Tópicos Especiais em Segurança da Informação

TP11 - IDS baseados em Aprendizado de Máquina

Arthur do Prado Labaki

09-08, 2022

GBC 235

Informações adicionais

Nem todos os exercícios estão com imagem aqui no relatório, mas todas as imagens integradas nesse relatório, quanto códigos, planilhas ou gifs de demonstração estão em meu repositório no GitHub abaixo.
Link do meu GitHub

Resolução do item 1)

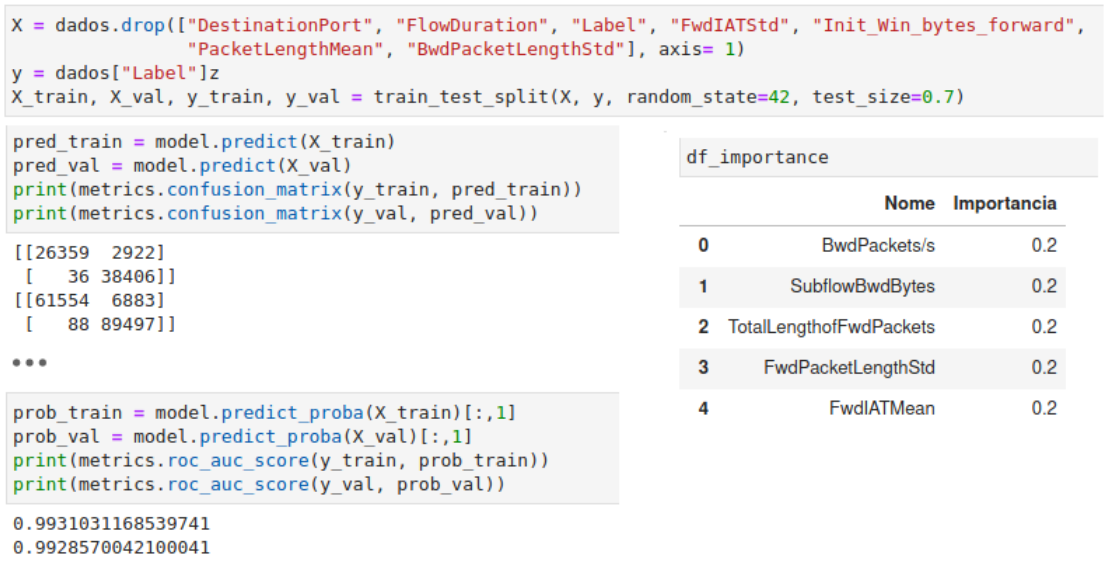
Precisamos comparar a performance do primeiro modelo com esse novo modelo sem as quatro primeiras variáveis. Para isso, verificamos que nosso modelo completo, temos como *score* de desempenho 0.9433391521521396 de treino e 0.9434432961011897 para teste, sendo sua matriz de confusão [[21608 7673] [48 38394]] [[50630 17807] [140 89445]]. Simplificando, temos que o teste valido de desempenho é de 94,3% aproximadamente.

Removendo as quatro primeiras variáveis em ordem de importância utilizada (*FwdIATStd, Init_Win_bytes_forward, PacketLengthMean e BwdPacketLengthStd*) utilizando o comando Drop: *"dados.drop(["FwdIATStd", "Init_Win_bytes_forward", "PacketLengthMean", "Bwd-PacketLengthStd"])"*, temos que o score de desempenho aumentou para 0.9931031168539741 de treino e 0.9928570042100041 para teste, com a matriz de confusão [[26359 2922] [36 38406]] [[61554 6883] [88 89497]].

Resumindo, foi possível obter melhor desempenho retirando as quatro primeiras variáveis do modelo, em que o número de predições corretas realizadas foi maior que o modelo anterior (obtendo menos falsos positivos ou negativos). Esse resultado foi possível pois o algoritmo de aprendizado de maquinas utilizou outras variáveis para realizar a predição.

Matriz de Confusão				
	Modelo Principal		Novo Modelo	
Treino	21608	7673	26359	2922
	48	38394	36	38406
Teste	50630	17807	61554	6883
	140	89445	88	89497

Figura 1: Tabela comparando os modelos



```
X = dados.drop(["DestinationPort", "FlowDuration", "Label", "FwdIATStd", "Init_Win_bytes_forward",
                "PacketLengthMean", "BwdPacketLengthStd"], axis= 1)
y = dados["Label"]z
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=42, test_size=0.7)
```

```
pred_train = model.predict(X_train)
pred_val = model.predict(X_val)
print(metrics.confusion_matrix(y_train, pred_train))
print(metrics.confusion_matrix(y_val, pred_val))
```

```
[[26359  2922]
 [   36 38406]]
[[61554  6883]
 [   88 89497]]
```

```
...
```

```
prob_train = model.predict_proba(X_train)[:,1]
prob_val = model.predict_proba(X_val)[:,1]
print(metrics.roc_auc_score(y_train, prob_train))
print(metrics.roc_auc_score(y_val, prob_val))
```

```
0.9931031168539741
0.9928570042100041
```

```
df_importance
```

