Relatório Final - Ciência de dados para Segurança

Marcos Vinicius Pontarolo[1], **Dante da Silva Aléo**[1]

[1]Departamento de Informática - Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 - 81.531-980 - Curitiba - PR - Brasil

{mvp19,dsa17}@inf.ufpr.br

Abstract. *This paper describes the entire process for the final project of the Data Science for Security discipline, introducing and specifying our entire dataset, as well as the features of the dataset, the original dataset used was extract from artificial DDoS attack dataset idealized with several different types of DDoS attacks , using real world data to be as close as possible to a real attack. The machine learning models used were KNN, Random Forest and MLP, which will be detailed in this paper.*

Resumo. *Este relatório descreve todo o processo para o projeto da disciplina de Ciência da dados para Segurança, introduzindo e especificando todo o nosso dataset, bem como as caracterísitcas do dataset, neste caso sendo um dataset de ataque DDoS artificial idealizado com diversos tipos distintos de ataque DDoS, utilizando dados do mundo real para ser o mais próximo possível de um ataque real. Os modelos de aprendizado de máquina utilizados foram KNN, Random Forest e MLP que serão discutidos com maiores detalhes*

1. Introdução

Este relatório apresenta o projeto final da disciplina de Ciência de Dados para Segurança. Resumidamente, o dataset foi obtido através do *DDoS Evaluation Dataset*[D]. Esse *Dataset* contém os mais diversos tipos de ataque DDoS, atualizados até 2019. A ideia para o projeto consiste em classificar um tráfego de rede entre normal ou ataque DDoS, criando assim uma detecção de um ataque DDoS. Espera-se que, ao classificar um ataque, o servidor consiga habilitar regras específicas para barrar o tráfego de rede mal-intencionado.

2. Dataset

O *Dataset* foi criado após a etapa de exploração do projeto. O pacote completo inclui diversos arquivos CSV para cada tipo de ataque, já com as características da análise feita pelo *CICFlowMeter*[C] em cima de arquivos PCAP da simulação do tráfego de rede e de diferentes tipos de ataques DDoS. A ferramenta chegou a ser utilizada para a exploração deste projeto, porém, os arquivos CSV já eram bem relevantes para a exploração do *dataset* e a elaboração dos modelos.

Para lidar com a grande densidade de dados que tinhamos em mão, criamos um script para incluir apenas uma pequena parte de cada arquivo csv em um único arquivo, chamando de **Dataset Completo**, que é de fato, o *dataset* que utilizamos na continuidade do projeto. Esse *Dataset* contém 0.05% do tráfego de rede de ataques DDoS de cada um dos arquivos CSV de tipos variados de ataques, e contém também, 50% de todo o trafégo de rede normal dos mesmos arquivos. A diferença notável das porcentagens se deve à

distribuição das classes dentro de cada arquivo CSV, tendo os ataques uma densidade bem maior do que o tráfego normal da rede. Iremos verificar isso nas próximas seções, analisando nosso **Dataset completo**.

Ainda nas tarefas exploratórias, foram definidos rótulos para todo o *Dataset*, classificando em Ataque DDoS ou Tráfego Normal. Tornando-se assim um problema de classificação binária. Sendo 0 para tráfego normal ou 1 para ataque DDoS. Essa rotulação foi incluída antes do processamento dos modelos. Também foram tratadas as inconsistências dos dados, tais como **INF, -INF, NAN**, etc.

O conjunto original dos dados continha 87 colunas, com as mais diversas características do tráfego de rede. Desse conjunto, retiramos algumas classes que nao seriam relevantes para nosso problema, como por exemplo, *Destionation IP* e outras características que identificavam os alvos e atacantes, pois nós não estamos interessados nessas informaçõs já que queremos classificar um ataque de alvo e atacante genéricos. O conjunto final conta com 82 colunas para realização da classificação.

A seguir, o **Dataset completo**, que continha 13623 registros de tráfego de rede normal e 137006 registros de tráfego de rede DDoS, foi separado em porções de 20% e 80% para teste e treino, respectivamente.

