BDA Project: Malicious and Benign Website URL detections

Nguyen Xuan Binh

January 2023

Table of Contents

# Introduction

## Central problem

Web Security is a challenging task amidst ever rising threats on the Internet. With billions of websites active on Internet, and hackers evolving newer techniques to trap web users, machine learning offers promising techniques to detect malicious websites. The dataset described in this manuscript is meant for such machine learning based analysis of malicious and benign webpages. The data has been collected from Internet using a specialized focused web crawler named MalCrawler [1]. The dataset comprises of various extracted attributes, and also raw webpage content including JavaScript code. It supports both supervised and unsupervised learning. For supervised learning, class labels for malicious and benign webpages have been added to the dataset using the Google Safe Browsing API.1 The most relevant attributes within the scope have already been extracted and included in this dataset. However, the raw web content, including JavaScript code included in this dataset supports further attribute extraction, if so desired. Also, this raw content and code can be used as unstructured data input for text-based analytics. This dataset consists of data from approximately 1.5 million webpages, which makes it suitable for deep learning algorithms. This article also provides code snippets used for data extraction and its analysis.

## Motivation

## Main modeling idea

# Dataset

## Data Description

The dataset contains extracted attributes from websites that can be used for Classification of webpages as malicious or benign. The dataset also includes raw page content including JavaScript code that can be used as unstructured data in Deep Learning or for extracting further attributes. The data has been collected by crawling the Internet using MalCrawler [1]. The labels have been verified using the Google Safe Browsing API [2]. Attributes have been selected based on their relevance [3]. The details of dataset attributes is as given below: ‘url’ - The URL of the webpage. ‘ip\_add’ - IP Address of the webpage. ‘geo\_loc’ - The geographic location where the webpage is hosted. ‘url\_len’ - The length of URL. ‘js\_len’ - Length of JavaScript code on the webpage. ‘js\_obf\_len - Length of obfuscated JavaScript code. ’tld’ - The Top Level Domain of the webpage. ‘who\_is’ - Whether the WHO IS domain information is compete or not. ‘https’ - Whether the site uses https or http. ‘content’ - The raw webpage content including JavaScript code. ‘label’ - The class label for benign or malicious webpage.

Python code for extraction of the above listed dataset attributes is attached. The Visualisation of this dataset and it python code is also attached. This visualisation can be seen online on Kaggle

## Data source and analysis difference

Kaggle: <https://www.kaggle.com/datasets/aksingh2411/dataset-of-malicious-and-benign-webpages> Data source: <https://data.mendeley.com/datasets/gdx3pkwp47/2> <https://www.researchgate.net/publication/347936136_Malicious_and_Benign_Webpages_Dataset>

# Data cleaning

# Feature selection and transformation

train\_websites <- read.csv("websites/train\_websites.csv")  
test\_websites <- read.csv("websites/test\_websites.csv")  
train\_websites\_top\_3 <- read.csv("websites/train\_websites\_top\_3.csv")  
test\_websites\_top\_3 <- read.csv("websites/test\_websites\_top\_3.csv")

cat("Number of training data:",nrow(train\_websites\_top\_3))

