (quantidade de anomalias que o classificador é capaz de identificar). Para uma análise mais completa, avaliamos o número de verdadeiros e falsos positivos, bem como verdadeiros e falsos negativos identificados por cada técnica.

1.3 Contribuições

Dados os pontos mencionados nas subseções anteriores, as principais contribuições deste trabalho são listadas a seguir.

- Estudo do estado da arte e identificação das principais técnicas existentes para *meta-learning*;
- Uma arquitetura que classifica fluxos de rede, utilizando diferentes técnicas de combinação de classificadores base;
- Estudo comparativo entre a aplicação de técnicas de *meta-learning* e classificadores base, em relação à acurácia obtida.

1.4 Organização do documento

Este documento está organizado da seguinte maneira: o Capítulo 2 apresenta os conceitos principais da pesquisa, bem como a fundamentação teórica para este trabalho. O Capítulo 3 reúne detalhes sobre o projeto desenvolvido. O Capítulo 4 apresenta detalhes sobre a implementação do protótipo, experimentos e resultados. O Capítulo 5 traz os principais trabalhos relacionados encontrados na literatura. Finalmente, o Capítulo 6 traz as conclusões e considerações finais deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos abordados no trabalho aqui apresentado. Dessa forma, consideramos duas grandes áreas da computação - redes de computadores e inteligência artificial. Dentro do ramo de inteligência artificial, definimos os conceitos de aprendizagem de máquina (Seção 2.1), algoritmos supervisionados (Seção 2.1.1) e algoritmos não supervisionados (Seção 2.1.2), bem como a relevância de cada um no escopo deste trabalho. Ainda, no contexto de redes de computadores, abordamos os conceitos de anomalias em redes (Seção 2.2.1) e técnicas para detecção de anomalias em redes (Seção 2.2.2).

2.1 Aprendizagem de máquina

Aprendizagem de máquina é o campo da Inteligência Artificial que diz respeito a programas que aprendem por experiência (RUSSELL; NORVIG, 2003). Este ramo é proveniente de técnicas de reconhecimento de padrões e da ideia de que máquinas podem aprender sem que sejam explicitamente programadas para isso. Isto é, as máquinas são expostas a conjuntos de dados confiáveis e com potencial de repetição e, com base nisso, conseguem tomar decisões e/ou classificar novos dados.

A área de aprendizagem de máquina tem ganhado cada vez mais visibilidade por conta de fatores como o elevado crescimento do volume de dados, a variedade de múltiplos conjuntos de dados e a capacidade de processamento das máquinas atuais, que, atualmente, têm menor custo. Todos estes fatores contribuem para que a área ganhe cada vez mais força e consiga, com maior acurácia, analisar conjuntos de dados maiores, mais complexos e em uma velocidade maior.

O ramo de aprendizagem de máquina é dividido em duas áreas principais: aprendizagem supervisionada e aprendizagem não supervisionada, de acordo com o tipo de dados que recebem como entrada. Estas áreas são apresentadas nas próximas subseções.

2.1.1 Aprendizagem supervisionada

O exercício de utilizar dados de entrada para prever o valor dos dados de saída é chamado de aprendizagem supervisionada (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).

No ramo de reconhecimento de padrões, os dados de entrada são chamados de *features*, e os dados de saída são chamados de respostas ou predições.

Alguns dos algoritmos mais comuns desta área envolvem os chamados *Support Vector Machines* (SVM), *Naïve Bayes*, *K-Nearest Neighbors* e Árvores de Decisão (*Decision Trees*), apresentados nas subseções a seguir. Esses algortimos aparecem com frequência em problemas de classificação de tráfego de rede, como por exemplo nos trabalhos (WANG; GUAN; QIN, 2017) e (REDDY; HOTA, 2013).

2.1.1.1 Support Vector Machines (SVM)

A ideia essencial deste método é encontrar um hiperplano que separe, perfeitamente, os dados *n*-dimensionais em dois conjuntos (BOSWELL, 2002). Em outras palavras, dado um conjunto com *n features*, consideramos cada item do conjunto a ser classificado como um ponto em um espaço *n*-dimensional. O valor de cada coordenada será o valor do seu atributo correspondente. Assim, a classificação é feita de forma a encontrar uma linha que divida todos esses pontos em diferentes conjuntos de dados. Esta linha é chamada de hiperplano.

O desafio é, portanto, encontrar o hiperplano correto - que melhor descreve o modelo em questão.

2.1.1.2 Naïve Bayes

No contexto deste algoritmo, estamos interessados em encontrar a hipótese *h* com maior probabilidade de acontecimento, baseado em um conjunto de dados *d*. Para isso, utiliza-se o teorema de Bayes, que nos permite calcular a probabilidade de um acontecimento, baseado em conhecimento pré-adquirido. O teorema é mostrado abaixo.

$$P(h|d) = (P(d|h) * P(h))/P(d)$$

P(h|d) é a probabilidade posterior, que representa a probabilidade de uma hipótese *h* acontecer, dado que *d* é verdade. *P(d|h)* é a probabilidade de *d*, dado que *h* aconteceu. *P(h)* é a probabilidade a priori de *h*. Isto é, a probabilidade de *h* independente de *d*. *P(d)* é a probabilidade a priori de *d*. Isto é, a probabilidade de *d* independente de *h*.

Assim, o objetivo é calcular a probabilidade posteiror *P(h|d)*, a partir dos dados de entrada. Para tanto, utiliza-se uma tabela de frequência de dados, criando uma tabela de probabilidades de um dado evento ocorrer, para cada entrada. Utilizando o teorema de

Bayes, calcula-se a probabilidade posterior de cada evento e o evento com maior probabilidade é selecionado como resultado previsto.

2.1.1.3 Árvores de Decisão (Decision Treess)

Árvores de decisão classificam instâncias ordenando-as a partir da raiz até um nodo folha. Esse caminho (raiz-nodo-folha) fornece a classificação da instância (MITCHELL, 1997). O processo é repetido para todas as sub-árvores da instância, até que se obtenha uma classificação final. Ainda, essas árvores também podem ser representadas através de regras do tipo *if-then-else*, para melhor legibilidade.

Uma árvore de decisão é uma estrutura recursiva que expressa um processo sequencial de classificação. Nesse processo, um elemento descrito por um conjunto de atributos é atribuído a uma classe dentre um conjunto disjunto de classes. Cada folha da árvore representa uma classe, ao passo que cada nodo interno representa um teste relativo a um ou mais atributos. Para cada saída do teste, existe uma sub-árvore correspondente, pela qual o processo deve se repetir.

Para classificar um elemento, inicia-se pela raiz da árvore. Se o nodo atual é uma folha, classificamos o elemento em questão como membro da classe associada; senão, este nodo expressa mais um teste. O processo se repete pela sub-árvore seguinte e assim por diante, até que se atinja um nodo folha e, consequentemente, uma classe associada. (QUINLAN, 1987)

2.1.1.4 K-Nearest Neighbors (KNN)

Também chamado de KNN, este é um classificador simples, que armazena todos os casos disponíveis e classifica novos casos com base em medidas de similaridade com os casos existentes.

Para tanto, KNN assume que os dados estão em um *feature space*. Mais especificamente, que os pontos de dados estão em um espaço métrico. Os dados podem ser escalares ou, possivelmente, vetores multidimensionais. Já que os pontos estão em um *feature space*, eles têm uma noção de distância.

Essencialmente, cada um dos dados de treinamento consiste em um conjunto de vetores e rótulos da classe associada a cada vetor. Para isso, o algoritmo recebe um número *K*. Esse número decide quantos vizinhos (estes, por sua vez, são definidos baseados em uma métrica de distância) influenciam na classificação de um determinado ponto.

Portanto, dado um ponto x a ser classificado, encontra-se os K vizinhos mais próximos e efetua-se uma votação sobre os rótulos desses vizinhos. A classificação final é a classificação da maioria dos vizinhos envolvidos. Tipicamente, o valor de K é ímpar quando o número de classes é 2. Por exemplo, para $K = 5$ e vizinhos com rótulos *C1* e *C2*, no caso de termos 3 instâncias com o rótulo *C1* e duas com o rótulo *C2*, a instância a ser classificada terá o rótulo *C1*. A acurácia do modelo pode aumentar com o aumento do parâmetro K, no entanto, o custo computacional também aumenta.

Uma possível variação deste método é utilizar uma votação com pesos, onde cada instância tem um peso associado, geralmente associado à distância entre os pontos. Isto é, vizinhos mais próximos têm maior influência sobre o resultado do que os vizinhos mais distantes.

2.1.2 Aprendizagem Não-Supervisionada

O objetivo da aprendizagem não-supervisionada é modelar os dados de entrada de forma a aprender mais sobre os mesmos. Os algoritmos com essa característica são chamados de algoritmos não-supervisionados pois não há interferência ou influência no resultado final. Exemplos clássicos desse tipo de aprendizagem são os algoritmos *K-Means* (para problemas de *clusterização*) e *Self-Organizing Maps* (SOM), apresentados nas subseções a seguir.

2.1.2.1 K-Means

Esta é uma técnica de aprendizagem não-supervisionada, comumente utilizada para resolver problemas de *clusterização* - divisão em conjuntos. O algoritmo consiste em classificar um conjunto de dados em um determinado número k de subconjuntos.

Primeiramente, são definidos k elementos centrais, que correspondem aos elementos característicos de um determinado *cluster*. Em seguida, cada item do conjunto é avaliado e posicionado no *cluster* de seu item mais semelhante. Isto é, cada item é posicionado no *cluster* cuja média tenha menor distância em relação ao item - em geral, utiliza-se distância euclidiana para este cálculo. O processo é repetido em um grande laço, até que os elementos não troquem de lugar e, assim, temos os *clusters* finais.

2.1.2.2 Self-Organizing Maps (SOM)

Uma classe particularmente interessante de sistemas não-supervisionados é baseada em aprendizado competitivo de redes neurais. Neste contexto, os neurônios de saída competem entre si para serem ativados, com a restrição de que apenas um é ativado por vez. O neurônio que for ativado é chamado de *winner-takes-all* ou simplesmente o neurônio vencedor. Essa competição pode ser induzida através de conexões laterais inibidas (caminhos que têm um *feedback* negativo) entre os neurônios. O resultado é que estes neurônios são forçados a se organizar entre si, em busca de um resultado melhor. Por essa razão, esse sistema é chamado de Self-Organizing-Map (BULLINARIA, 2004).

Esta técnica permite representar dados multidimensionais em espaços com dimensões muito inferiores - geralmente, uma ou duas dimensões. Esse processo é, essencialmente, uma compressão de dados, conhecida como quantização de vetores (*vector quantization*). Com isso, o método de Self-Organizing Maps cria uma rede que armazena essas informações de forma que qualquer relação topológica dentro dos dados de treinamento sejam mantidas.

Portanto, *Self-Organizing Maps* reduzem as dimensões de um sistema produzindo um mapa que possui, geralmente, uma ou duas dimensões. Esse mapa representa as similaridades do dados e agrupa itens semelhantes no mesmo conjunto (PANG, 2003).

2.2 Detecção e classificação de tráfego em redes de computadores

Entende-se por tráfego de rede um fluxo de troca de informações entre um emissor e um receptor. Dessa forma, ao analisar diferentes tráfegos de rede, procuramos por características singulares nessa comunicação, características que foram estudadas e identificadas por estudos prévios. Neste trabalho, as características pelas quais vamos procurar foram investigadas em (SILVA et al., 2016) e (SILVA et al., 2015). Nesse contexto, um fluxo é formado pelos dados trocados entre uma comunicação definida pela tupla <*IP origem, porta origem, IP destino, porta destino, protocolo*>.

2.2.1 Tipos de anomalias em redes

Uma anomalia de rede é um desvio repentino do comportamento normal de uma rede. Algumas anomalias são propositalmente causadas com intenções maliciosas - como

o chamado DoS (*Denial of Service*) -, enquanto outras são acidentais, causados por consequências de uma rede sobrecarregada (AHMED; ORESHKIN; COATES, 2007).

Com isso em mente, é preciso utilizar métodos de identificação de anomalias com rápido tempo de resposta. Uma alternativa é a monitoração de tráfego de rede, permitindo a coleta de dados e identificação de anomalias, extraindo as informações relevantes.

As estatísticas de rede extraídas dos fluxos têm diferentes formas em diferentes anomalias; por isso, a dificuldade de estabelecer uma maneira única de identificar fluxos maliciosos. Algoritmos baseados em modelos pré-estabelecidos não oferecem portabilidade e são demasiadamente sensíveis a quaisquer mudanças na natureza do tráfego, o que permite que o fluxo monitorado possa ser classificado incorretamente. Por outro lado, algoritmos de aprendizagem, conseguem aprender as métricas mais relevantes e adaptar as possíveis variações de um fluxo considerado normal, e, portanto, são mais adequados (AHMED; ORESHKIN; COATES, 2007).

Um tipo de ataque muito comum, que aparece no conjunto de dados, é chamado de DDoS - *Distributed Denial of Service attack*. DoS consiste em sobrecarregar o sistema, com o intuito de consumir a máxima largura de banda do sistema ou, ainda, consumir recursos de um sistema alvo - geralmente, um ou mais servidores. DDoS é a versão de sistemas distribuídos deste ataque, que ocorre quando diversos agentes atuam colaborativamente para este fim.

2.2.2 Detecção e classificação de anomalias em redes

A área de detecção de anomalias corresponde a um conjunto de técnicas que são utilizadas para identificar anomalias no tráfego da rede (LAZAREVIC et al., 2003). Esta é uma área bastante emergente nas pesquisas de segurança da informação principalmente por conta do vasto uso da Internet (MARNERIDES; SCHAEFFER-FILHO; MAUTHE, 2014).

Ao longo do tempo, muitas pesquisas abordaram o assunto e recolheram características comuns a diferentes anomalias. Assim, o objetivo da maioria dos algoritmos de aprendizagem de máquina é identificar se, em um dado fluxo de pacotes, essas características estão presentes. No entanto, muitas dessas anomalias têm características muito semelhantes a fluxos de tráfego normais, o que dificulta a tarefa de classificar diferentes fluxos.