2.1. Porção para teste

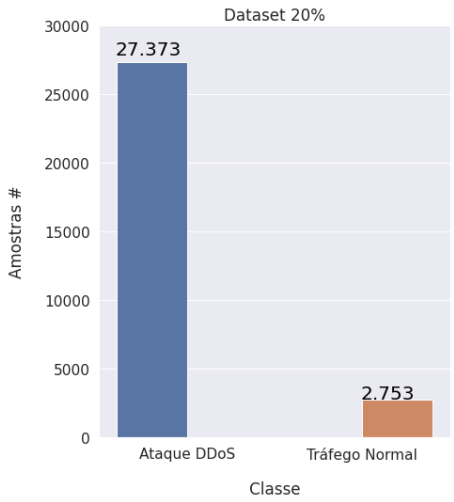
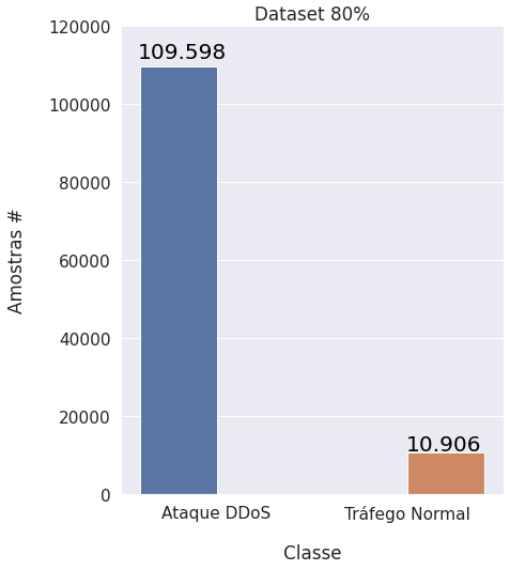


Figura 1. Distribuição das amostras por classe no dataset de teste

A distribuição do *dataset* utilizado como teste conta com 23.373 registros de tráfego de rede de Ataque DDoS e 2.753 tráfego de rede normal.

2.2. Porção para treino



A distribuição do *dataset* utilizado como treino conta com 109.598 registros de tráfego de rede de Ataque DDoS e 10.906 tráfego de rede normal.

3. Modelos

Após o tratamento do nosso *dataset* e a partição para testes e treino, elaboramos nossos Modelos para classificar, treinar e validar nosso *dataset*, com o intuito de criar predições mais precisas possíveis para a detecção ou não de um ataque DDoS, ajustando os parâmetros de cada modelo, alinhado com uma análise de todas as métricas obtidas, para melhor aperfeiçoamento do treinamento.

Os modelos utilizados foram: *KNN, Random Forest* e *MLP - Multi-layer Perceptron*, a seguir iremos mostrar e discutir sobre cada um deles.

Como citado anteriormente, nosso *Dataset* é desbalanceado entre as classes que criamos, por isso, algumas medidas foram adotadas. A utilização do F1Score como uma das métricas mais importantes na classificação do nosso modelo, pois o F1Score é bem mais preciso com uma densidade descrepante das classes. Outra medida é a utilização das curvas de *Precision Recall*, que é também a mais aconselhada para dados desbalanceados, ao invés da curva ROC.

Para cada algoritmo, utilizamos configurações variadas, nos quais iremos citar na seção de cada modelo. Contudo, utilizamos validação cruzada ou *K-fold Cross Validation* com 5 pastas (*5-fold*). Para a validação dos resultados obtidos na aplicação de cada algoritmo na porção de treino e teste, criando assim uma validação um pouco mais realística do que poderiamos enfrentar na vida real.