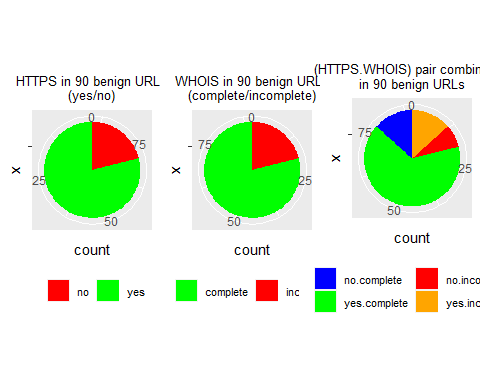
## Number of training data: 62

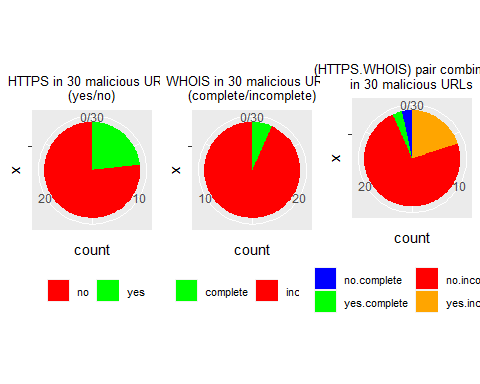
cat("\nNumber of testing data:",nrow(test\_websites\_top\_3))

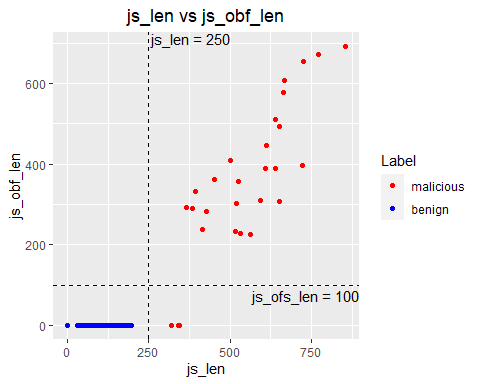
##   
## Number of testing data: 131

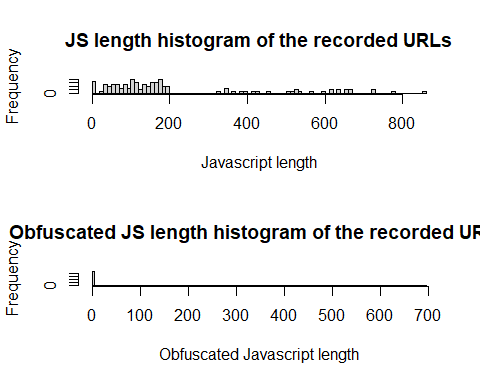
head(train\_websites\_top\_3)

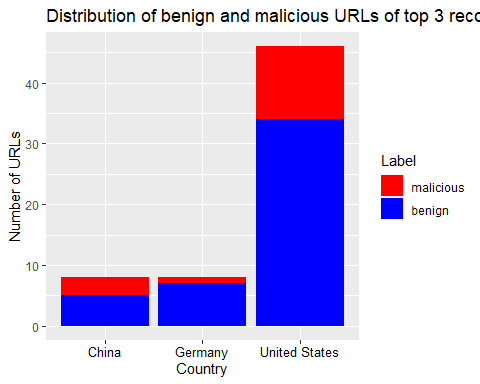
## X label geo\_loc js\_len js\_obf\_len https https\_bin whois whois\_bin  
## 1 1 bad China 594.0 308.880 yes 0 incomplete 1  
## 2 2 bad United States 610.2 390.528 no 1 incomplete 1  
## 3 3 bad Germany 668.7 608.517 no 1 incomplete 1  
## 4 4 bad United States 453.6 362.880 no 1 incomplete 1  
## 5 5 bad United States 0.0 0.000 no 1 incomplete 1  
## 6 6 bad China 340.2 0.000 no 1 incomplete 1  
## js\_len\_bin js\_obf\_len\_bin label\_bin  
## 1 1 1 1  
## 2 1 1 1  
## 3 1 1 1  
## 4 1 1 1  
## 5 0 0 1  
## 6 1 0 1











# Separate model

## Model description

## Prior choice and justifications

Default protocol https is used by 81.5% of all the websites. (**https://w3techs.com/technologies/details/ce-httpsdefault?**)

There are 1.24 billion with complete WHOIS registration, while there are currently 1.7 billion websites. So the ratio of complete WHOIS website is 0.73.