Uma anomalia é definida como um evento diferente do fluxo considerado nor-

mal (HASTIE; TIBSHIRANI; FRIEDMAN, 2009). Assim, as técnicas para detecção de ataques são subdivididas em duas categorias principais: baseada em anomalia (*anomaly-based*) e baseada em assinatura (*signature-based*) (HAMDI; GRIRA; BOUDRIGA, 2005), explicadas a seguir. No trabalho aqui apresentado, a técnica utilizada é baseada em anomalia.

Técnicas baseadas em anomalias consistem em monitorar a atividade do sistema e classificá-la entre normal ou anormal. O principal diferencial desta técnica é a capacidade de detectar possíveis novos tipos de anomalias. Isto é, dado que a técnica em questão é capaz de detectar qualquer tipo de atividade incomum na rede, através de heurísticas, não é necessário conhecer o comportamento esperado de um ataque já conhecido - característica de técnicas baseadas em assinatura. Dessa forma, o sistema que deve identificar as anomalias é treinado para diferenciar o uso comum de uma rede de um uso anômalo. Para isso, técnicas de aprendizagem de máquina, geralmente, são utilizadas. Estas técnicas são compostas por duas fases: a fase de treinamento, onde um perfil de atividade comum na rede é construído; e a fase de avaliação, quando os fluxos são comparados com o perfil de uso normal, para detectar se é uma anomalia ou não. O problema dessa abordagem são os falsos-positivos. Ou seja, muitos dos fluxos analisados são classificados como anomalias, quando, na verdade, são fluxos comuns de usuário.

Em contrapartida, técnicas baseadas em assinatura procuram por padrões no conjunto de dados analisado (GARCÍA-TEODORO et al., 2009). Isto é, analisam eventos e procuram por características de anomalias já conhecidas. Dessa forma, ao implementar técnicas desse tipo, obtemos bons resultados para anomalias específicas e bem conhecidas. No entanto, não conseguimos detectar novos tipos, mesmo que sejam construídos a partir de características de anomalias previamente conhecidos (GARCÍA-TEODORO et al., 2009).

Ainda, dependendo da fonte de informação que consideramos, um sistema de detecção de invasão de rede pode ser classificado como baseado em *host* (*host-based*) ou baseado em rede (*network-based*). Um sistema baseado em *host* analisa eventos relacionados ao sistema operacional, como identificadores de processos e chamadas de sistema. Por outro lado, sistemas baseados em rede analisam eventos de tráfego como volume, endereços IP, portas utilizadas, utilização de protocolos, etc (GARCÍA-TEODORO et al., 2009). Este trabalho tem foco nas técnicas baseadas em rede.

3 PROJETO DE UM SISTEMA BASEADO EM *META-LEARNING* PARA CLASSIFICAÇÃO DE TRÁFEGO DE REDE

Neste capítulo, é apresentada a proposta do trabalho que foi desenvolvido. A Seção 3.1 traz uma análise dos requisitos do sistema em questão. A Seção 3.2 apresenta os principais conceitos de *meta-learning*, técnicas encontradas na literatura e os desafios da área. Na Seção 3.3, encontra-se a formalização do problema e dos elementos que o compõem. Finalmente, na Seção 3.4, apresenta-se o projeto da solução proposta, bem como cada um de seus componentes.

3.1 Análise de requisitos

O principal desafio na automação e detecção de anomalias é o fato de que as anomalias podem abranger uma vasta gama de eventos - desde abuso de tráfego de rede até falha de equipamentos. Um sistema de detecção, portanto, deve ser capaz de detectar uma variedade de anomalias de diversas estruturas, identificando aquelas que, potencialmente, pertencem à mesma classe (LAKHINA; CROVELLA; DIOT, 2005).

A maioria dos sistemas de detecção de anomalias utiliza um único classificador para determinar se o tráfego de rede é anômalo ou não. No entanto, classificadores únicos não conseguem alcançar uma acurácia boa o suficiente, que não considere os falsos positivos (fluxos classificados como anômalos que, na verdade, são fluxos normais) e falsos negativos (PANDA; ABRAHAM; PATRA, 2012).

Assim surge a importância de experimentar técnicas de *meta-learning* para detecção de anomalias em tráfego de rede. A área de *meta-learning* também é conhecida pelo conceito de *ensemble learners*, e consiste em combinar diferentes classificadores, em busca de melhor desempenho.

No contexto deste trabalho, a detecção de anomalias é feita *offline* (a partir de dados previamente coletados) e utilizando classificadores supervisionados. Dessa forma, o processo consiste em, a partir de um conjunto de dados rotulados, compor um conjunto de treinamento para o sistema. Esse sistema, por sua vez, combina as classificações de diferentes classificadores base.

Sistemas que utilizam *meta-learning* têm mostrado resultados de sucesso em problemas tradicionais de classificação, tais como detecção de anomalias. Esses sistemas

são formados por classificadores como meta-árvores de decisão (TODOROVSKI; DŽEROSKI, 2003), redes neurais, *Support Vector Machines*, *K-Nearest Neighbors* (BRAZDIL et al., 2008), entre outros.

No entanto, para desenvolver um sistema de *meta-learning*, temos um conjunto de requisitos a se considerar. No caso aqui apresentado, porque utilizamos aprendizagem supervisionada, é importante que o conjunto de treinamento tenha as características necessárias para identificar uma anomalia em um tráfego de rede. Assim, cada fluxo de tráfego de rede é representado pelos seguintes dados: IP de origem, IP de destino, protocolo, duração do fluxo, e um conjunto de características calculadas a partir de estatísticas deste fluxo - essas características são chamadas de *features*. Dessa forma, é importante que o conjunto de *features* extraídas do conjunto de dados bruto seja representativo do domínio do sistema. Por isso, as *features* extraídas neste trabalho foram introduzidas e estudadas em (SILVA et al., 2015).

Portanto, o objetivo deste trabalho é estabelecer um sistema capaz de identificar anomalias em tráfegos de rede que satisfaçam aos critérios mencionados, utilizando técnicas de *meta-learning*. O trabalho foi desenvolvido visando maior acurácia ao detectar anomalias em um vasto conjunto de dados, quando comparado a sistemas de aprendizagem de máquina que utilizam classificadores individuais.

3.2 Meta-learning

Meta-learning pode ser definido como o aprendizado a partir de informações geradas por outros sistemas de aprendizado (CHAN; STOLFO, 1993). Assim, um sistema de *meta-learning* deve incluir um subsistema de aprendizado, que se adapta de acordo com a experiência adquirida. Essa experiência é obtida através da exploração de dados extraídos de outros sistemas de aprendizado ou, ainda, de diferentes domínios do problema (LEMKE; BUDKA; GABRYS, 2015).

A técnica de aprender pelas observações dos dados é chamada de aprendizagem indutiva. Dado um conjunto de exemplos já classificados, a tarefa da aprendizagem indutiva é formar modelos que descrevam as relações entre os dados e prever com precisão a classificação de instâncias futuras não classificadas; estes modelos também são chamados de classificadores.

No caso de aprendizagem indutiva, utilizando aprendizagem supervisionada, *meta-learning* significa, então, aprender a partir das classificações produzidas pelos sistemas e

predições destes classificadores em um conjunto de treinamento - o chamado *training set*. Neste contexto, um classificador é a saída de um sistema de aprendizagem indutiva, e uma predição (ou classificação) é a classe prevista indicada pelo classificador quando uma instância é fornecida (CHAN; STOLFO, 1993).

Meta-learning é bastante utilizado pois melhora a eficiência através de execuções paralelas de diferentes processos (no caso, programas com diferentes inteligências) sobre um mesmo conjunto de dados. Como os resultados variam de acordo com o conjunto utilizado, a técnica de *meta-learning* permite, potencialmente, obter resultados melhores, selecionando o melhor algoritmo para cada um dos conjuntos.

No contexto deste trabalho, os classificadores chamados de classificadores base são os algoritmos de classificação aplicados individualmente, como por exemplo *K-Nearest Neighbors* e *Support Vector Machine* (SVM), apresentados no capítulo anterior.

Finalmente, a tarefa de um sistema que aplica técnicas de *meta-learning* é combinar os diferentes classificadores, isto é, aprender uma regra de integração baseada no comportamento dos classificadores treinados. O novo desafio é, portanto, definir qual é a melhor técnica para combinar os diferentes classificadores. As técnicas estudadas para este trabalho são apresentadas na Seção 3.2.2.

3.2.1 Desafios

Pesquisas na área de aprendizagem de máquina têm mostrado maior acurácia quando *meta-learning* é utilizado - em comparação a classificadores base (REDDY; HOTA, 2013). No entanto, a técnica de *meta-learning* também enfrenta desafios. Estes desafios são descritos nesta subseção.

Um dos principais desafios da área de *meta-learning* é o dimensionamento de memória utilizada para a execução dos processos (PRODROMIDIS; CHAN; STOLFO, 2000). Dado que a maioria das abordagens considera um conjunto de dados na memória principal, para sistemas de grande porte, seria infactível armazenar todos os dados necessários. Por isso, existe a necessidade de definir restrições de tempo e memória, ao utilizar técnicas de *meta-learning*.

Outro ponto a se considerar é a dificuldade de interpretação da técnica e de seus resultados. Uma combinação de classificadores (bem como os respectivos resultados produzidos) é mais difícil de interpretar do que quando se utiliza um único classificador.

Além disso, para incentivar diferentes aspectos da aprendizagem, é importante que

os classificadores combinados não sejam correlacionados. Isto é, que esses classificadores produzam diferentes taxas de erro para o mesmo conjunto de dados. Dessa forma, será possível combinar os classificadores de forma que estes se complementem e, potencialmente, obtenham maior acurácia nos resultados.

Finalmente, determinar qual é o melhor algoritmo para um dado conjunto de informações é uma tarefa difícil, pois precisamos de métricas para avaliar o seu desempenho e, ainda, garantir que os algoritmos são comparáveis. No contexto deste trabalho, um algoritmo é considerado o melhor para um determinado conjunto de dados se este apresentou a melhor acurácia nos resultados. Isto é, se o algoritmo conseguiu identificar, com maior exatidão, se um fluxo de pacotes é anômalo ou não. Isso é determinado através da taxa de erro de cada um dos classificadores, quando são realizados os experimentos do trabalho. Estes experimentos consistem em classificar os dados com cada um dos classificadores base individualmente, bem como com diferentes técnicas de *meta-learning*. Em seguida, a partir das predições obtidas, calcula-se a taxa de erro de cada um dos métodos. O método que apresentar menor taxa de erro é, então, considerado o melhor.

As principais técnicas de combinação de classificadores relatadas na literatura estão definidas na subseção a seguir.

3.2.2 Técnicas de *meta-learning*

Meta-learning pode ser empregado com uma variedade de configurações e técnicas de combinação de classificadores. As principais técnicas de *meta-learning* são apresentadas nesta subseção.

3.2.2.1 Voting

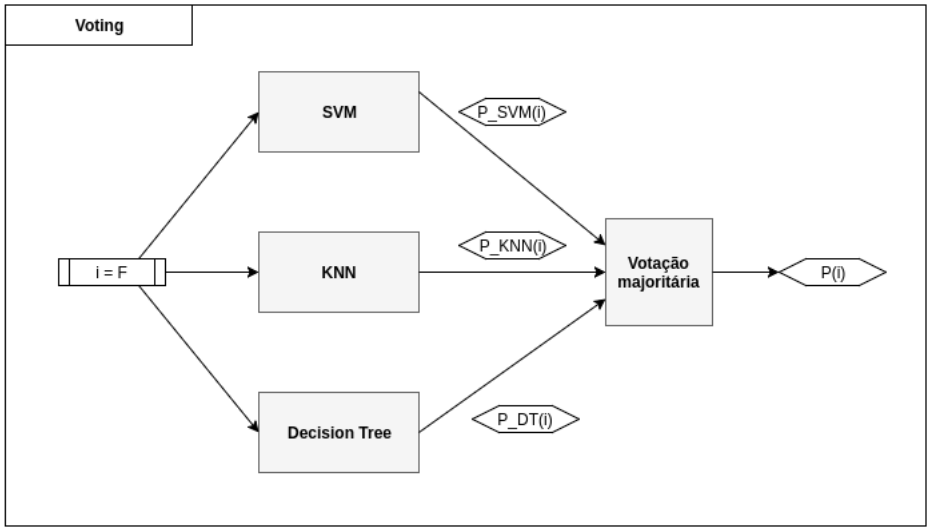
A técnica de votação é a abordagem mais simples para se combinar classificadores (MAGLOGIANNIS, 2007). Essencialmente, cada classificador base tem direito a um voto e a opção que tiver o maior número de votos é a predição final. Isto é, cada classificador gera uma predição, de forma que a mais recorrente será a predição final. Esta técnica é utilizada para classificação de dados.

O primeiro passo consiste em criar diferentes classificadores com um conjunto de dados de treinamento - por exemplo, utilizando SVM, *Naïve Bayes* e *K-Nearest Neighbors*. Então, após treinados, dada uma nova instância do problema, cada classificador

gera uma predição. A técnica de votação consiste em verificar, para uma dada instância x, qual foi a predição mais votada. Assim, a predição mais votada é eleita a predição final para essa instância.

Existe ainda uma variação desta técnica, que considera votos com um determinado peso, fornecendo tratamento preferencial para alguns classificadores votantes (PRODROMIDIS; CHAN; STOLFO, 2000). Nesse caso, cada classificador base tem um nível de influência sobre o resultado final. Assim, a saída desse sistema fornece a previsão mais votada pelos classificadores, considerando o peso de decisão de cada um.

Figura 3.1: Diagrama que ilustra o sistema de aprendizagem utilizando a técnica *Voting*. Na imagem, os classificadores base utilizados são SVM, KNN e *Decision Tree*.