Os resultados foram bastante promissores em nossos treinamentos e testes realizados, isso pode ser dar ao fato de que a nossa classificação binária é um pouco genérica, somente estamos classificando se é ou não um ataque DDoS, tendo bastante dados do tráfego de rede que contém um ataque, com isso cada algoritmo consegue criar e associar facilmente as características de um ataque, criando probabilidades com uma boa acurácia,

tendo classificado um ataque DDoS no tráfego de rede, a classificação de um tráfego normal é natural, pela negação do primeiro. Uma extensão do projeto seria rotular os tipos de ataques DDoS e incluir uma classificação para cada tipo de ataque, assim um servidor poderia distinguir ataques e criar defesas ainda melhores para certo tipo de ataque. Porém, a ideia é que o modelo tenho uma acurácia menor para a classificação dos tipos de ataques, tendo em vista que não iremos mais partir de um problema de classificação binária. Também teremos que readequar o nosso **Dataset completo**, tentando equilibrar as informações de cada tipo de ataque. Uma abordagem por agrupamento do tráfego de redes desses ataques pode ser uma maneira eficaz de trabalhar os novos dados.

Uma das tarefas exploratórias interessantes que fizemos, foi a classificação e listagem das caracteríticas pelo algoritmo de *Decision Tree*, de acordo com o seu nível de *Score* na validação cruzada, o gráfico da figura 2 mostra o resultado que obtivemos.

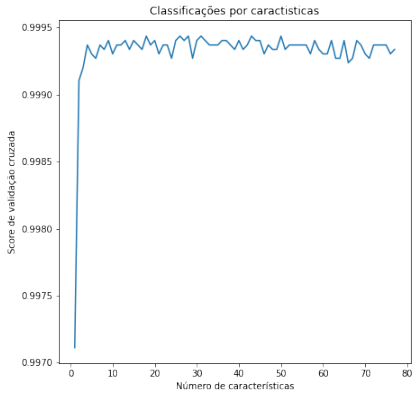


Figura 2. Nível de score por número de características selecionadas

Ou seja, selecionando cerca de 20 características podemos ter um bom resultado para nosso classificador, isso parte de uma premissa e teste inicial que fizemos antes da construção e criação dos modelos que serão listados abaixo.

Para encontrar as melhores configurações de cada algoritmo, realizamos alguns testes individuais, variando parâmetros básicos, e por fim, incluímos um teste com o *Grid Search* derivando as nossas escolhas com os testes individuais realizados. Com isso, esperávamos criar um certo filtro para o *Grid Search*, pois o algoritmo executa diversas combinações dos parâmetros para obter um melhor resultado, sendo p a soma da quantidade de todos os valores dos parâmetros, então o algoritmo executa $p!$ análises. Os resultados apresentados para cada algoritmo inclui então as melhores configurações provenientes dos testes realizados.

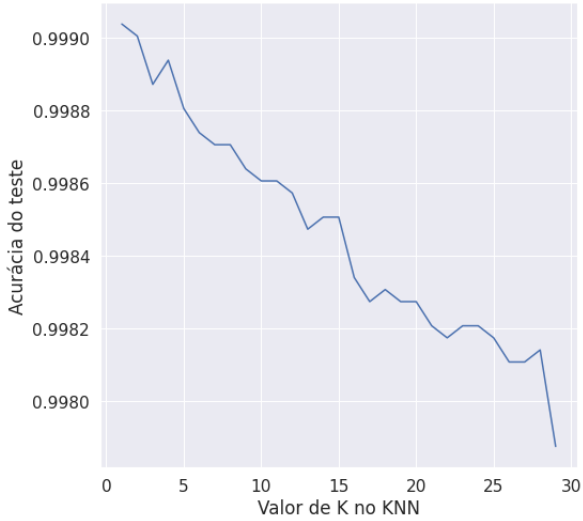
3.1. KNN

O primeiro modelo a ser criado, foi utilizando o algoritmo de KNN(*K-Nearest Neighbors*), tendo em vista principalmente a sua simplicidade de aplicação, também com a ideia dos exemplares do espaço de características estarem próximos de outros, tendo assim a mesma classe. A hipótese é que ataques DDoS tem um fluxo grande em um determinado tempo, aumentando assim o tráfego de rede e caindo na mesma rotulação de ataque, fi-

cando cada exemplar do ataque próximo de outro. Então a classificação do KNN pode ser uma boa opção para esse tipo.

