Trabalho Prático MPEI 2024/25

(Classificação de URLs – malignos e benignos)

Alexandre Andrade, 119279
Miguel Neto, 119302

Tema:

- Num mundo cada vez mais digital, são cada vez mais comuns os ataques cibernéticos, quer seja direcionado a grandes empresas, como para o "consumidor comum".
- Desde ataques de phishing, malware, defacement, entre outros, vários são os perigos que nos ameaçam constantemente. Uma das formas através do qual os criminosos podem conseguir ter acesso aos nossos dados é através de **links/URLs maliciosos**, muitas vezes disfarçados ou similares a URLs comuns.

  Como tal, o objetivo principal do nosso trabalho passa por criar um programa que consiga **detetar com a maior precisão possível URLs maliciosos** (e, consequentemente, URLs benignos) com base no(s) conjunto(s) de URLs passados, e nas características inerentes a cada um.
- Para tal, dividimos o programa em **3 componentes**: um algoritmo de Naïve Bayes (binário no caso), um Bloom Filter e um algoritmo de MinHash

Dataset:

- Para o problema em questão, achámos um dataset do Kaggle com mais de 600 000 URLs, divididos em 4 classes (benign, phishing, malware e defacement). Por questões de simplificação, juntámos os últimos 3 numa classe "malign".
- Com base nos URLs dados, procedemos com a extração de features deles (através de código em Python) como o tamanho do URL, presença de alguns caracteres especiais, entre outras. Inicialmente, tornámos todas estas features binárias, de modo a facilitar a aplicação do módulo de Naïve Bayes. Contudo, acabámos igualmente por desenvolver uma versão mista do dataset, com features binárias e numéricas.

url Actual url	type Class of malicious url
641119 unique values	benign 66% defacement 15% Other (126631) 19%
br-icloud.com.br	phishing
mp3raid.com/music/krizz_kaliko.html	benign
bopsecrets.org/rexroth/cr/1.htm	benign
http://www.garage-pirenne.be/index.php?option=com_content&view=article&id=70&vsig70_0=15	defacement

Fluxo do sistema:

1. **Pré-processamento:**
   - O sistema extrai as características do conjunto de URLs (comprimento, caracteres especiais, etc) e divide-o nas partes necessárias.
2. **Bloom FIlter:**
   - Verifica se os URLs já são conhecidos como URLs malignos ou benignos. Dependendo do resultado, são marcados imediatamente como malignos ou benignos.
3. **Naïve Bayes:**
   - Se não se verificar correspondência concreta em qualquer dos filtros de Bloom, os URLs são passados para o algoritmo de Naïve Bayes, que os analisa e calcula a probabilidade de serem malignos ou benignos.
4. **MinHash:**
   - O MinHash faz uma verificação de similaridade, comparando cada URL com URLs previamente identificados.
5. **Conclusão final:**
   - Com base nos resultados dos passos anteriores, o sistema classifica os URLs como malignos ou benignos.

Naïve Bayes:

- O Naïve Bayes é um algoritmo de classificação baseado no **Teorema de Bayes**, que assume independência condicional entre as características.
- Com a utilização deste algoritmo, pretendemos calcular a classe mais provável de um determinado URL, com base nas características do mesmo.
- Tendo em conta as características das features adotadas (todas convertidas para binário para facilitar a aplicação do algoritmo), recorremos à versão do Naïve Bayes Binário.
- Recorrendo a métricas como a precisão, o recall e o F1, conseguiremos avaliar a prestação do nosso algoritmo e calibrá-lo ou alterar as features escolhidas, se necessário, de modo a atingir a maior exatidão possível.

Naïve Bayes Binário vs Naïve Bayes Misto

- AMBOS OS ALGORITMOS POSSUEM UMA ELEVADA PRESTAÇÃO (F1 SCORE ACIMA DE 0.835)

- O NAÏVE BAYES MISTO POSSUI MELHOR PRECISÃO, MAS É PIOR NAS RESTANTES CATEGORIAS.