Fonte: A autora

Cada classificador recebe um conjunto de treinamento que é formado por instâncias. Cada instância é composta pelo conjunto de *features* extraídas e seus respectivos valores, bem como a classificação da instância. O processo consiste, então, em efetuar uma votação majoritária sobre essas classificações.

Na Figura 3.1, o dado de entrada i representa uma instância do problema, que é representada pelo conjunto de *features* F calculado a partir dos dados disponíveis. Os classificadores base no exemplo são SVM, KNN e *Decision Tree*, portanto, cada um deles recebe como entrada a instância i e fornece como saída uma predição associada. Os valores *P_SVM(i)*, *P_KNN(i)* e *P_DT(i)* representam as classes atribuidas pelos classificadores individuais. Essas predições são consideradas em um processo de votação majoritária, que fornece a predição final *P(i)*, que corresponde a classe que será atribuída à instância i.

Existe ainda uma variação desta técnica, chamada de *Arbitraring*. A estratégia de *arbitraring* utiliza os artifícios da técnica de votação, agregando um juíz objetivo, cuja previsão é escolhida no caso de os restantes classificadores não atingirem um consenso. Neste caso, o árbitro, sendo também um classificador, escolhe uma previsão baseada na sua mas também considerando as previsões dos outros classificadores (PRODROMIDIS; CHAN; STOLFO, 2000).

3.2.2.2 Stacking

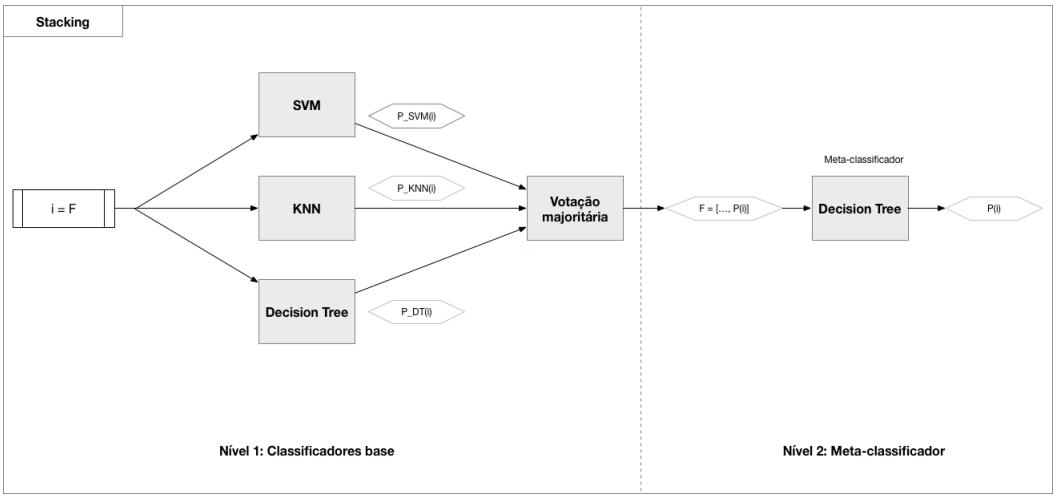
A técnica de empilhamento (*Stacking*) surgiu da necessidade de melhorar a escalabilidade e eficiência dos sistemas de aprendizagem. Assim, o objetivo é executar um conjunto de processos de aprendizagem e combinar o seu resultado final de forma não-linear e, com isso, melhorar a escalabilidade do sistema (MAGLOGIANNIS, 2007).

A ideia por trás do *Stacking* é utilizar as predições dos classificadores originais como atributos em um novo conjunto de treinamento. Essencialmente, este método combina a saída de um conjunto de classificadores através de um outro classificador (SEEWALD, 2002). Dessa forma, cria-se um novo conjunto de treinamento, que é composto, também, pelas predições dos classificadores do nível anterior; esse novo conjunto de treinamento é utilizado para treinar o nível subsequente do sistema, que também será um classificador.

O processo funciona com uma arquitetura em camadas. Cada um dos classificadores base é treinado com um conjunto de dados; a representação original das *features* é, então, estendida para incluir as predições desses classificadores. Camadas posteriores recebem como entrada as predições da camada imediatamente anterior e a saída é passada para a próxima camada. Um único classificador no nível mais acima produz a predição final (VILALTA; GIRAUD-CARRIER; BRAZDIL, 2010). A maioria das pesquisas desta área foca em arquiteturas com dois níveis ((WOLPERT, 1992), (BREIMAN, 1996a), (CHAN; STOLFO, 1998), (TING; WITTEN, 1997)). Neste caso, as predições dos classificadores base do primeiro nível são utilizadas para formar um novo conjunto de treinamento. Esse conjunto de treinamento é usado para treinar o *meta-learner* e, finalmente, fornecer a predição final de uma instância. A Figura 3.2 ilustra este conceito.

A Figura 3.2 apresenta um sistema que implementa a técnica de *Stacking*. A entrada do sistema é representada por $i = F$ que corresponde uma instância a ser classificada, representada pelo seu vetor de *features*. Esta entrada é classificada por cada um dos classificadores base e a predição de cada um deles (no caso, *P_SVM(i)*, *P_KNN(i)* e *P_DT(i)*)

Figura 3.2: Diagrama que ilustra o processo de aprendizagem utilizando a técnica *Stacking*. Na imagem, os classificadores base do primeiro nível são SVM, KNN e *Decision Tree*. O meta-classificador, representado no segundo nível, é o classificador *Decision Tree*.



Fonte: A autora

é enviada para um módulo onde se realiza uma votação majoritária. A classe mais votada é, finalmente, acrescentada ao vetor de *features* da instância *i* e enviada ao segundo nível do sistema. O meta-classificador, por sua vez, recebe este novo vetor de *features* e, com base no classificador utilizado (neste caso, *Decision Tree*), provê a classificação final da instância *i*, na figura, representada por *P(i)*.

Essa técnica é considerada uma forma de *meta-learning* porque a transformação do conjunto de teste transmite informação sobre as predições dos classificadores base. Dessa forma, se um classificador base aprendeu incorretamente sobre o conjunto de *features*, a segunda camada (*meta-learner*) é capaz de detectar este comportamento e, em conjunto com os outros classificadores base, aprender a forma correta.

Ainda, em relação a estratégia *Voting*, a técnica de *Stacking* permite maior liberdade na forma em que os classificadores são combinados, ao passo que a primeira (Seção 3.2.2) limita-se à combinação linear de classificadores.

3.2.2.3 Bootstrap Agreggating (Bagging)

Bagging é uma técnica que gera uma combinação de classificadores ao manipular o conjunto de treinamento fornecido a um classificador base. Consiste em selecionar um único classificador base e invocá-lo diversas vezes, utilizando diferentes conjuntos de treinamento (DIETTERICH, 2000).

Esta é uma das técnicas mais antigas de *meta-learning*, apresentada em (BREIMAN, 1996b). O diferencial do método é o fato de que são criados diferentes subconjuntos de treinamento. Esses subconjuntos são formados por instâncias selecionadas aleatoriamente no conjunto de treinamento original. Cada subconjunto de treinamento é utilizado para treinar um classificador base (WANG et al., 2011). As predições de cada um dos classificadores base são combinadas por uma votação majoritária.

O uso de *Bagging* é particularmente atraente quando o conjunto de dados disponível é de tamanho limitado. Para garantir que cada subconjunto contenha um número de amostras significativo e representativo do domínio do problema, porções relativamente grandes (entre 75% e 100% do tamanho do conjunto de treinamento original) dessas amostras são alocadas em cada subconjunto. Isso faz com que a intersecção desses subconjuntos de treinamento individuais seja significativamente grande, com muitas das mesmas instâncias aparecendo na maioria dos subconjuntos, e algumas instâncias aparecendo múltiplas vezes no mesmo subconjunto de treinamento - instâncias repetidas. Assim, para garantir a diversidade neste cenário, um classificador base relativamente instável é usado para que limites de decisão suficientemente diferentes possam ser obtidos para pequenas perturbações em diferentes conjuntos de dados de treinamento (WANG et al., 2011). Por esse motivo, essa técnica é conhecida por explorar as vulnerabilidades do classificador em questão e, com isso, melhorar suas predições. A Figura 3.4 ilustra este conceito.

Figura 3.3: Pseudo-algoritmo da técnica de *Bagging*

Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$;
Base learning algorithm L;
Number of learning rounds T.

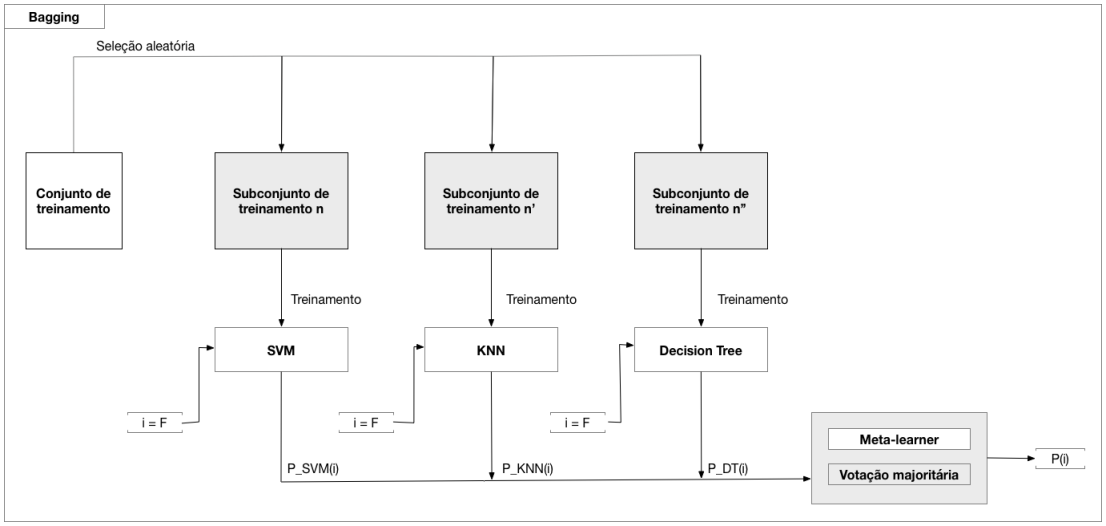
Process:
For $t = 1, 2, \cdots, T$:
$D_t = Bootstrap(D)$; % Generate a bootstrap sample from D
$h_t = L(D_t)$ % Train a base learner h_t from the bootstrap sample
end.

Output: $H(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} 1(y = h_t(x))$ % the value of $1(\alpha)$ is 1 if α is true % and 0 otherwise

Fonte: (WANG et al., 2011).

A Figura 3.4 mostra o processo de classificação de uma instância *i*, representada pelo conjunto de *features F*. O processo consiste em treinar, separadamente, diferentes classificadores base (no exemplo, SVM, KNN e *Decision Tree*) e, com isso, prover uma

Figura 3.4: Diagrama que ilustra o funcionamento da técnica de *Bagging*.



Fonte: A autora.

classificação para a nova instância *i*. O meta-classificador, por sua vez, consiste em uma regra de combinação (no exemplo, votação majoritária) para combinar estas diferentes predições e fornecer a classificação final *P(i)* da instância *i*.

3.2.2.4 Boosting

Esta técnica foi apresentada em (FREUND; SCHAPIRE, 1996) e (SCHAPIRE, 1990) e abrange um conjunto de métodos. Diferente da estratégia de *Bagging*, *Boosting* cria diferentes classificadores base através de um processo onde as instâncias do conjunto de dados recebem, sequencialmente, novos pesos - relevância ou prioridade da instância para que esta seja sorteada. Isto é, cada instância classificada incorretamente pelo classificador base anterior terá um peso maior na próxima rodada de treinamento.

A ideia básica de *Boosting* é aplicar, repetidamente, um classificador base para modificar versões do conjunto de dados; assim, produzindo uma sequência de classificadores base para um número de iterações previamente definido.

No primeiro passo, todas as instâncias são inicializadas com pesos uniformes. Após essa inicialização, cada iteração adapta um classificador base às instâncias de treinamento com os respectivos pesos. O erro é computado e o peso das instâncias classificadas corretamente é reduzido, ao passo que o peso das instâncias classificadas incorretamente é aumentado. O modelo final obtido pela técnica de *Boosting* é uma combinação linear de diversos classificadores base, ponderados pelo seu melhor desempenho (WANG et al., 2011). Este processo é repetido até que se atinja a acurácia desejada ou, até que não se

obtenha melhorias.

Muitas versões da técnica de *Boosting* foram apresentadas, no entanto, a mais utilizada é a proposta por (FREUND; SCHAPIRE, 1996), conhecida como *AdaBoost* (WANG et al., 2011). *AdaBoost* requer algoritmos menos instáveis em relação à técnica de *Bagging*, pois é capaz de gerar maior diversidade nos conjuntos de treinamento (através dos pesos introduzidos) (DIETTERICH, 2000). Portanto, utilizamos esta técnica para o desenvolvimento deste trabalho. O pseudo-algoritmo da técnica *AdaBoost* é apresentado abaixo.

Figura 3.5: Pseudo-algoritmo da técnica *AdaBoost*

Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$;
Base learning algorithm L;
Number of learning rounds T.