3.2. Configurações

Os testes foram realizados utilizando K = 3 e os parâmetros padrão do algoritmo de KNN. Porém, após todos os testes utilizando essa configuração, voltamos para analisar o desempenho do KNN utilizando diferentes tipos de configurações, buscando obter o melhor desempenho do classificador. Primeiramente, fizemos testes com o K variando de 1 até 30, para obter o melhor K possível utilizando os parâmetros padrão, plotamos o desempenho de scores em um gráfico para realizar uma análise e escolher o melhor K.



Com esse resultado, concluímos que, quanto maior o K (número de vizinhos) para nosso problema, pior será a acurácia do nosso modelo de KNN. Nosso K utilizado anteriormente, sem embasamento em testes, foi um K que de fato é uma boa escolha para o modelo, embora K = 1 seria uma opção mais relevante.

Agora, olhando para as outras configurações, foi utilizada a técnica de Grid Search para buscar os melhores parâmetros para nosso algoritmo de KNN. Utilizando o resultado da acurácia do KNN variando o K, incluímos como imput no Grid Search os valores de 1 e 3 para K. Além disso, temos outros 3 parâmetros que podemos passar para o KNN: *Weights*, *algorithm* e *Metric*. O *algorithm* será calculado automaticamente baseado nos outros 2 parâmetros. Além disso, utilizamos *5 K fold cross validation* nos parâmetros do *Grid Search*.

Para o parâmetro *Weight* foram testados *Uniform* e *Distance*. No parâmetro *Metric*, utilizamos as métricas *minkowski, euclidean* e *manhattan*. Executando o *Grid Search*, chegando no resultado de que as melhores configurações poderiam ser:
metric = manhattan, k = 1, weights = uniform.

Com isso, incluímos essas métricas em nosso modelo KNN e realizamos os testes novamente. A seguir, segue um resumo dos resultados obtidos através do novo teste.

3.2.1. Valores obtidos

Aqui apresentaremos os valores obtidos após o treinamento do modelo e a sua execução de predição em cima da porção de teste do *dataset*.

Precisão: 0.999561

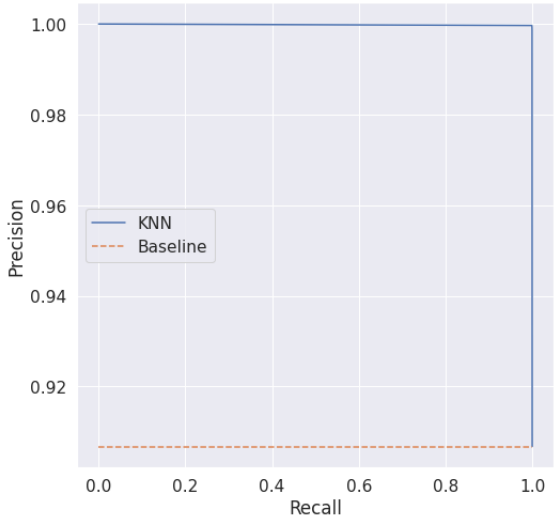
Recall: 0.999487

F1Score: 0.999524

3.2.2. Matriz de Confusão

Predito	0	1	All
Real			
0	2797	12	2809
1	14	27303	27317
All	2811	27315	30126

3.2.3. Gráfico P/R



3.2.4. Validação Cruzada

Precisão: 0.999438
Recall: 0.999540
F1Score: 0.999489

3.2.5. Conclusão do modelo

O modelo de KNN utilizando as configurações padrões já é bem preciso para a classificação do nosso problema binário. Quando realizamos *tuning* e criamos experi-