## Stan code and running options

The Stan model code:

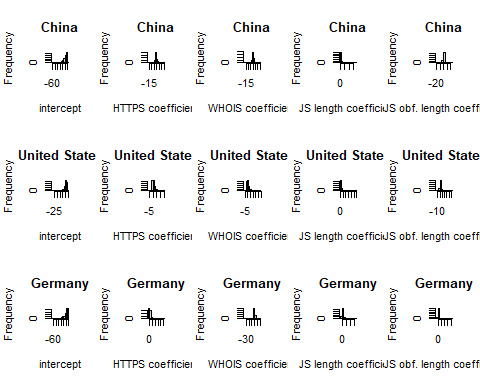
"  
data {  
 int<lower=1> Nmax; // Number of maximum URLs among all countries (training)  
 int<lower=1> Mmax; // Number of maximum URLs among all countries (testing)  
 int<lower=1> K; // Number of countries  
 array[K] int<lower=1> N\_list; // Number of URLs of each country (training)  
 array[K] int<lower=1> M\_list; // Number of URLs of each country (testing)  
 // The training features  
 array[K, Nmax] int<lower=0,upper=1> js\_len\_list;  
 array[K, Nmax] int<lower=0,upper=1> js\_obf\_len\_list;  
 array[K, Nmax] int<lower=0,upper=1> https\_list;  
 array[K, Nmax] int<lower=0,upper=1> whois\_list;  
 // The testing predicting features  
 array[K, Mmax] int<lower=0,upper=1> js\_len\_pred\_list;  
 array[K, Mmax] int<lower=0,upper=1> js\_obf\_len\_pred\_list;  
 array[K, Mmax] int<lower=0,upper=1> https\_pred\_list;  
 array[K, Mmax] int<lower=0,upper=1> whois\_pred\_list;  
 // label for each URL: benign(0) or malicious(1)  
 array[K, Nmax] int<lower=0,upper=1> label\_list;   
}  
  
parameters {  
 array[K] real<lower=0, upper=1> theta\_js\_len; // probability for js\_len  
 array[K] real<lower=0, upper=1> theta\_js\_obf\_len; // probability for js\_obf\_len  
 array[K] real<lower=0, upper=1> theta\_https; // probability for https  
 array[K] real<lower=0, upper=1> theta\_whois; // probability for whois  
 array[K] real js\_len\_coeff; // Slope coefficient for js\_len  
 array[K] real js\_obf\_len\_coeff; // Slope coefficient for js\_obf\_len  
 array[K] real https\_coeff; // Slope coefficient for https\_coeff  
 array[K] real whois\_coeff; // Slope coefficient for whois\_coeff  
 array[K] real intercept; // Intercept coefficient  
}  
  
model {  
 // Prior probabilities of the features  
 for (k in 1:K){  
 theta\_js\_len[k] ~ beta(1,10);  
 theta\_js\_obf\_len[k] ~ beta(1,10);  
 theta\_https[k] ~ beta(8,10);  
 theta\_whois[k] ~ beta(7,10);  
 }  
 // likelihood for the features  
 for (k in 1:K){  
 js\_len\_list[k, 1:N\_list[k]] ~ bernoulli(theta\_js\_len[K]);  
 js\_obf\_len\_list[k, 1:N\_list[k]] ~ bernoulli(theta\_js\_obf\_len[K]);  
 https\_list[k, 1:N\_list[k]] ~ bernoulli(theta\_https[K]);  
 whois\_list[k, 1:N\_list[k]] ~ bernoulli(theta\_whois[K]);   
 }  
 // priors of the coefficients  
 for (k in 1:K){  
 js\_len\_coeff[k] ~ cauchy(1,1);  
 js\_obf\_len\_coeff[k] ~ cauchy(1,1);  
 https\_coeff[k] ~ cauchy(-1,1);  
 whois\_coeff[k] ~ cauchy(-1,1);  
 intercept[k] ~ normal(0,20);   
 }  
 // Modelling of the label based on bernoulli logistic regression by   
 // multiple variable linear regression   
 for (k in 1:K){  
 for (i in 1:N\_list[k]){  
 label\_list[k, i] ~ bernoulli(inv\_logit(intercept[k]   
 + https\_coeff[k] \* https\_list[k, i]  
 + whois\_coeff[k] \* whois\_list[k, i]   
 + js\_len\_coeff[k] \* js\_len\_list[k, i]   
 + js\_obf\_len\_coeff[k] \* js\_obf\_len\_list[k, i]));  
 }  
 }  
}  
  