Process:
$D_1(i) = 1/m.$ % Initialize the weight distribution
For $t = 1, 2, \cdots, T$:
$h_t = L(D, D_t)$; % Train a base learner h_t from D using distribution D_t
$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i)]$; % Measure the error of h_t

$\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$; % Determine the weight of h_t

$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & if\ h_t(x_i) = y_i \\ \exp(\alpha_t) & f\ h_t(x_i) \neq y_i \end{cases}$ % Update the distribution, where Z_t is a

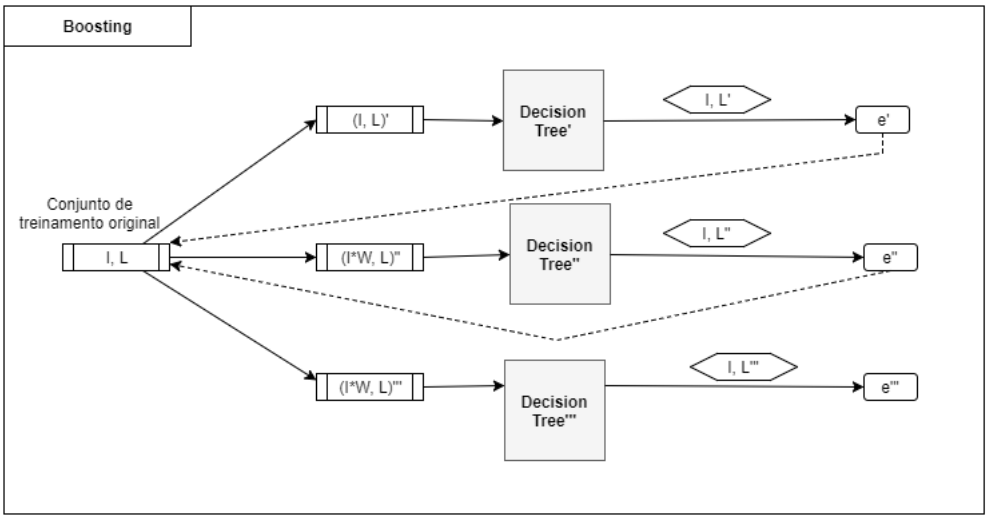
$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ %normalization factor with enables D_{t+1} to be a distribution

end.
Output: $H(x) = \text{sign}(f(x)) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$

Fonte: (WANG et al., 2011).

A Figura 3.6 mostra o funcionamento da técnica *Boosting*. No exemplo, o classificador base utilizado é *Decision Tree* e o conjunto de treinamento é formado por um vetor de instâncias *I* e um vetor das classificações associadas a essas instâncias *L*. Neste processo, ilustra-se o funcionamento da técnica com 3 iterações, sempre computando as predições do classificador *Decision Tree* e o erro associado ao classificador em questão (*e*). Este erro *e*, por sua vez, é utilizado para determinar os pesos das instâncias originais para gerar um novo conjunto de treinamento. No exemplo, este processo é repetido 3 vezes, para fins didáticos. Assim, para classificar uma nova instância *i*, esta serve de entrada para o último classificador gerado pelas iterações, no caso, *Decision Tree*, que provê a classificação final da instância.

Figura 3.6: Diagrama que ilustra o funcionamento da técnica de *Boosting*.



Fonte: A autora.

3.3 Formalização do problema

No trabalho aqui apresentado, utilizamos *meta-learning* para combinar diferentes classificadores e, potencialmente, aprimorar as predições associadas. Isto implica em aplicar diferentes algoritmos ao mesmo conjunto de dados, para que as predições resultantes sejam combinadas por técnicas de *meta-learning*. Os elementos da solução deste problema são definidos a seguir.

Cada fluxo f do conjunto de dados original é representado como uma tupla na forma:

$$d = [IP_{origem} : porta, IP_{destino} : porta, protocolo, timestamp, bytes, packets]$$

Dado um conjunto de treinamento *T* de tamanho *n* $\{(x_0, y_0), ..., (x_n, y_n)\}$, tal que x_i é um vetor de *features* representativo de uma instância *i* e y_i é a classificação (*label*) associada à instância *i*, um classificador supervisionado *C* procura a função $g : X \rightarrow Y$, onde *X* é o espaço de parâmetros de entrada (vetor de *features* e *label*) e *Y* é o espaço de classes possíveis para a instância *i*.

Neste contexto, uma técnica de *meta-learning* consiste em uma regra capaz de combinar diferentes classificadores *C*, em busca de uma única classificação para uma nova instância *i*.

Portanto, esse trabalho é fundamentado na seguinte hipótese: um sistema de *meta-*

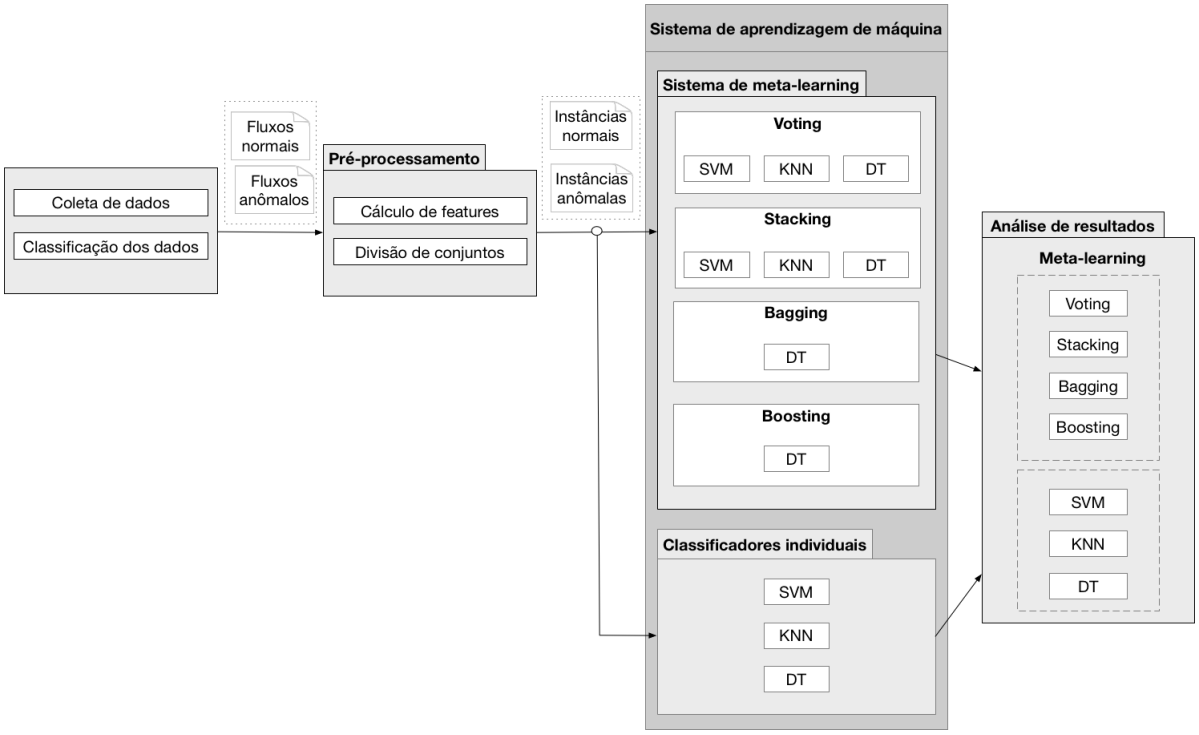
learning é capaz de gerar resultados mais precisos para classificação de anomalias em tráfego de redes, quando comparado aos classificadores aplicados individualmente.

Neste contexto, quatro técnicas de *meta-learning* são utilizadas, de forma a encontrar a técnica que melhor aproxima os resultados. Portanto, este trabalho é desenvolvido a partir da hipótese de que as técnicas de *meta-learning* conseguem classificar corretamente um maior número de instâncias do que os classificadores base quando aplicados individualmente.

3.4 Projeto

Esta seção apresenta detalhes sobre o projeto desenvolvido, considerando os aspectos apresentados nas subseções anteriores. O objetivo do projeto apresentado é desenvolver um sistema de aprendizagem de máquina capaz de detectar anomalias em fluxos de rede. As técnicas de *meta-learning* utilizadas são as chamadas de *Voting*, *Stacking*, *Bagging* e *Boosting*. A Figura 3.7 apresenta uma visão global do projeto.

Figura 3.7: Diagrama que ilustra o projeto proposto neste trabalho.



Fonte: A autora.

3.4.1 Coleta e classificação dos dados

A etapa inicial do projeto requer uma fase de classificação e coleta de dados. Dado que o projeto em questão utiliza aprendizagem supervisionada, é necessário um conjunto de dados já classificados. Isto é, um conjunto de fluxos já classificados como fluxo anômalo ou fluxo normal. É importante destacar que a classificação destes fluxos é pré-definida pela fonte dos dados - simulador ou base de dados da Internet.

Estes dados serão utilizados para as etapas de treinamento e validação do sistema de aprendizagem de máquina e podem ser resultantes de sistemas de simulação de rede ou tráfego real, disponível na Internet.

O conjunto de fluxos, com sua respectiva classificação, é armazenado em um arquivo. Este arquivo passa, então, por uma etapa de pré-processamento, com o objetivo de gerar as instâncias utilizadas no sistema de aprendizagem de máquina.

3.4.2 Pré-processamento

Esta etapa consiste em reunir os dados brutos coletados e a extrair as informações mais relevantes desses dados.

Considerando as definições estabelecidas na Seção 3.3, é possível agrupar informações de um mesmo fluxo (conjunto de instâncias *d*) e obter a seguinte representação de um fluxo:

$$d = [IP_o : porta, IP_d : porta, protocolo, data, timestamp, bytes, packets]$$

Para cada fluxo *d*, calcula-se um conjunto de *n features*. Essas, por sua vez, armazenadas em um vetor com o seguinte formato:

$$f = [f_1, f_2, f_3, ..., f_n]$$

O vetor de *features*, em conjunto com as informações do fluxo (*d*) formam uma instância *i* do problema. Essa instância tem o seguinte formato:

$$i = [d', f, label]$$

Finalmente, estas instâncias são divididas em dois grupos: instâncias que representam fluxos normais e instâncias que representam fluxos anômalos. Assim, gera-se um arquivo no formato CSV para cada grupo de instâncias. Estes arquivos fazem parte dos parâmetros de entrada para o sistema de aprendizagem de máquina (*meta-learning* e classificadores individuais).

3.4.3 Sistema de aprendizagem de máquina

Os dados processados na etapa anterior são enviados para o componente de aprendizagem de máquina, composto por dois componentes: sistema de *meta-learning* e o conjunto de classificadores individuais.

Considerando os objetivos deste trabalho, o sistema de aprendizagem de máquinha é responsável por receber as instâncias do problema e os parâmetros associados aos classificadores em questão e, então, encaminhar os resultados para o componente de análise de resultados, para conclusões finais.

A divisão do conjunto de dados entre conjunto de treinamento e conjunto de validação é feita a partir de parâmetros estabelecidos pelo usuário - por exemplo, 20% do conjunto total é destinado para treinamento e os restantes 80%, para validação dos modelos. Também são parâmetros definidos pelo usuário todos os parâmetros dos classificadores individuais e *ensemble learners* do sistema de *meta-learning* (bem como dos classificadores base associados).

3.4.3.1 Sistema de meta-learning

Após a divisão dos conjuntos de treinamento e validação, cada um dos componentes do sistema de *meta-learning* recebe o conjunto de instâncias $I_{treinamento}$, realizando o treinamento de seus modelos. Em seguida, o conjunto I_{teste} também é fornecido para cada modelo, esperando a classificação de cada uma das instâncias - este conjunto corresponde ao conjunto de validação dos modelos.

Com a predição final de cada uma das instâncias de I_{teste}, calcula-se as métricas para análisa dos resultados. Neste projeto, as métricas calculadas são precisão, recall, acurácia, taxa de erro, falsos positivos e negativos, verdadeiros positivos e negativos dos modelos. Esses valores são utilizadas para análise de resultados.

3.4.3.2 Classificadores individuais

Ao passo que cada um dos modelos de *meta-learning* está sendo treinado e utilizado para classificar novas instâncias, os classificadores individuais também passam pelo mesmo processo. Cada classificador recebe o conjunto $I_{treinamento}$, treina o modelo em questão e, ao receber o conjunto I_{teste} provê a classificação de cada uma das instâncias.

Da mesma forma, com a predição final de cada uma das instâncias de I_{teste}, calcula-se as mesmas métricas de cada um dos classificadores individuais. Esses valores também são utilizados na etapa de análise de resultados.

3.4.4 Análise de resultados

Nesta etapa, compara-se os resultados obtidos entre os modelos de *meta-learning* e classificadores individuais. Assim, é possível aceitar ou rejeitar a hipótese apresentada na Seção 3.3.

4 PROTOTIPAÇÃO E ANÁLISE EXPERIMENTAL

Este capítulo apresenta as principais etapas do processo de prototipação e análise experimental deste trabalho. A Seção 4.1 descreve detalhes do protótipo desenvolvido. A Seção 4.2 apresenta o processo de avaliação experimental e os resultados com dados artificiais (Seção 4.2.1) e com dados reais (Seção 4.2.2). Por fim, a discussão dos resultados é feita na Seção 4.2.3.

4.1 Protótipo

O protótipo foi desenvolvido em seis etapas principais: extração de features, escolha dos classificadores base, desenvolvimento do sistema de *meta-learning*, implementação da técnica *Stacking*, implementação dos classificadores individuais e avaliação estatística. Cada uma dessas etapas é descrita em detalhes a seguir.

4.1.1 Cálculo de features

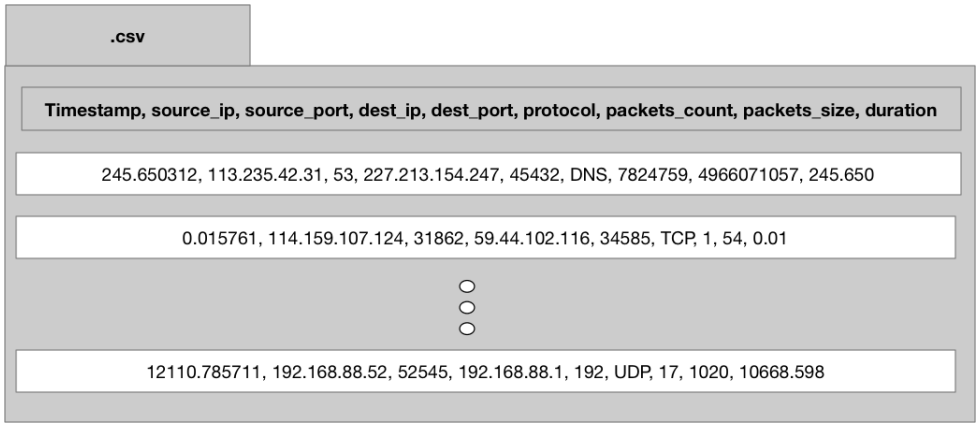
O conjunto de dados brutos do problema é formado por fluxos de tráfego de rede. Cada um destes fluxos é representado pelo seguinte conjunto de informações: *timestamp*, IP de origem, porta de origem, IP de destino, porta de destino, protocolo, número de pacotes (*packets_count*), tamanho total dos pacotes (*packets_size*) e duração. Conforme foi apresentado na Seção 3.4, para a etapa do cálculo de *features*, utiliza-se os dados dos fluxos, como ilustra a Figura 4.1.