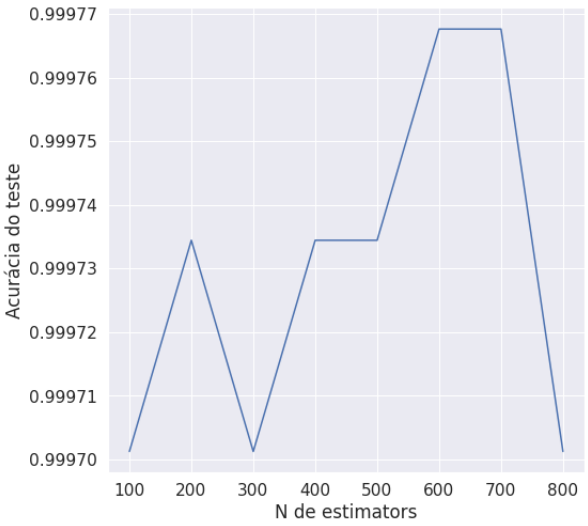
mentos em cima de novos parâmetros, até encontrarmos os possíveis melhores parâmetros para nossa classificação, temos uma pequena melhora no desempenho do modelo, caso a primeira classificação com os valores padrão tivesse sido uma classificação falha, com certeza teríamos ganhos bem melhores aplicando os novos parâmetros.

3.3. Random Forest

Random Forest é um método de aprendizado de máquina utilizado para problemas que envolvam classificação ou regressão. Ele se baseia em uma coleção de árvores de decisão. Outra grande qualidade das florestas aleatórias é a facilidade para se medir a importância relativa de cada característica (feature) para a predição. Pelo fato de nosso dataset ser fortemente beaseado em features, e estamos justamente realizando classificações, faz com que o random florest seja uma ótima escolha de algoritmo

3.3.1. Configurações

Os testes foram realizados utilizando todas as configurações padrões do algoritmo de *Random Forest*. Tendo em mente o mesmo procedimento do KNN 3.1, fizemos uma análise de desempenho testando diversas configurações para o algoritmo. Em uma primeira análise, realizamos um teste semelhante ao do KNN, utilizando o parâmetro *N_estimators*, fizemos uma pequena amostragem e variamos o parâmetro entre os valores de 100 até 800, subindo em 100 para cada teste.



Observando esse experimento, incluímos o N de 650 em nossos testes, pois para valores acima de 700 temos uma tendência de perder a acurácia dos nossos modelos.

Ademais, juntamente com os testes, relembramos das nossas tarefas exploratórias, principalmente da classificação de características utilizando *Decision Tree*, como visto na figura 2, uma boa quantidade de *features* seria um número próximo de 20, pois para valores diferentes a classificação não ficaria muito restrita a uma taxa de *score* estável.

Voltando ao nosso modelo, temos também um parâmetro denominado *Max Features*, no qual definimos um N máximo de características do nosso *Dataset*, tendo a ideia de que *Random Forest* é um modelo baseado em *Decision Tree*, podemos testar um n que

se aproxima dos testes realizados. Nesse caso, iremos testar com N = 20,28 e 50 para acompanhar os resultados.

Outros parâmetros definidos por nós foram *Min samples split y* e *Max Depth z*, variamos diversos valores para os testes dos resultados nesses campos, a varição final ficou em y = 2,3,5 e z = none, 15, 25.

Por fim, incluímos todos os parâmetros finais nos testes do *Grid Search* e tivemos as seguintes melhores configurações selecionadas:
Max Depth = 25, Max Features = 50, Min Samples Split = 3, N Estimators = 200

Embora tivessemos estimado o *Max Features* em cerca de 20, concluímos que o melhor valor seria 50, mas como mencionado anteriormente, a linha do gráfico determinando o N de características tende a ser estável conforme N cresce.

Com isso, incluímos os novos parâmetros em nosso modelo *Random Forest* e realizamos os testes novamente. A seguir, segue um resumo dos resultados obtidos através do novo teste.