- ISTO PODE DEVER-SE AO FACTO DE OS DADOS NUMÉRICOS DO DATASET PODEREM NÃO SER OS MAIS ADEQUADOS PARA SE APLICAR A FÓRMULA GAUSSIANA

	Accuracy	**Precisão**	**Recall**	$F_1 = \frac{2PR}{P+R}$
	0,8401	0,8475	0,9224	0,8833
	0,8409	0,8484	0,9231	0,8842
	0,8437	0,8505	0,9251	0,8863
	0,842	0,849	0,9244	0,8851
	0,8412	0,848	0,9245	0,8846
	0,8425	0,8488	0,9246	0,8851
	0,8415	0,8482	0,924	0,8844
	0,8424	0,8488	0,9248	0,8852
	0,8411	0,8477	0,9238	0,8841
	0,8421	0,8496	0,9238	0,8852
Média	0,84175	0,8486	0,9242	0,88485

	Accuracy	**Precisão**	**Recall**	$F_1 = \frac{2PR}{P+R}$
	0,7927	0,8638	0,8136	0,8379
	0,7938	0,8622	0,8167	0,8388
	0,7951	0,864	0,8172	0,84
	0,7958	0,8651	0,8168	0,8402
	0,7932	0,8638	0,8141	0,8382
	0,7936	0,8647	0,815	0,8392
	0,7931	0,8624	0,8152	0,8381
	0,796	0,8647	0,8178	0,8406
	0,7941	0,863	0,8154	0,8385
Média	0,7945	0,8644	0,8156	0,8393

Bloom Filter:

- O Bloom Filter é uma estrutura de dados usada para verificar a presença de um elemento num determinado conjunto. São usadas múltiplas funções hash para mapear elementos num vetor de bits (0s e 1s).
- Para o nosso problema em questão, dadas as propriedades do Bloom FIlter, decidimos dividir esta estrutura em duas, criando **um Bloom FIlter para URLs malignos**, e **um para URLs benignos**. Com isto feito, começávamos por verificar se os URLs de teste pertenciam a algum dos Bloom Filters. Se pertencessem a apenas um deles, podíamos proceder com a classificação desse URL tendo em conta o Bloom Filter correspondente, evitando passar esse URL para o algoritmo de Naïve Bayes.
- Caso contrário, seriam encaminhados para o classificador, onde lhes seria determinada a classe.
- Ou seja, como o próprio nome indica, serviriam como uma primeira "filtragem", antes do algoritmo de Naïve Bayes.

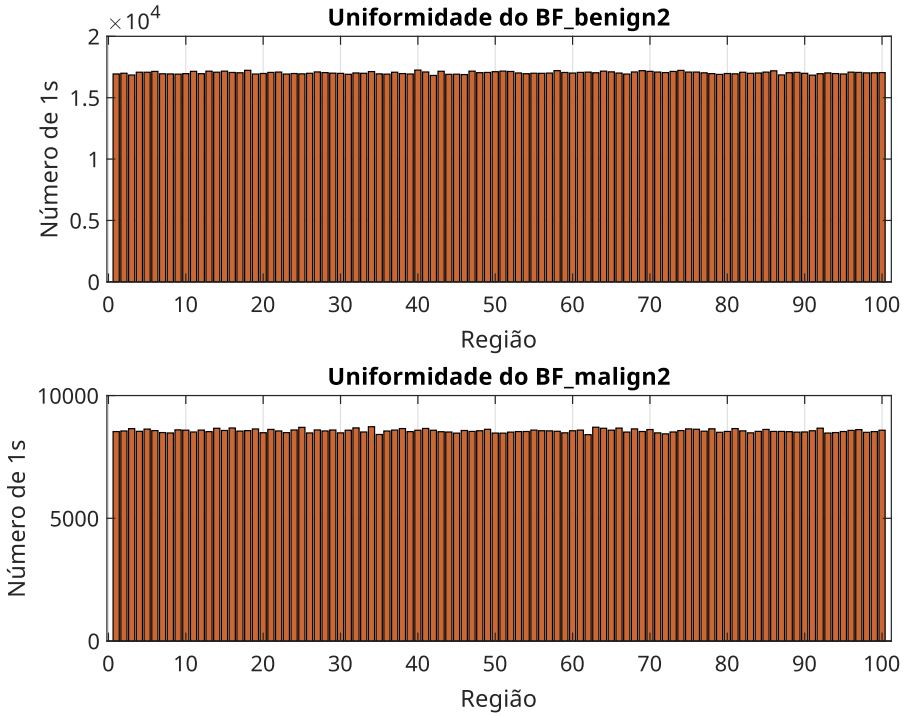
```
% -----------------BloomFilter setup---------------------
BF_malign = BloomInit(m_Malign);
BF_benign = BloomInit(m_Benign);

for i=1:length(Urls_MTr)
    BF_malign = BloomAdd2(Urls_MTr{i},BF_malign,kOtimoM);
end

for i=1:length(Urls_BTr)
    BF_benign = BloomAdd2(Urls_BTr{i},BF_benign,kOtimoB);
end
```