Para interpretar cada fluxo com mais detalhes, foram calculadas um conjunto de características adicionais para cada fluxo. Essas características são chamadas de *features* e são calculadas a partir dos dados diponíveis para cada fluxo e foram estudadas em (SILVA et al., 2015), trabalho que explora uma variedade de informações que podemos extrair de um conjunto de fluxos de tráfego de rede. No entanto, dadas as informações disponíveis no conjunto de dados que utilizamos para estes experimentos e o tempo disposto para o projeto, só foi possível calcular 4 dessas características, que são mostradas na Tabela 4.1.

Com isso, cada fluxo passa a ser representado pelas seguintes *features*:

- número de pacotes (*packets_count*);
- tamanho total dos pacotes (*packets_size*);

Figura 4.1: Exemplo de arquivo de entrada para o cálculo de features adicionais.



Fonte: A autora

Tabela 4.1: Features adicionais calculadas para cada fluxo.

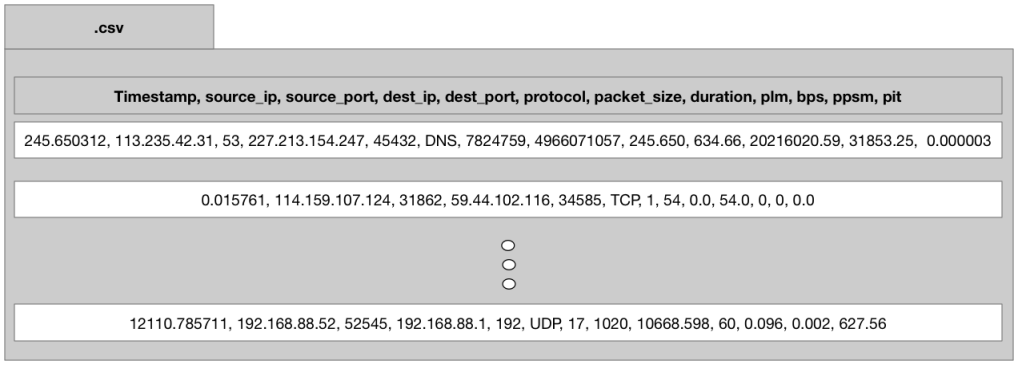
Feature	**Descrição**	**Cálculo**
packet_length_mean	Tamanho de um pacote, em média, do fluxo.	*packet_lenght_mean = packets_size / packets_count*
bytes_per_second_mean	Quantidade de bytes enviados por segundo, em média.	*bytes_per_second_mean = packets_size / duration*
packets_per_second_mean	Quantidade de pacotes enviados por segundo, em média.	*packets_per_second_mean = packets_count / duration*
packet_interarrival_time	Tempo decorrido entre a chegada de dois pacotes. Este valor é calculado com base em uma estimativa	*packet_interarrival_time = duration / packets_count*

- duração (*duration*);
- tamanho médio de um pacote (*packet_lenght_mean*);
- quantidade de bytes, em média, enviados por segundo (*bytes_per_second_mean*);
- quantidade de pacotes, em média, enviados por segundo (*packets_per_second_mean*);
- tempo entre a chegada de dois pacotes, em média (*packet_interarrival_time*).

O cálculo dessas *features* foi feito por um script escrito em Python 3.6, que recebe como entrada um arquivo no formato CSV, com as informações coletadas sobre cada um dos fluxos. Conforme mostra a Figura 4.1, uma instância a ser classificada corresponde a um fluxo, que por sua vez, possui um conjunto de informações que o descrevem, formado por 9 dados. Esses 9 dados são utilizados para calcular as *features* adicionais. O cálculo de cada uma dessas *features* também é mostrado na Tabela 4.1.

Ao final desta etapa, é gerado um novo arquivo CSV, no mesmo formato do arquivo de entrada, adicionando apenas as *features* adicionais calculadas, relativas ao respectivo fluxo. Este é o arquivo utilizado como base para o sistema de aprendizagem de máquina e é ilustrado na Figura 4.2.

Figura 4.2: Exemplo de arquivo de saída do cálculo de features adicionais. Os valores identificados por *plm*, *bps*, *ppsm* e *pit* representam, respectivamente, as features *packet_length_mean*, *bytes_per_second_mean*, *packets_per_second_mean* e *packet_interarrival_time*.



Fonte: A autora

4.1.2 Escolha dos classificadores base

Conforme discutido no Capítulo 3, cada uma das técnicas de *meta-learning* combina o resultado de um conjunto de classificadores base. Desta forma, é importante que os classificadores base utilizados sejam pouco relacionados. Isto é, que explorem diferentes características de um conjunto de dados, utilizando as mais variadas técnicas, com diferentes taxas de erro.

Os classificadores base utilizados foram *Support Vector Machine* (SVM), *Decision Tree* (DT) e *K-Nearest Neighbors* (KNN). A escolha destes classificadores se deu ao fato de serem algoritmos pouco relacionados - com diferentes taxas de erro para o mesmo conjunto de dados e porque são algoritmos altamente utilizados na literatura no contexto de *meta-learning* (NGUYEN; ARMITAGE, 2008), (REDDY; HOTA, 2013), (WANG; GUAN; QIN, 2017).

4.1.3 Desenvolvimento do sistema de meta-learning

Para a implementação do protótipo, foi utilizada a biblioteca *scikit-learn* [1] na versão 0.19.1, para Python (versão 3.6). Essa biblioteca contém a implementação dos classificadores base utilizados e facilitou o desenvolvimento do projeto, dado o tempo que era disposto. A implementação do meta-classificador variou de acordo com a técnica

[1] http://scikit-learn.org/stable/

utilizada, como é explicado a seguir.

Considerando que um dos objetivos deste trabalho é estudar diferentes técnicas de *meta-learning* e verificar qual técnica apresenta melhor desempenho no problema de classificação de tráfego, quatro técnicas foram selecionadas para este estudo. Conforme é mostrado no Capítulo 3, técnicas muito relevantes na literatura são *Voting*, *Stacking*, *Bagging* e *AdaBoost*; portanto, foram escolhidas para a avaliação experimental . A biblioteca *scikit-learn* disponibiliza técnicas como *Voting*[2], *Bagging* [3] e *AdaBoost* [4] prontas para utilização e, portanto, apenas a técnica *Stacking* teve de ser implementada.

Os parâmetros utilizados para cada *ensemble learner* cuja implementação de *scikit-learn* foi utilizada são mostrados nas tabelas 4.2, 4.3 e 4.4. Parâmetros opcionais não são mostrados nestas tabelas e podem ser consultados na documentação da biblioteca.

Tabela 4.2: Parâmetros utilizados para a técnica *Voting*.

Parâmetro	**Valor**	**Descrição**
estimators	*svm_classifier, knn_classifier, dt_classifier*	Classificadores base utilizados. Neste caso, utiliza-se os mesmos classificadores individuais selecionados.
weights	*None*	*weights = None* significa que o voto de todos os classificadores tem o mesmo peso.

A utilização da biblioteca *scikit-learn* é bastante simples. Para utilizar cada uma das técnicas é necessário, em primeiro lugar, treinar os diferentes classificadores base, individualmente, com um conjunto de dados de treinamento. Com os classificadores base já treinados, gera-se uma instância do chamado *ensemble learner*, que também passa pelo processo de treinamento com o mesmo conjunto de dados. Por fim, o *ensemble learner* gerado retorna suas predições para um dado conjunto de avaliação.

No caso do trabalho aqui apresentado, a implementação dos classificadores base utilizados também é fornecida pela biblioteca. Esses classificadores são algoritmos supervisionados e, portanto, recebem como argumento não apenas o conjunto de treinamento,

[2]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
[3]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
[4]http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Tabela 4.3: Parâmetros utilizados para a técnica *AdaBoost*.

Parâmetro	**Valor**	**Descrição**
base_estimator	*dt_classifier*	Classificador base a ser utilizado. Neste caso, é a mesma *Decision Tree* utilizada individualmente.
n_estimators	15	Número de iterações do processo de *boosting*.
algorithm	*'SAMME'*	Uso do algoritmo discreto de *Boosting* (pois as classes em questão são 0 ou 1).

Tabela 4.4: Parâmetros utilizados para a técnica *Bagging*.

Parâmetro	**Valor**	**Descrição**
base_estimator	*dt_classifier*	Classificador base a ser utilizado. Neste caso, é a mesma *Decision Tree* utilizada individualmente.
n_estimators	10	Número de iterações do processo de *bagging*.

mas também a classe associada a cada item do conjunto de treinamento - isto é, se o fluxo é considerado anômalo ou normal. Uma instância do *ensemble learner* é criada com os classificadores base selecionados e, em seguida, é treinada com o conjunto de treinamento. Por fim, a instância retorna as predições para cada um dos itens do conjunto de avaliação.

Também é importante mencionar que cada instância do classificador base é criada com base nos parâmetros de cada classificador. Por exemplo, para o classificador *Decision Tree*, deve-se definir qual é a profundidade máxima da árvore. Assim, os argumentos utilizados para cada um dos classificadores são mostrados nas tabelas 4.5, 4.6 e 4.7. Parâmetros opcionais, permitidos pela biblioteca *scikit-learn*, não são apresentados nestas tabelas e podem ser consultados na documentação da biblioteca [5].

Tabela 4.5: Parâmetros utilizados para o classificador SVM.

Parâmetro	**Valor**	**Descrição**
kernel	*'linear'*	Especifica o tipo de *kernel* utilizado no algoritmo. Neste contexto, um *kernel* é uma função de similaridade.
max_iter	-1	O valor -1 define que não há limite de iterações do solver.

Tabela 4.6: Parâmetros utilizados para o classificador KNN.

Parâmetro	**Valor**	**Descrição**
N	3	Número de vizinhos a se considerar no cálculo
weights	*'distance'*	Função de pesos utilizada nas predições. O valor *'distance'* corresponde a ponderar as instâncias de acordo com o inverso da distância. Assim, vizinhos mais próximos têm maior influência do que vizinhos mais afastados da instância em questão.
algorithm	*'brute'*	Algoritmo utilizado para calcular as vizinhas mais próximas. O valor *'brute'* corresponde a uma busca de força bruta.
p	2	Define qual o tipo de distância utilizada. *p = 2* indica o uso da distância euclidiana.

Os parâmetros apresentados nas tabelas 4.5, 4.6 e 4.7 foram determinados com base em uma série de testes cujo objetivo era determinar quais eram os argumentos com os quais se obtia melhores resultados.

[5] http://scikit-learn.org/stable/

Tabela 4.7: Parâmetros utilizados para o classificador *Decision Tree*.

Parâmetro	**Valor**	**Descrição**
criterion	*'entropy'*	Função para determinar a qualidade de uma divisão. O valor 'entropy' é utilizado para maximizar o ganho de informação.
splitter	*'best'*	Estratégia utilizada para selecionar uma divisão a cada nodo. O valor 'best' indica que a melhor divisão será a selecionada.
max_depth	*None*	Profundidade máxima da árvore. O valor None indica que não há limite estabelecido.

4.1.4 Implementação da técnica *Stacking*

Para uma análise mais robusta das técnicas de *meta-learning*, a técnica adicional implementada foi *Stacking*. A implementação dessa técnica segue o que mostra a literatura e foi apresentado na Seção 3.2.2.2.

O nível 1 consiste na utilização clássica dos classificadores individuais: cada classificador provê, para cada instância, uma classificação associada. Os classificadores base utilizados neste nível foram os mesmos utilizados pelas outras técnicas de *meta-learning* (*Voting*, *AdaBoost* e *Bagging*), para validar o processo de comparação, - isto é, SVM, KNN e *Decision Tree*. Dessa forma, cada instância do conjunto possui três classes associadas, que correspondem a classificação fornecida por cada um dos classificadores base utilizados.

No entanto, cada instância só pode ter uma classe associada e, portanto, utiliza-se uma regra de combinação para determinar qual será a classificação utilizada. A regra de combinação utilizada é a votação majoritária, que consiste em considerar cada classificação como um voto, e a classe mais votada é reconhecida como a classificação final. Em outras palavras, supondo que os classificadores SVM e KNN tenham classificado uma dada instância *x* como *"fluxo normal"* e, por sua vez, o classificador *Decision Tree* tenha classificado como *"fluxo anômalo"*. Seguindo a lógica de votação majoritária, a instância *x* é classificada como *"fluxo normal"* e essa classificação é acrescentada ao vetor de features da instância *x*.

Em seguida, no nível 2, recebe-se o novo vetor de *features* associado à instância (agora com a classificação obtida no nível 1). O nível 2, por sua vez, consiste em um único classificador, responsável por aprender a partir do que foi aprendido no nível 1 (com base no vetor de features gerado). Este classificador é o chamado meta-classificador (ou *meta-learner*). Para o protótipo desenvolvido, o classificador escolhido para o nível 2 foi *Decision Tree*, por apresentar bons resultados como meta-classificador na literatura

(REDDY; HOTA, 2013). Por sua vez, o meta-classificador recebe o vetor de *features* de cada instância e, com base nele, provê a respectiva classificação final. Essa classificação é analisada nos resultados dos experimentos, comparada com as outras técnicas.

4.1.5 Implementação dos classificadores individuais

Para fins de avaliação do protótipo, também foi analisado o comportamento dos classificadores individuais. Isto é, como são os resultados dos classificadores SVM, KNN e *Decision Tree* quando aplicados isoladamente a um conjunto de dados.