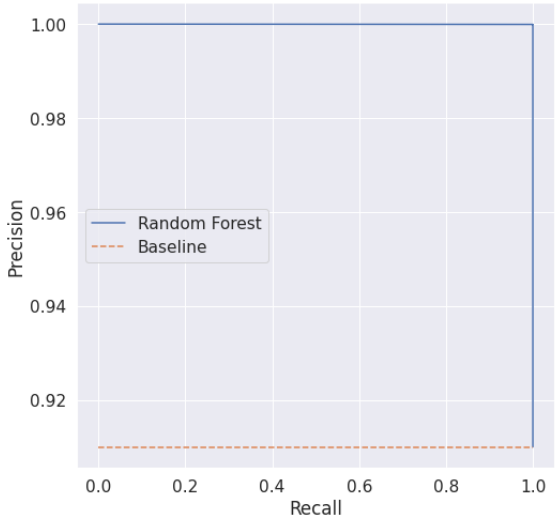
3.3.2. Valores obtidos

Aqui apresentaremos os valores obtidos após o treinamento do modelo e a sua execução de predição em cima da porção de teste do *dataset*.
Precisão: 0.999963
Recall: 0.999817
F1Score: 0.999890

3.3.3. Matriz de Confusão

Predito	0	1	All
Real			
0	2808	1	2809
1	5	27312	27317
All	2813	27313	30126

3.3.4. Gráfico P/R



3.3.5. Validação Cruzada

Aqui apresentaremos os valores obtidos através de k-fold cross validation, utilizando 5 pastas.

Precisão: 0.999934

Recall: 0.999796

F1Score: 0.999865

3.3.6. Conclusão do modelo

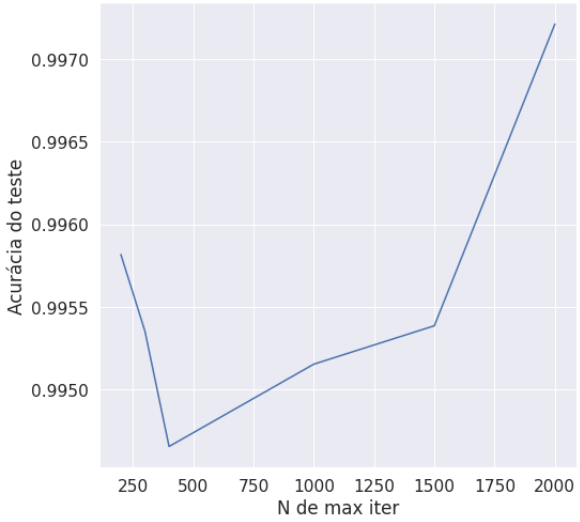
O modelo se Random Forest para nosso dataset possui seu melhor desempenho com N = 650, contudo houve uma ”surpresa”referente ao número de *features* utilizadas, visto que a *Decision Tree* nos indicou que um bom parâmetro a ser escolhido seria um número próximo de 20, porém, ao final percebemos que o melhor número é 50. O modelo *Random Forest* obteve resultados tão bons quanto o KNN, sendo também uma ótima escolha como algoritmo para nosso modelo.

3.4. MLP - Multi-layer Perceptron

O último modelo é uma rede neural com várias camadas de neurônios em alimentação direta e que utiliza o algoritimo de *ackpropagation*. Contudo, o tempo de treinamento de redes neurais utilizando *backpropagation* tende a ser muito lento. Algumas vezes são necessários milhares de ciclos para se chegar a níveis de erros aceitáveis, e que faz com que esse algoritmo tenha um desempenho ruim com grande quantidade de dados, que é justamente o caso de nosso dataset

3.4.1. Configurações

Novamente, os testes foram realizados utilizando todas as configurações padrões do MLP. Tentando novamente ajustar os parâmetros do nosso modelo. Realizamos alguns testes mudando os parâmetros padrões do MLP, um dos principais parâmetros, mesmo sendo genérico, que identificamos foi o *Max iter*. Realizamos então um teste modificando esse parâmetro e plotamos sua acurácia para melhor visualizarmos o resultado, e termos um ponto inicial para incluirmos no algoritmo de *Grid Search*.



Concluimos que, ao aumentar nossas iterações, conseguimos ter resultados com uma melhor acurácia dentro do nosso modelo. Porém, como esse parâmetro pode ser grande o bastante para não conseguirmos medir, ficamos satisfeitos com uma iteração de 2000 para realizarmos nossos testes.