Explicação módulos + breves resultados

- Na implementação do Bloom filter, podemos destacar:
  - Uso de uma matriz de valores lógicos
  - Função de hash implementada com recurso a double hashing, sendo usadas as versões djb2 e sdbm da função string2hash como as funções envolvidas

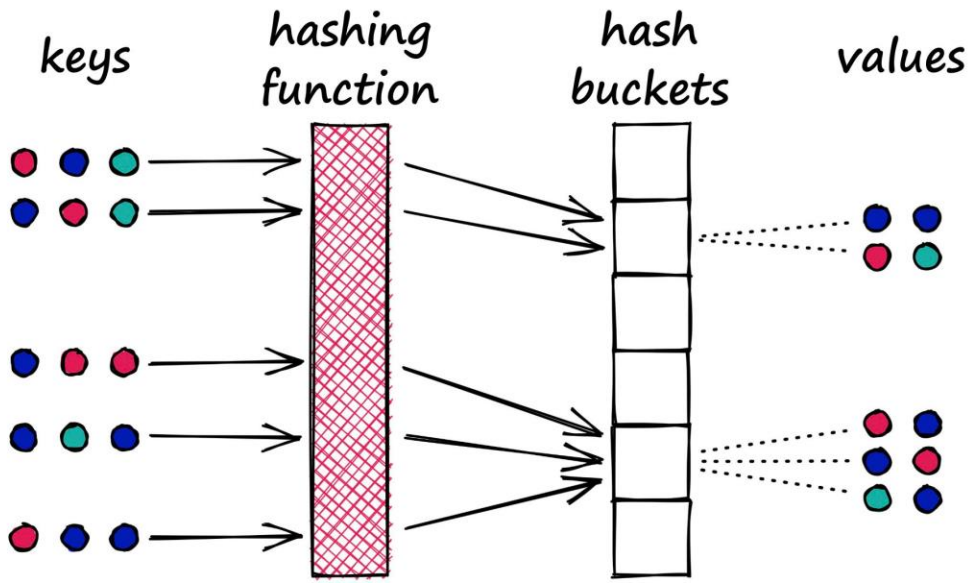


MinHash:

- O MinHash é uma técnica utilizada para estimar rapidamente a similaridade entre dois conjuntos.
- Isto envolve a associação desses mesmos conjuntos, que podem ter ou não dimensões radicais, a pequenas assinaturas geradas a partir de funções rápidas, de modo a poupar memória RAM, por exemplo. Mantém-se então propriedades como a similaridade de conjuntos.
- No problema em concreto, o MinHash compara um URL incitado àqueles já existentes, para concluir se é similar ou não a outros.
- Torna-se uma ferramenta poderosa para indicar se será ou não maligno - conclusão importante a tirar de todo o processamento do URL que foi dado.

Explicação módulos + breves resultados

- Na implementação do MinHash, podemos destacar:
  - Criação de uma função de hash personalizada
  - Recurso ao LSH de modo a aumentar a performance do programa



Aplicações do programa + Considerações

- Um exemplo prático de aplicação poderia ser nas redes escolares, do ensino básico e secundário, que por norma utilizam uma rede "MinEdu", constantemente a ser utilizada para aceder a diferentes websites na Internet.
- Com um sistema destes, os estabelecimentos de ensino teriam então uma ferramenta quase automática que avaliava todos os URLs acedidos pelas suas redes pela primeira vez, determinando assim se deve ser uma página acessível ou não dentro de uma escola, para o bem do meio estudantil.
- Contudo, o nosso sistema também possui limitações, como elevado tempo de execução e consumo de RAM, bem como as limitações associadas à classificação de URLs apenas pelo seu conteúdo lexical.