Mais uma vez, a biblioteca *scikit-learn* foi utilizada. Para garantir uma avaliação justa, os classificadores foram criados com os mesmo parâmetros de quando eram utilizados com os *ensemble learners*.

4.1.6 Avaliação estatística

Para melhor avaliar cada uma das técnicas, foram extraídas algumas métricas do resultados obtidos. Os dados foram extraídos no próprio sistema implementado e correspondem ao número de falsos positivos, número de falsos negativos, número de verdadeiros positivos, número de verdadeiros negativos, *recall*, precisão e acurácia média, para um dado número de repetições dos experimentos (YINGCHAREONTHAWORNCHAI et al., 2016).

No caso aqui apresentado, uma classificação positiva corresponde a um fluxo anômalo, ao passo que uma classificação negativa corresponde ao fluxo normal. Portanto, o número de falsos positivos corresponde ao número de instâncias consideradas fluxo normal que foi classificado como fluxo anômalo. Da mesma forma, o número de verdadeiros positivos corresponde ao número de instâncias consideradas anomalias que foram classificadas como, de fato, anomalias. Analogamente, calculamos os número de falsos negativos e número de verdadeiros negativos.

A métrica chamada de *recall* corresponde a quantidade de casos positivos (fluxos anômalos) que o classificador foi capaz de identificar. Ainda, a métrica de precisão corresponde a quantidade de predições positivas que estavam corretas, isto é, fluxos anômalos que eram, de fato, anomalias.

4.2 Avaliação experimental

Para fins de avaliação, o sistema implementado recebe como parâmetro o tamanho do conjunto de treinamento e o tamanho do conjunto de avaliação. Isto é, dado um conjunto de dados, qual a porcentagem desse conjunto representa o que será utilizado na fase de treinamento e o que será utilizado para a avaliação do modelo. Para os experimentos apresentados nesta seção, o conjunto de treinamento corresponde a 30% do conjunto de dados, e os restantes 70% formam o conjunto de avaliação.

As Seções 4.2.1 e 4.2.2 mostram, respectivamente, os resultados obtidos com conjuntos de dados artificiais e conjuntos de dados reais. Finalmente, Seção 4.2.3 propõe uma discussão sobre os resultados encontrados.

4.2.1 Experimentos com dados artificiais

O conjunto de dados destes experimentos é proveniente do trabalho (SILVA et al., 2016) e é composto por 41 fluxos simulados pela ferramenta Mininet [6]. A topologia utilizada para gerar esses dados é composta por 100 *hosts* e 11 *switches*, cenário escolhido devido a ataques similares estudados pelos autores (SILVA et al., 2016).

Destes 41 fluxos, 9 são anomalias geradas pela ferramenta *scapy* [7]; os restantes 32 fluxos são considerados tráfego normal. Para simular o comportamento de um usuário comum, gerou-se um cenário onde os usuários assistem a um vídeo por um determinado tempo, param o vídeo e, em seguida, acessam alguma página web. As anomalias são compostas por um conjunto de ataques do tipo *port scanning* e *DDoS* simulados pela ferramenta *scapy*. Mais detalhes sobre este conjunto de dados podem ser encontrados em (SILVA et al., 2016).

Seguindo a lógica do sistema implementado, o conjunto de dados foi separado em dois subconjuntos: subconjunto de treinamento e subconjunto de avaliação. Cada subconjunto possui 10 e 31 fluxos, respectivamente, respeitando a distribuição de fluxos anômalos e normais. O conjunto de testes é formado por 3 instâncias anômalas e 7 instâncias normais, ao passo que o conjunto de avaliação é composto por 6 instâncias anômalas e 25 instâncias normais.

Os experimentos foram executados 50 vezes, com o objetivo de analisar a variação

[6] http://mininet.org

[7] http://www.secdev.org/projects/scapy/

das predições. Os resultados são mostrados na Tabela 4.8 e Figuras 4.3, 4.4 e 4.5.

Tabela 4.8: Resultados dos experimentos com dados artificiais.

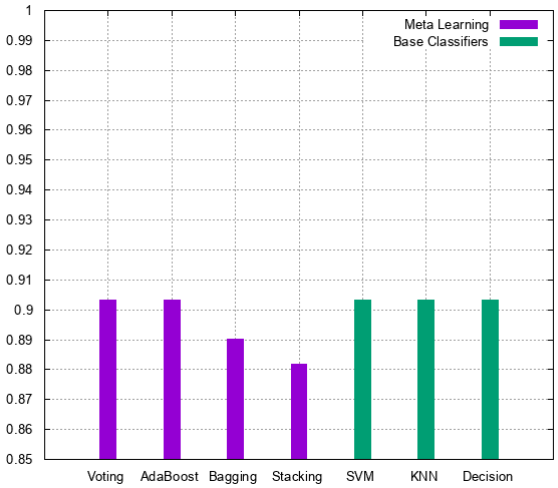
		Voting	**AdaBoost**	**Bagging**	**Stacking**	**SVM**	**KNN**	**Decision Tree**
Falsos negativos	Média	0	0	0.76	1.32	0	0	0
	Variância	0	0	4.2624	6.1776	0	0	0
	Desvio padrão	0	0	2.0646	2.4855	0	0	0
Falsos positivos	Média	3	3	2.64	2.34	3	3	3
	Variância	0	0	0.9504	1.5444	0	0	0
	Desvio padrão	0	0	0.9749	1.2427	0	0	0
Verdadeiros negativos	Média	21	21	21.36	21.66	21	21	21
	Variância	0	0	0.9504	1.5444	0	0	0
	Desvio padrão	0	0	0.9749	1.2427	0	0	0
Verdadeiros positivos	Média	7	7	6.24	5.68	7	7	7
	Variância	0	0	4.2624	6.1776	0	0	0
	Desvio padrão	0	0	2.0645	2.4855	0	0	0
Recall	Média	1	1	0.8914	0.8114	1	1	1
	Variância	0	0	0.0870	0.1261	0	0	0
	Desvio padrão	0	0	0.2950	0.3551	0	0	0
Precisão	Média	0.7	0.7	0.6960	0.7660	0.7	0.7	0.7
	Variância	$1.23 * 10^{-32}$	$1.23 * 10^{-32}$	0.0268	0.0154	$1.23 * 10^{-32}$	$1.23 * 10^{-32}$	$1.23 * 10^{-32}$
	Desvio padrão	$1.11 * 10^{-16}$	$1.11 * 10^{-16}$	0.1637	0.1242	$1.11 * 10^{-16}$	$1.11 * 10^{-16}$	$1.11 * 10^{-16}$
Acurácia	Média	0.9032	0.9032	0.8903	0.8819	0.9032	0.9032	0.9032
	Variância	$1.1093 * 10^{-31}$	$1.1093 * 10^{-31}$	0.0012	0.0016	$1.1093 * 10^{-31}$	$1.1093 * 10^{-31}$	$1.1093 * 10^{-31}$
	Desvio padrão	$3.33 * 10^{-16}$	$3.33 * 10^{-16}$	0.0353	0.0401	$3.33 * 10^{-16}$	$3.33 * 10^{-16}$	$3.33 * 10^{-16}$
Taxa de erro	Média	0.0968	0.0968	0.1097	0.1181	0.0968	0.0968	0.0968
	Variância	$1.73 * 10^{-33}$	$1.73 * 10^{-33}$	0.0012	0.0016	$1.73 * 10^{-33}$	$1.73 * 10^{-33}$	$1.73 * 10^{-33}$
	Desvio padrão	$4.16 * 10^{17}$	$4.16 * 10^{17}$	0.0353	0.040	$4.16 * 10^{17}$	$4.16 * 10^{17}$	$4.16 * 10^{17}$

Observado a Tabela 4.8 e as Figuras 4.3, 4.4 e 4.5 vemos que os classificadores individuais são bastante correlacionados - isto é, fornecem os mesmos resultados para o mesmo conjunto de dados. Pelo mesmo motivo, a técnica *Voting* tem resultados idênticos àqueles obtidos com os classificadores individuais - afinal, utilizamos votação majoritária entre as predições.

Conforme é mencionado no Capítulo 3, o fato de os classificadores serem bastante correlacionados é bastante prejudicial para o desempenho dos *ensemble learners*, o que é verificado com as técnicas *Bagging* e *Stacking*, que têm pior desempenho. Essas técnicas trouxeram benefícios ao conseguir reduzir o número de falsos positivos e, consequentemente, aumentar o número de verdadeiros negativos. Entretanto, também aumentaram o número de falsos negativos e, por consequência, o número de verdadeiros positivos. Estes fatores, aliados, reduziram a acurácia das técnicas e a taxa de erro associada.

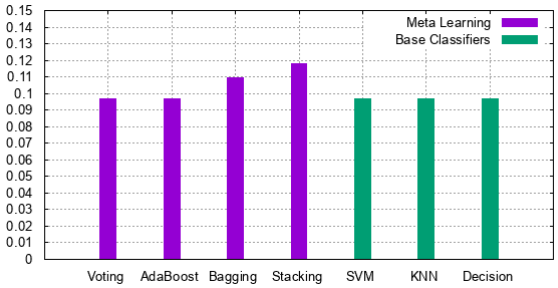
No caso da técnica *AdaBoost*, segundo (QUINLAN, 1996), a principal razão de falha (ou não melhoria) quando se utiliza *Adaboost* é *overfitting*. Isto é, quando se utiliza um grande número de iterações para gerar uma versão melhor do classificador, acabamos gerando um classificador demasiadamente complexo, o que reduz significativamente a acurácia em relação a um classificador individual. Neste caso, utilizando o classificador *Decision Tree*, 15 iterações após, ainda não foi possível melhorar o seu desempenho, o

Figura 4.3: Gráfico que ilustra a acurácia obtida por cada método nos experimentos com dados artificiais.



Fonte: A autora

Figura 4.4: Gráfico que ilustra a taxa de erro obtida por cada método nos experimentos com dados artificiais.



Fonte: A autora

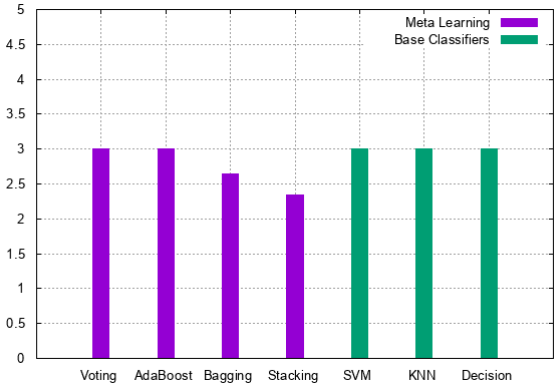
que indica provável overfitting dos dados.

Finalmente, é importante considerar que o conjunto de dados artificiais tem pouca representatividade do problema, o que não garante melhorias no desempenho dos *ensemble learners*.

4.2.2 Experimentos com dados reais

Devido às limitações de dados gerados artificialmente, também foram feitos experimentos com dados reais. Os dados utilizados nesta etapa são fluxos extraídos de pacotes de rede disponibilizados publicamente pela 4SICS (Conferência Industrial de Cyber Se-

Figura 4.5: Gráfico que ilustra a quantidade de falsos positivos obtida por cada método nos experimentos com dados artificiais.



Fonte: A autora

curity)[8]. Esses pacotes são considerados de fluxo normal e, portanto foram utilizados para representar tais fluxos. Para gerar os fluxos a partir desses pacotes, foi utilizada uma ferramenta *open source* chamada *pcap converter*[9], que gera um arquivo de texto com os fluxos agregados.

Para representar possíveis anomalias, os dados utilizados são provenientes do trabalho (SANTANNA et al., 2015). Este trabalho fez experimentos com bots que realizam ataques DDoS. Os autores disponibilizam publicamente os dados dos pacotes (arquivos com extensão .pcap) na plataforma Simple Web [10]. Os fluxos destes tráfegos também foram gerados a partir dos pacotes pela ferramenta *pcap converter*[11].

No entanto, devido à dificuldade de encontrar dados de tráfego anômalo disponíveis na Internet, o conjunto de dados resultante é bastante desbalanceado - 13749 fluxos normais e apenas 7 fluxos anômalos. Para contornar este problema, foram selecionados aleatoriamente apenas 100 fluxos normais, resultando em conjunto de 107 fluxos.

Seguindo a lógica do sistema implementado, o conjunto de dados foi separado em dois subconjuntos: subconjunto de treinamento e subconjunto de avaliação. Utilizando a divisão de 30% do conjunto de dados para treinamento, e os restantes 70% para avaliação, o conjunto de treinamento é composto por 2 fluxos anômalos e 30 fluxos normais, ao passo que o conjunto de avaliação contém 5 fluxos anômalos e 70 fluxos normais.

Semelhante ao que foi realizado com os dados artificiais, os experimentos foram

[8] http://www.netresec.com/?page=PCAP4SICS

[9] https://github.com/hbhzwj/pcap-converter

[10] https://www.simpleweb.org/wiki/index.php/TracesBooters_-_An_analysis_of_DDoS-as-a-Service_Attacks

[11] https://github.com/hbhzwj/pcap-converter

executados 50 vezes, com o objetivo de avaliar a variação das predições. Os resultados são mostrados na Tabela 4.9 e Figuras 4.6, 4.7 e 4.8.

Tabela 4.9: Resultados dos experimentos com dados reais.