Foram incluídos alguns parâmetros importantes para a análise de desempenho do nosso modelo, o primeiro deles é o número de camadas e neurônios em nossas camadas. *hidden layer sizes*, fizemos testes com variádos parâmetros analisando cada situação, um dos testes finais foram: (10,10,10), (50,50,50),(50,100,50), (100,), sendo o último o padrão do modelo, o tamanho dos vetores definem o tanto de camadas que temos, sendo 5 camadas, tendo como número de neurônios na i-ezima camada o i-ezimo elemento de cada vetor. Outro parâmetro que foi incluído, foi o *Acivation*, definindo a função de ativação para as camadas da rede neural, foram testados as funções tahn e relu nos testes finais. O parâmetro*Solver* também foi setado, utilizando sgd e adam para os testes. Ele verificará a resolução dos problemas de otimização da rede neural. O *Alpha* da nossa rede neural também foi determinado, dessa vez testando o parâmetro padrão 0.0001 e 0.05. Por último, especificamos também um *Learning Rate*, utilizando o **Constant** e *Adaptative* como parâmetros para teste.

Há diversos outros parâmetros que podemos utilizar e testar em nosso modelo de MLP, porém, como já estamos tendo uma precisão boa em nossos testes, não achamos relevante continuar testando N parâmetros, porém, para um problema mais complexo teríamos que realizar uma análise mais profunda. É interessante citar aqui que o artigo [h] nos ajudou com as configurações do algoritmo.

Utilizando os parâmetros exemplifcados acima, proveniente de outros testes, in-

cluímos em nosso *Grid Search*, para buscarmos o melhor resultado da combinação, e encontramos o seguinte:
activation = relu, alpha = 0.0001, hidden layer sizes = (100,), learning rate = constant, max iter = 2000, solver = adam

Portanto, após os testes de alguns parâmetros e ajustes para analisar o desempenho, incluímos todos os novos parâmetros otimizados em nosso modelo de MLP final. É interessante ressaltar aqui que os testes com o MLP demoraram diversas horas para serem concluídos, isso se deve dos testes com um n de iterações elevado que incluímos, além das combinações dos parâmetros. A seguir iremos resumir os testes utilizando os melhores parâmetros encontrados.

3.4.2. Valores obtidos

Aqui apresentaremos os valores obtidos após o treinamento do modelo e a sua execução de predição em cima da porção de teste do *dataset*.

Precisão: 0.992061

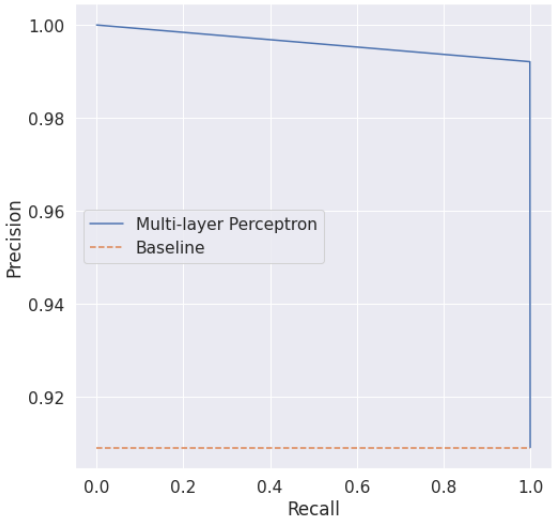
Recall: 0.999306

F1Score: 0.995671

3.4.3. Matriz de Confusão

Predito	0	1	All
Real			
0	2521	219	2740
1	19	27367	27386
All	2540	27586	30126

3.4.4. Gráfico P/R



3.4.5. Validação Cruzada

Aqui apresentaremos os valores obtidos através de k-fold cross validation *dataset*.