generated quantities {  
 array[K, Nmax] real label\_train\_pred;  
 array[K, Mmax] real label\_test\_pred;  
 array[Nmax] real log\_likelihood;   
 // Predictions for the training data  
 for (k in 1:K){  
 for (i in 1:N\_list[k]){  
 label\_train\_pred[k, i] = bernoulli\_rng(inv\_logit(intercept[k]   
 + https\_coeff[k] \* https\_list[k, i]   
 + whois\_coeff[k] \* whois\_list[k, i]   
 + js\_len\_coeff[k] \* js\_len\_list[k, i]   
 + js\_obf\_len\_coeff[k] \* js\_obf\_len\_list[k, i]));  
 }  
 }  
 // Predictions for the testing data  
 for (k in 1:K){  
 for (i in 1:M\_list[k]){  
 label\_test\_pred[k, i] = bernoulli\_rng(inv\_logit(intercept[k]   
 + https\_coeff[k] \* https\_pred\_list[k, i]   
 + whois\_coeff[k] \* whois\_pred\_list[k, i]   
 + js\_len\_coeff[k] \* js\_len\_pred\_list[k, i]   
 + js\_obf\_len\_coeff[k] \* js\_obf\_len\_pred\_list[k, i]));  
 }  
 }  
 for (k in 1:K) {  
 if (N\_list[k] == Nmax){  
 for (i in 1:Nmax){  
 log\_likelihood[i] = bernoulli\_lpmf(label\_list[k, i] | inv\_logit(intercept[k]   
 + https\_coeff[k] \* https\_list[k, i]   
 + whois\_coeff[k] \* whois\_list[k, i]   
 + js\_len\_coeff[k] \* js\_len\_list[k, i]   
 + js\_obf\_len\_coeff[k] \* js\_obf\_len\_list[k, i]));  
 }  
 }  
 }  
}  
  
"