	Nome	Importancia
0	BwdPackets/s	0.2
1	SubflowBwdBytes	0.2
2	TotalLengthofFwdPackets	0.2
3	FwdPacketLengthStd	0.2
4	FwdIATMean	0.2

Figura 2: Modelo novo sem as quatro variáveis

Resolução do item 2)

Sim, como foi mostrado no exercício anterior, podemos diminuir o numero de variáveis e manter ou até melhorar a performance. Podemo testar retirando mais algumas variáveis. Nesse exemplo, conseguimos manter o score de desempenho, retirando nove variáveis diferentes.

```
X = dados.drop(["DestinationPort", "FlowDuration", "Label", "FINFlagCount", "SYNFlagCount",
                "RSTFlagCount", "PSHFlagCount", "ACKFlagCount", "URGFlagCount",
                "CWEFlagCount", "ECEFlagCount", "Down/UpRatio" ], axis= 1)
y = dados["Label"]
X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=42, test_size=0.7)
```

```
pred_train = model.predict(X_train)
pred_val = model.predict(X_val)
print(metrics.confusion_matrix(y_train, pred_train))
print(metrics.confusion_matrix(y_val, pred_val))
```

```
[[26087  3194]
 [    8 38434]]
[[61032  7405]
 [   29 89556]]
...
```

```
prob_train = model.predict_proba(X_train)[:,1]
prob_val = model.predict_proba(X_val)[:,1]
print(metrics.roc_auc_score(y_train, prob_train))
print(metrics.roc_auc_score(y_val, prob_val))
```

```
0.9873457734903021
0.9873174764375356
```

```
df_importance
```

	Nome	Importancia
0	FwdIATStd	0.2
1	FwdIATMax	0.2
2	FwdIATMin	0.2
3	FwdPacketLengthMean	0.2
4	BwdPacketLengthMin	0.2

Figura 3: Modelo sem nove variáveis

Resolução do item 3)

No modelo demonstrado, os parâmetros do *Random Forest* são:

- *n_estimators:* O número de árvores na floresta;
- *max_depth:* A profundidade máxima da árvore;
- *max_features:* O número de recursos a serem considerados ao procurar a melhor divisão.

Podemos modifica-los para verificar se obtemos uma performance melhor. Analisando-os, podemo concluir que, quanto maior for o número de arvores e a sua profundidade (*n_estimators* e *max_depth*), melhor será a performance, porém o custo de processamento aumenta. Também, o numero de recursos utilizados depende do modelo que é utilizado, em que, nesse caso, o melhor número encontrado foi o 10. Maior que isso, a sua performance começa a cair, mostrando que existe variáveis que atrapalham o aprendizado da maquina.

Nos testes, o melhor resultado obtido foi com valor 100 no tamanho da arvore e sua profundidade e 5 no número de recursos: *model = RandomForestClassifier(n_estimators = 100,*

max_depth = 100, max_features = 5), em que foi possível obter *score* 1 (acertando todos os casos) no treino e 0.9999935623944943 no teste. Também vale lembrar que no repositório citado foi disponibilizado uma imagem com os testes feitos de cada parâmetro.

Resolução do item extra)

Hiperparâmetros são parâmetros de modelos que devem ser definidos antes de treinar o modelo. Para isso existem diferentes técnicas que buscam otimizá-los que resultará uma melhor acurácia. Encontra-los é uma tarefa complicada.

Uma maneira de encontra-los seria por forca bruta, ou seja, testar parâmetro por parâmetro, analisando e comparando performance. Pesquisando mais, essa técnica é chamada de *Grid search*, e consiste em testar todas as combinações possíveis dos hiperparâmetros, exaustivamente, em que é fornecido alguns valores de *input* e testar todas as combinações plotando em um plano cartesiano. Em seguida, selecionará os hiperparâmetros que obtiveram o menor erro.

Uma outra técnica possível seria escolher parâmetros aleatórios (*Random search*), em que a combinações aleatórias dos hiperparâmetros são usadas para encontrar a melhor solução para o modelo construído. Ele tenta combinações aleatórias de um intervalo de valores. Para otimizar com busca aleatória, a função é avaliada em um certo número de configurações aleatórias no espaço de parâmetros.