Precisão: 0.992133

Recall: 0.998256

F1Score: 0.995169

3.4.6. Conclusão do modelo

O modelo de MLP que utiliza redes neurais, foi um modelo interessante, principalmente para estudarmos os melhores parâmetros para o MLP. Tendo em vista que, comparado aos outros 2 modelos citados anteriormente, existem diversos parâmetros que podem ser ajustados e escolhidos para melhorar a classificação do MLP. Porém, para realizar os mais diversos testes, principalmente utilizando a técnica do *Grid Search*, demanda um bom tempo de processamento para a conclusão. Então a abordagem para a escolha dos melhores parâmetros tem que se dar por alguns testes individuais, gerando gráficos, igual realizado com o *Max iter*. Contudo, o modelo obteve um excelente desempenho, principalmente comparado com o MLP sem a alteração dos parâmetros, no qual chegamos a alcançar até 88% de f1score.

4. Apresentação

A seguir, temos um link para a apresentação do projeto final desenvolvido.

https://drive.google.com/file/d/17VpI2SMUM3U530BK32GBRZXHU43ljL9i/view?usp=sharing

5. Reprodutibilidade

Todo o *Dataset*, seus códigos de exploração, código final, modelos, testes realizados, explorações realizadas e todas as demais partes do projeto podem ser encontrada no link a seguir do GitHub. Também foram incluidos os slides da apresentação e demais arquivos relevantes. Porém, para o *dataset* completo, incluindo arquivos PCAP e todos os outros tipos de ataque, foi incluído somente um link para a página de *download* oficial, visto que os arquivos são muito pesados para o *upload* no Git.

Nos códigos de todas as etapas do projeto, há comentários relevantes para auxiliar o melhor entendimento dos passos realizados.

Caminhos:
Dataset/ - nosso **Dataset Completo**, criado a partir das amostras dos arquivos CSV; Arquivo csv, criado a partir dos arquivos PCAP; Um arquivo TXT com o link para os demais arquivos do *Dataset* extraído, além de um link para o *CicFlowMeter*[C] - programa utilizado para a extração dos arquivos brutos de rede.
exploratorio/ - Códigos, scripts e demais arquivos utilizados nas tarefas exploratórias.
projeto/ - Código final e limpo para a criação desse relatório final, contendo os modelos exemplificados e toda o tratamento realizado no *Dataset*; Os slides utilizados na apresentação, além de links para a apresentação e este próprio PDF.

Link para o GitHub:
https://github.com/marcospontarolo/CI1030

6. Conclusão

Por fim, concluímos que o algoritmo de *Random Forest* foi melhor modelo para a classificação e treinamento do nosso *Dataset* aliado com o nosso problema binário de classificação de um ataque DDoS. Isso deve bastante ao fato de que conseguimos otimizar bastante os parâmetros do modelo, pois a execução de cada teste foi relativamente rápida, outros modelos tiveram um tempo muito maior de processamento. Contudo, como mencionado anteriormente, o problema é genérico e simples, pode ter sim uma aplicação no mundo real, principalmente com as nossas validações, um classificador conseguiria atuar bem identificando novos tráfegos de redes processados diretamente do PCAP. Para uma continuação do projeto, acreditamos que a melhor solução seria a classificação de cada tipo de ataque DDoS, porém, com o surgimento de novos ataques, temos que incluir essa variável em nossos testes. Para essa classificação mais avançada, um servidor conseguiria criar melhores regras de firewall para se proteger dos ataques.

Ademais, o conhecimento criado em ciência de dados implementando todas as técnicas e partes do projeto foi excelente para conseguirmos dar novos passos e construir classificadores e projetos muito mais complexos dentro da área.

Referências

CICFlowMeter. https://www.unb.ca/cic/research/applications.html#CICFlowMeter. Accessed: 2021-11-16.

DDoS Evaluation Dataset (CIC-DDoS2019). https://www.unb.ca/cic/datasets/ddos-2019.html. Accessed: 2021-11-16.

Hyperparameter Tuning in Neural Networks. https://towardsdatascience.com/... Accessed: 2021-12-02.

KNN - Scikit. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. Accessed: 2021-12-02.

MLP - Scikit. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. Accessed: 2021-12-02.

Random Forest - Scikit. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed: 2021-12-02.

Oliveira, L. S., Ceschin, F., and Grégio, A. (2019). Aprendizado de máquina para segurança: Algoritmos e aplicações. *SBSeg 2019*, pages 1–50.