		Voting	**AdaBoost**	**Bagging**	**Stacking**	**SVM**	**KNN**	**Decision Tree**
Falsos negativos	Média	4	2.4	3.12	2.46	4	4	2
	Variância	0	1.68	0.9456	1.0084	0	0	0
	Desvio padrão	0	1.2961	0.9724	1.0042	0	0	0
Falsos positivos	Média	0	0	0	1.28	0	0	0
	Variância	0	0	0	8.6016	0	0	0
	Desvio padrão	0	0	0	2.9328	0	0	0
Verdadeiros negativos	Média	70	70	70	68.72	70	70	70
	Variância	0	0	0	8.6016	0	0	0
	Desvio padrão	0	0	0	2.9328	0	0	0
Verdadeiros positivos	Média	1	2.6	1.88	2.54	1	1	3
	Variância	0	1.68	0.9456	1.0084	0	0	0
	Desvio padrão	0	1.2961	0.9724	1.0042	0	0	0
Recall	Média	0.20	0.52	0.376	0.508	0.2	0.2	0.6
	Variância	$6.93 * 10^{-33}$	0.0672	0.0378	0.0403	$6.93 * 10^{-33}$	$6.93 * 10^{-33}$	$3.0815 * 10^{-31}$
	Desvio padrão	$8.33 * 10^{-17}$	0.2592	0.1945	0.2008	$8.33 * 10^{-17}$	$8.33 * 10^{-17}$	$5.55 * 10^{-16}$
Precisão	Média	1	1	1	0.8836	1	1	1
	Variância	0	0	0	0.0711	0	0	0
	Desvio padrão	0	0	0	0.2667	0	0	0
Acurácia	Média	0.9467	0.968	0.9584	0.9501	0.9467	0.9467	0.9733
	Variância	$4.43 * 10^{-31}$	0.0003	0.0002	0.0015	$4.43 * 10^{-31}$	$4.43 * 10^{-31}$	$6.04 * 10^{-31}$
	Desvio padrão	$6.66 * 10^{-16}$	0.0173	0.0123	0.0387	$6.66 * 10^{-16}$	$6.66 * 10^{-16}$	$7.77 * 10^{-16}$
Taxa de erro	Média	0.0533	0.032	0.0416	0.0499	0.0533	0.0533	0.0267
	Variância	$3.9 * 10^{-33}$	0.0003	0.0002	0.0015	$3.9 * 10^{-33}$	$3.9 * 10^{-33}$	$9.75 * 10^{-34}$
	Desvio padrão	$6.24 * 10^{17}$	0.0173	0.013	0.0387	$6.24 * 10^{17}$	$6.24 * 10^{17}$	$3.12 * 10^{-17}$

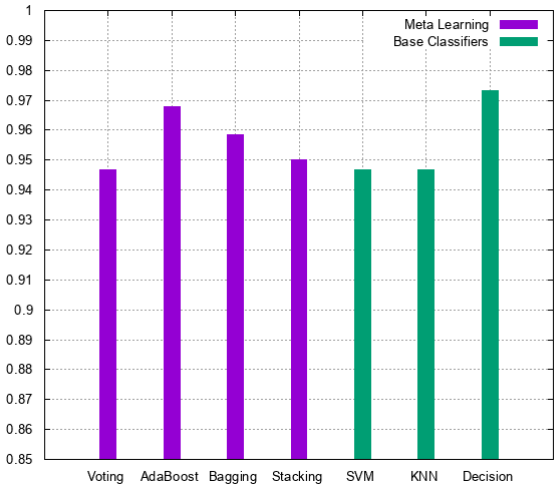
Observando os resultados, vemos que os classificadores individuais, quando aplicados a este conjunto de dados, são pouco relacionados. Isso fica evidente ao analisar os resultados de *Decision Tree*, que se destaca com a maior acurácia obtida (0.9733). Com classificadores base pouco relacionados, conforme é dito na literatura, os *ensemble learners* têm maiores chances de melhorar o desempenho desses classificadores individuais.

De fato, analisando as técnicas de *meta-learning* (*Voting*, *AdaBoost*, *Bagging* e *Stacking*), vemos que todas elas conseguiram superar o desempenho dos classificadores individuais - a acurácia teve um aumento de até 2,25% (no caso de *AdaBoost*, com uma acurácia de 96,8%).

O desempenho da técnica de *Voting* é idêntico ao desempenho dos classificadores individuais SVM e KNN pois estamos usando votação majoritária. Neste caso, ainda que *Decision Tree* forneça a classificação correta, sua predição é minoria, o que prejudica o resultado final de *Voting*.

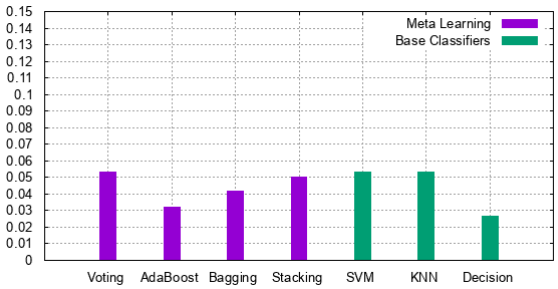
Além disso, vemos que o desempenho de *Stacking* não é tão bom quanto as técnicas *AdaBoost* e *Bagging*. Isso pode ser explicado pelo fato de *Stacking* ser a única entre essas técnicas que leva em consideração as predições dos classificadores SVM e KNN,

Figura 4.6: Gráfico que ilustra a acurácia obtida por cada método nos experimentos com dados reais.



Fonte: A autora

Figura 4.7: Gráfico que ilustra a taxa de erro obtida por cada método nos experimentos com dados reais.



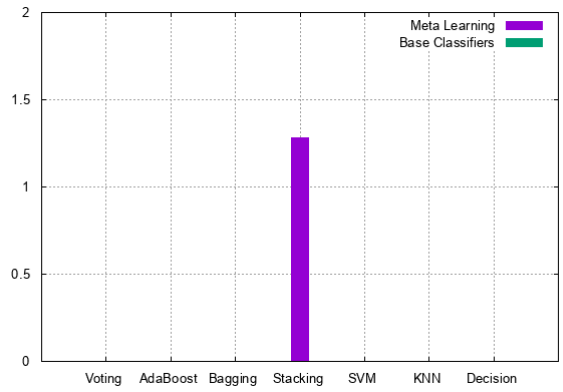
Fonte: A autora

que, conforme é mostrado, não são tão corretas.

Ainda, a diferença de resultados entre as técnicas *AdaBoost* e *Bagging* deve-se não apenas à diferença de aplicação dos métodos, mas também devido ao fator aleatório que é introduzido nestes métodos. Isto é, as instâncias sorteadas para os subconjuntos são randômicas e, portanto, têm grande relação com os resultados obtidos.

Finalmente, considerando que os principais objetivos da classificação são aumentar a acurácia e reduzir o número de falsos positivos identificados, vemos que as técnicas de *meta-learning AdaBoost*, *Bagging* e *Stacking* são as mais indicadas para este problema. Em especial, a técnica *AdaBoost* se destaca com o melhor desempenho de classificação de tráfego.

Figura 4.8: Gráfico que ilustra a quantidade de falsos positivos obtida por cada método nos experimentos com dados reais.



Fonte: A autora

4.2.3 Discussão

A proposta inicial deste trabalho era fazer uma análise de diferentes técnicas de *meta-learning*, em comparação com os classificadores individuais, utilizando dados reais. Para isso, tentamos utilizar dados coletados de tráfego de rede de uma empresa com grande volume de dados. No entanto, não foi possível utilizar esses dados devido a restrições das ferramentas disponíveis para coleta.

Para resolver este problema, buscamos dados reais de tráfego de rede disponíveis na Internet. Sites como NETRESEC [12] e Simple Web [13] contêm uma quantidade significativa de recursos neste contexto. No entanto, a maioria do conteúdo é focado em tráfego que representa atividade de um usuário comum, e não em algo que reflita uma atividade anômala. Dessa forma, conseguimos obter uma quantidade significativamente maior de fluxos considerados normais do que fluxos anômalos.

O desbalanceamento de um conjunto de dados, quando aplicado a um sistema de aprendizagem de máquina, é um problema que pode adicionar *bias* aos resultados. Isto é, como garantir que o sistema aprendeu com as informações fornecidas e não apenas está representando um sistema desbalanceado? Com isso em mente, o conjunto de dados reais inicialmente obtido foi reduzido a cerca de 10% do tamanho original. Este tamanho foi definido após uma série de experimentos, buscando bons resultados.

Por fim, optou-se por fazer uma análise comparativa entre os experimentos com dados artificias e dados reais. No entanto, espera-se que a análise preliminar apresentada

[12] http://www.netresec.com/
[13] https://www.simpleweb.org/

nesse capítulo possa servir de base para experimentos futuros, condicionados à obtenção de conjuntos de dados mais representativos.

5 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais trabalhos relacionados encontrados na literatura. Trabalhos como (DIDACI; GIACINTO; ROLI, 2002) e (ZAINAL; MAAROF; SHAMSUDDIN, 2009) utilizam técnicas de *ensemble learning* para detectar anomalias no contexto de detecção de intrusões (*Intrusion Detection*). Mais especificamente, na área de classificação de tráfego de rede, publicações como (HE et al., 2009), (WANG; GUAN; QIN, 2017) e (REDDY; HOTA, 2013) exploram o uso de *meta-learning*. Aspectos relevantes destes trabalhos, bem como as diferenças com o trabalho aqui proposto são descritos a seguir.

O trabalho apresentado em (DIDACI; GIACINTO; ROLI, 2002) propõe um sistema de deteccção de instrusões (IDS) baseado em *ensemble learning*. Cada classificador base é treinado com um conjunto distinto de *features*, que representam os padrões do conjunto de dados. Em seguida, os resultados de cada classificador base são combinados. Essa abordagem é motivada pela observação de que especialistas da área combinam características de ataques a partir de diferentes conjuntos de *features* para gerar novos ataques. Três regras de combinação diferentes foram utilizadas: votação majoritária, média e a chamada de função de crença (*belief function*) - estimativas probabilísticas com base em padrões observados. Os experimentos foram executados com um conjunto de dados criado por DARPA, em um *framework* de 1998 chamado *Intrusion Detection Evaluation Program*. Os resultados mostraram que *ensemble learners* são capazes de reduzir a taxa de erro geral, mas, com isso, também diminui a capacidade de generalização do sistema.

Também no ramo de IDS, o trabalho proposto em (ZAINAL; MAAROF; SHAMSUDDIN, 2009) tem como objetivo aumentar a acurácia e reduzir o número de falsos positivos de IDS. Para tanto, dois meios foram adotados: primeiro, selecionar um conjunto relevante de features, que representam os padrões do tráfego; segundo, combinar classificadores com diferentes paradigmas para gerar um ensemble learner. Neste sistema, os classificadores base utilizados foram Linear Genetic Programming (LGP) - algoritmo genético, Adaptive Neural Fuzzy Inference System (ANFIS) e Random Forest (RF). Os experimentos foram realizados com a base de dados KDD Cup 1999, extraída do programa de avaliação de detecção de intrusões DARPA (1998). Os resultados mostraram que é possível melhorar a acurácia de detecção de intrusões, em especial, quando se utiliza pesos para diferentes classificadores, que foi uma abordagem também utilizada nesse trabalho.

Já na área de classificação de tráfego, com maior relevância para o trabalho aqui descrito, (HE et al., 2009) apresenta um novo modelo de aprendizagem de máquina que combina *meta-learning* com técnicas de co-treinamento (técnica semi-supervisionada, que utiliza vários subconjuntos de treinamento, alguns classificados e outros não). Este trabalho compara abordagens anteriores que utilizam classificadores individuais com a nova proposta, o que ajuda a superar três principais deficiências: taxa de precisão de fluxo limitada, baixa adaptabilidade das técnicas de aprendizagem de máquina e a necessidade de um conjunto de treinamento classificado. O conjunto de dados, assim como no trabalho aqui proposto, é representado por um conjunto de features, extraídas de registros de fluxos. Os experimentos comprovam a eficácia da técnica.

Ainda, o trabalho de (WANG; GUAN; QIN, 2017) propõe uma abordagem de classificação baseada em características de subfluxos, utilizando *meta-learning*. Foi desenvolvido um método de truncamento de fluxos para processamento em tempo real, e um sistema de aprendizagem de máquina de agregação, baseado na precisão de cada classificador para diferentes aplicações. Os experimentos foram executados com dados reais - fluxos de rede da Universidade Xian Jiaotong, que verificam a eficácia dos métodos propostos. O resultado final mostra que o método proposto performa, em média, 8% melhor do que os classificadores base utilizados, e cerca de 3% melhor do que o melhor dos classificadores base utilizados.

Também semelhante ao trabalho aqui proposto, (REDDY; HOTA, 2013) introduz um modelo de *meta-learning* que combina variados modelos para melhorar a acurácia de classificadores individuais ou técnicas semi-supervisionadas de aprendizagem. Da mesma forma, utiliza características estatísticas extraídas de fluxos TCP e UDP para representar o conjunto de dados. Além disso, aplica-se um processo de feature selection, para reduzir e determinar quais são as melhores características a se utilizar. As técnicas de *meta-learning* utilizadas são *Stacking* e *Voting*, com os classificadores base *Naïve Bayes*, *Bayesian Network* e *Decision Tree*. Por fim, os experimentos mostram melhoria da acurácia de classificação para 99.9%. Além disso, os resultados indicam que a técnica de *Stacking* tem desempenho melhor do que a técnica de *Voting* ao classificar esses fluxos.

Diferentemente dos trabalhos acima, a proposta apresentada aqui é estudar o impacto do uso de *ensemble learners* em diferentes métricas - acurácia, precisão, *recall*, falsos positivos e negativos, verdadeiros positivos e negativos e, também, na taxa de erro. Para tanto, outra diferença dos trabalhos mencionados, o desempenho dos classificadores base quando utilizados individualmente também foi medido. Além disso, os experimentos

foram realizados com dois conjuntos de dados distintos: um conjunto de dados artificiais e um conjunto de dados reais. Os resultados, como mostram as pesquisas relacionadas, indicam dificuldade de generalização por parte dos *ensemble learners*, portanto, a acurácia dos modelos diminuiu e o número de falsos positivos, em geral, aumentou.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

A classificação de tráfego de rede e detecção de anomalias neste contexto são tarefas difíceis devido a vários motivos, incluindo a necessidade de obter uma visão precisa e abrangente da rede, a capacidade de detectar a ocorrência de novos tipos de ataque e a necessidade de lidar com erros de classificação (SILVA et al., 2016).

Neste trabalho foi feita uma análise comparativa entre técnicas de *meta-learning* e classificadores individuais para classificação de tráfego de rede. As técnicas utilizadas foram as mais comumente encontradas na literatura e disponibilizadas pela biblioteca *scikit-learn*, para a linguagem Python. Além disso, uma técnica adicional foi implementada (*Stacking*), devido aos bons resultados deste método apresentados na literatura.

As técnicas foram utilizadas em experimentos realizados com dados artifiais, gerados pelo trabalho desenvolvido em (SILVA et al., 2016) e com dados reais, disponibilizados publicamente pelo trabalho (SANTANNA et al., 2015) e pela 4SICS. Divididos em dois grupos (dados artificiais e dados reais), os experimentos foram executados 50 vezes, para que fosse possível avaliar a variação das predições.

Com base nos experimentos com dados artificiais, as técnicas de *meta-learning* apresentam melhores resultados apenas na identificação correta de fluxos considerados normais - identificação de verdadeiros negativos e consequente redução de falsos positivos. Em contrapartida, os classificadores individuais (e, por consequência, também a técnica *Voting*) apresentam maior acurácia de classificação das instâncias. A técnica *AdaBoost*, em especial, apresentou os mesmos resultados dos classificadores individuais, o que pode sugerir a ocorrência de *overfitting*.

Por outro lado, os experimentos com dados reais mostram que as técnicas de *meta-learning* conseguem identificar com maior sucesso as instâncias anômalas - verdadeiros positivos - exceto quando comparadas ao classificador *Decision Tree*. O número de falsos positivos, grande desafio do trabalho, se manteve o mesmo (0), exceto quando aplicada a técnica *Stacking*. Além disso, também apresentam maior acurácia quando comparadas com os classificadores individuais e a técnica *Voting*.

Em relação às técnicas de *meta-learning* utilizadas, destaca-se o desempenho das técnicas *AdaBoost* e *Stacking*, quando aplicadas nos experimentos com dados reais. Essas abordagens tiveram maior número de verdadeiros positivos encontrados. No entanto, no caso de *Stacking*, a redução na acurácia da classificação deve-se ao número menor de verdadeiros negativos e um número maior de falsos positivos. Isto é, a técnica classificou

um maior número de instâncias normais como anomalias.

Os resultados dos experimentos são justificados, primeiramente, pelos conjuntos de dados utilizados. Ao longo do trabalho discutiu-se a dificuldade de encontrar conjuntos de dados que reflitam o comportamento de fluxos reais e fluxos anômalos. Ademais, conforme discutido no Capítulo 3, a combinação de classificadores também pode provocar aumento na taxa de erro de predições, o que resultou, no estudo aqui apresentado, na redução de acurácia de algumas das técnicas de *meta-learning*, quando comparadas aos classificadores individuais, em experimentos com dados artificiais.

Por fim, destacam-se as seguintes contribuições deste trabalho:

- uma revisão de literatura das principais técnicas de *meta-learning*;
- o desenvolvimento de um sistema (e a correspondente arquitetura) que classifica fluxos de rede utilizando diferentes técnicas de combinação de classificadores
- um estudo comparativo entre a aplicação de técnicas de *meta-learning* e classificadores individuais, em relação à acurácia de classificação obtida.

Trabalhos futuros incluem a obtenção de conjuntos de dados mais representativos para realizar novos experimentos. Destaca-se também o possível estudo aprofundado dos melhores classificadores individuais a serem utilizados neste contexto, bem como os parâmetros associados. Além disso, consideramos a utilização de técnicas de *Deep Reinforcement Learning* (SUTTON; BARTO, 1998) para classificação de tráfego de rede para um novo estudo, por conta dos bons resultados que a literatura tem mostrado (LILLICRAP et al., 2015) (SCHMIDHUBER, 2015).

REFERÊNCIAS

AHMED, T.; ORESHKIN, B.; COATES, M. Machine learning approaches to network anomaly detection. In: **Proceedings of the 2Nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques**. Berkeley, CA, USA: USENIX Association, 2007. (SYSML'07), p. 7:1–7:6. Available from Internet: <http://dl.acm.org/citation.cfm?id=1361442.1361449>.

BOSWELL, D. Introduction to support vector machines. 2002.

BRAZDIL, P. et al. **Metalearning: Applications to Data Mining**. 1. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 3540732624, 9783540732624.

BREIMAN, L. Bagging predictors. **Machine Learning**, v. 24, n. 2, p. 123–140, Aug 1996. ISSN 1573-0565. Available from Internet: <https://doi.org/10.1023/A:1018054314350>.

BREIMAN, L. Bagging predictors. **Mach. Learn.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 24, n. 2, p. 123–140, aug. 1996. ISSN 0885-6125. Available from Internet: <http://dx.doi.org/10.1023/A:1018054314350>.

BULLINARIA, J. A. **Self Organizing Maps: Fundamentals**. [S.l.]: The University of Birmingham, 2004.

CHAN, P. K.; STOLFO, S. J. Experiments on multistrategy learning by meta-learning. In: **Proceedings of the Second International Conference on Information and Knowledge Management**. New York, NY, USA: ACM, 1993. (CIKM '93), p. 314–323. ISBN 0-89791-626-3. Available from Internet: <http://doi.acm.org/10.1145/170088.170160>.

CHAN, P. K.; STOLFO, S. J. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In: **KDD**. [S.l.: s.n.], 1998. v. 1998, p. 164–168.

DIDACI, L.; GIACINTO, G.; ROLI, F. Ensemble learning for intrusion detection in computer networks. **ACM JOURNAL NAME, VOL. V, NO. N**, p. 533620, 2002.

DIETTERICH, T. G. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. **Machine Learning**, v. 40, n. 2, p. 139–157, Aug 2000. ISSN 1573-0565. Available from Internet: <https://doi.org/10.1023/A:1007607513941>.

DŽEROSKI, S.; ŽENKO, B. Is combining classifiers with stacking better than selecting the best one? **Machine Learning**, v. 54, n. 3, p. 255–273, Mar 2004. ISSN 1573-0565. Available from Internet: <https://doi.org/10.1023/B:MACH.0000015881.36452.6e>.

FREUND, Y.; SCHAPIRE, R. E. **A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting**. 1996.

GARCÍA-TEODORO, P. et al. Anomaly-based network intrusion detection: Techniques, systems and challenges. **Comput. Secur.**, Elsevier Advanced Technology Publications, Oxford, UK, UK, v. 28, n. 1-2, p. 18–28, feb. 2009. ISSN 0167-4048. Available from Internet: <http://dx.doi.org/10.1016/j.cose.2008.08.003>.

HAMDI, M.; GRIRA, N.; BOUDRIGA, N. Detecting distributed computer network attacks: A multi-dimensional wavelet approach. In: **2005 12th IEEE International Conference on Electronics, Circuits and Systems**. [S.l.: s.n.], 2005. p. 1–5.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. Overview of supervised learning. In: ____. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. New York, NY: Springer New York, 2009. p. 9–41. ISBN 978-0-387-84858-7. Available from Internet: <http://dx.doi.org/10.1007/978-0-387-84858-7_2>.

HE, H. et al. Network traffic classification based on ensemble learning and co-training. **Science in China Series F: Information Sciences**, v. 52, n. 2, p. 338–346, Feb 2009. ISSN 1862-2836. Available from Internet: <https://doi.org/10.1007/s11432-009-0050-8>.

LAKHINA, A.; CROVELLA, M.; DIOT, C. Mining anomalies using traffic feature distributions. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 35, n. 4, p. 217–228, aug. 2005. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1090191.1080118>.

LAZAREVIC, A. et al. A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In: **Proceedings of the Third SIAM International Conference on Data Mining**. [S.l.: s.n.], 2003.

LEMKE, C.; BUDKA, M.; GABRYS, B. Metalearning: a survey of trends and technologies. **Artificial Intelligence Review**, v. 44, n. 1, p. 117–130, Jun 2015. ISSN 1573-7462. Available from Internet: <https://doi.org/10.1007/s10462-013-9406-y>.

LILLICRAP, T. P. et al. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015.

MAGLOGIANNIS, I. **Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies**. IOS Press, 2007. (EBSCO ebook academic collection). ISBN 9781435625198. Available from Internet: <https://books.google.com.br/books?id=TXuWnQAACAAJ>.

MARNERIDES, A.; SCHAEFFER-FILHO, A.; MAUTHE, A. Traffic anomaly diagnosis in internet backbone networks: a survey. **Computer Networks**, Elsevier, v. 73, p. 224–243, 11 2014. ISSN 1389-1286.

MITCHELL, T. M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

MOORE, A. W.; ZUEV, D. Internet traffic classification using bayesian analysis techniques. In: **Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems**. New York, NY, USA: ACM, 2005. (SIGMETRICS '05), p. 50–60. ISBN 1-59593-022-1. Available from Internet: <http://doi.acm.org/10.1145/1064212.1064220>.

NGUYEN, T. T. T.; ARMITAGE, G. A survey of techniques for internet traffic classification using machine learning. **IEEE Communications Surveys Tutorials**, v. 10, n. 4, p. 56–76, Fourth 2008. ISSN 1553-877X.

PANDA, M.; ABRAHAM, A.; PATRA, M. R. A hybrid intelligent approach for network intrusion detection. **Procedia Engineering**, v. 30, p. 1 – 9, 2012. ISSN 1877-7058. International Conference on Communication Technology and System Design 2011. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1877705812008375>.

PANG, K. **Self-Organizing Maps**. 2003. <https://www.cs.hmc.edu/~kpang/nn/som.html>. Acessado em 27/07/2017.

PRODROMIDIS, A. L.; CHAN, P. K.; STOLFO, S. J. Meta-learning in distributed data mining systems: Issues and approaches. In: . [S.l.: s.n.], 2000.

QUINLAN, J. R. Generating production rules from decision trees. In: **ijcai**. [S.l.: s.n.], 1987. v. 87, p. 304–307.

QUINLAN, J. R. Improved use of continuous attributes in c4.5. **J. Artif. Int. Res.**, AI Access Foundation, USA, v. 4, n. 1, p. 77–90, mar. 1996. ISSN 1076-9757. Available from Internet: <http://dl.acm.org/citation.cfm?id=1622737.1622742>.

REDDY, J. M.; HOTA, C. P2p traffic classification using ensemble learning. In: **Proceedings of the 5th IBM Collaborative Academia Research Exchange Workshop**. New York, NY, USA: ACM, 2013. (I-CARE '13), p. 14:1–14:4. ISBN 978-1-4503-2320-8. Available from Internet: <http://doi.acm.org/10.1145/2528228.2528243>.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 2. ed. [S.l.]: Pearson Education, 2003. ISBN 0137903952.

SANTANNA, J. J. et al. Booters x2014; an analysis of ddos-as-a-service attacks. In: **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2015. p. 243–251. ISSN 1573-0077.

SCHAPIRE, R. E. The strength of weak learnability. **Mach. Learn.**, Kluwer Academic Publishers, Hingham, MA, USA, v. 5, n. 2, p. 197–227, jul. 1990. ISSN 0885-6125. Available from Internet: <https://doi.org/10.1023/A:1022648800760>.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, v. 61, n. Supplement C, p. 85 – 117, 2015. ISSN 0893-6080. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0893608014002135>.

SEEWALD, A. K. Meta-Learning for Stacked Classification. In: **Proceedings of the Second International Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning (IDDM-2002)**. [S.l.: s.n.], 2002.

SILVA, A. S. d. et al. Identification and selection of flow features for accurate traffic classification in sdn. In: **Proceedings of the 2015 IEEE 14th International Symposium on Network Computing and Applications (NCA)**. Washington, DC, USA: IEEE Computer Society, 2015. (NCA '15), p. 134–141. ISBN 978-1-5090-1849-9. Available from Internet: <http://dx.doi.org/10.1109/NCA.2015.12>.

SILVA, A. S. da et al. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 27–35.

SUTHAHARAN, S. Big data classification: Problems and challenges in network intrusion prediction with machine learning. **SIGMETRICS Perform. Eval. Rev.**, ACM, New York, NY, USA, v. 41, n. 4, p. 70–73, abr. 2014. ISSN 0163-5999. Available from Internet: <http://doi.acm.org/10.1145/2627534.2627557>.

SUTTON, R. S.; BARTO, A. G. **Introduction to Reinforcement Learning**. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

TING, K. M.; WITTEN, I. H. Stacked generalization: when does it work. In: **in Procs. International Joint Conference on Artificial Intelligence**. [S.l.]: Morgan Kaufmann, 1997. p. 866–871.

TODOROVSKI, L.; DŽEROSKI, S. Combining classifiers with meta decision trees. **Machine Learning**, v. 50, n. 3, p. 223–249, Mar 2003. ISSN 1573-0565. Available from Internet: <https://doi.org/10.1023/A:1021709817809>.

VILALTA, R.; GIRAUD-CARRIER, C.; BRAZDIL, P. Meta-learning - concepts and techniques. In: ____. **Data Mining and Knowledge Discovery Handbook**. Boston, MA: Springer US, 2010. p. 717–731. ISBN 978-0-387-09823-4. Available from Internet: <https://doi.org/10.1007/978-0-387-09823-4_36>.

WANG, C.; GUAN, X.; QIN, T. A traffic classification approach based on characteristics of subflows and ensemble learning. In: **2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2017. p. 588–591.

WANG, G. et al. A comparative assessment of ensemble learning for credit scoring. **Expert Systems with Applications**, v. 38, n. 1, p. 223 – 230, 2011. ISSN 0957-4174. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S095741741000552X>.

WOLPERT, D. H. Stacked generalization. **Neural Networks**, v. 5, p. 241–259, 1992.

YINGCHAREONTHAWORNCHAI, S. et al. Precision, recall, and sensitivity of monitoring partially synchronous distributed systems. **CoRR**, abs/1607.03369, 2016. Available from Internet: <http://arxiv.org/abs/1607.03369>.

ZAINAL, A.; MAAROF, M.; SHAMSUDDIN, S. M. Ensemble classifiers for network intrusion detection system. v. 4, p. 217–225, 07 2009.