The sampling running options

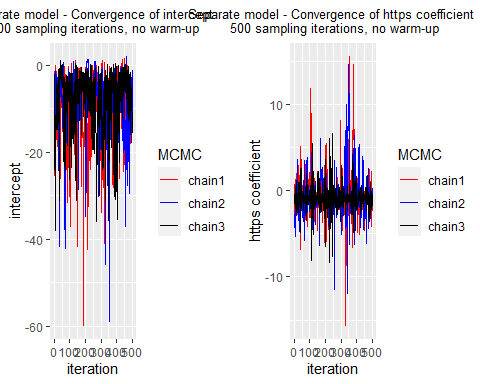
separate\_sampling <- model\_separate$sample(data = stan\_data, chains=3, iter\_warmup = 1000, iter\_sampling = 500)#, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 Iteration: 1 / 1500 [ 0%] (Warmup)   
## Chain 1 Iteration: 100 / 1500 [ 6%] (Warmup)   
## Chain 1 Iteration: 200 / 1500 [ 13%] (Warmup)   
## Chain 1 Iteration: 300 / 1500 [ 20%] (Warmup)   
## Chain 1 Iteration: 400 / 1500 [ 26%] (Warmup)   
## Chain 1 Iteration: 500 / 1500 [ 33%] (Warmup)   
## Chain 1 Iteration: 600 / 1500 [ 40%] (Warmup)   
## Chain 1 Iteration: 700 / 1500 [ 46%] (Warmup)   
## Chain 1 Iteration: 800 / 1500 [ 53%] (Warmup)   
## Chain 1 Iteration: 900 / 1500 [ 60%] (Warmup)   
## Chain 1 Iteration: 1000 / 1500 [ 66%] (Warmup)   
## Chain 1 Iteration: 1001 / 1500 [ 66%] (Sampling)   
## Chain 1 Iteration: 1100 / 1500 [ 73%] (Sampling)   
## Chain 1 Iteration: 1200 / 1500 [ 80%] (Sampling)   
## Chain 1 Iteration: 1300 / 1500 [ 86%] (Sampling)   
## Chain 1 Iteration: 1400 / 1500 [ 93%] (Sampling)   
## Chain 1 Iteration: 1500 / 1500 [100%] (Sampling)   
## Chain 1 finished in 6.7 seconds.  
## Chain 2 Iteration: 1 / 1500 [ 0%] (Warmup)   
## Chain 2 Iteration: 100 / 1500 [ 6%] (Warmup)   
## Chain 2 Iteration: 200 / 1500 [ 13%] (Warmup)   
## Chain 2 Iteration: 300 / 1500 [ 20%] (Warmup)   
## Chain 2 Iteration: 400 / 1500 [ 26%] (Warmup)   
## Chain 2 Iteration: 500 / 1500 [ 33%] (Warmup)   
## Chain 2 Iteration: 600 / 1500 [ 40%] (Warmup)   
## Chain 2 Iteration: 700 / 1500 [ 46%] (Warmup)   
## Chain 2 Iteration: 800 / 1500 [ 53%] (Warmup)   
## Chain 2 Iteration: 900 / 1500 [ 60%] (Warmup)   
## Chain 2 Iteration: 1000 / 1500 [ 66%] (Warmup)   
## Chain 2 Iteration: 1001 / 1500 [ 66%] (Sampling)   
## Chain 2 Iteration: 1100 / 1500 [ 73%] (Sampling)   
## Chain 2 Iteration: 1200 / 1500 [ 80%] (Sampling)   
## Chain 2 Iteration: 1300 / 1500 [ 86%] (Sampling)   
## Chain 2 Iteration: 1400 / 1500 [ 93%] (Sampling)   
## Chain 2 Iteration: 1500 / 1500 [100%] (Sampling)   
## Chain 2 finished in 11.7 seconds.  
## Chain 3 Iteration: 1 / 1500 [ 0%] (Warmup)   
## Chain 3 Iteration: 100 / 1500 [ 6%] (Warmup)   
## Chain 3 Iteration: 200 / 1500 [ 13%] (Warmup)   
## Chain 3 Iteration: 300 / 1500 [ 20%] (Warmup)   
## Chain 3 Iteration: 400 / 1500 [ 26%] (Warmup)   
## Chain 3 Iteration: 500 / 1500 [ 33%] (Warmup)   
## Chain 3 Iteration: 600 / 1500 [ 40%] (Warmup)   
## Chain 3 Iteration: 700 / 1500 [ 46%] (Warmup)   
## Chain 3 Iteration: 800 / 1500 [ 53%] (Warmup)   
## Chain 3 Iteration: 900 / 1500 [ 60%] (Warmup)   
## Chain 3 Iteration: 1000 / 1500 [ 66%] (Warmup)   
## Chain 3 Iteration: 1001 / 1500 [ 66%] (Sampling)   
## Chain 3 Iteration: 1100 / 1500 [ 73%] (Sampling)   
## Chain 3 Iteration: 1200 / 1500 [ 80%] (Sampling)   
## Chain 3 Iteration: 1300 / 1500 [ 86%] (Sampling)   
## Chain 3 Iteration: 1400 / 1500 [ 93%] (Sampling)   
## Chain 3 Iteration: 1500 / 1500 [100%] (Sampling)   
## Chain 3 finished in 15.8 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 11.4 seconds.  
## Total execution time: 34.5 seconds.

## Warning: 14 of 1500 (1.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.



## Convergence diagnostics

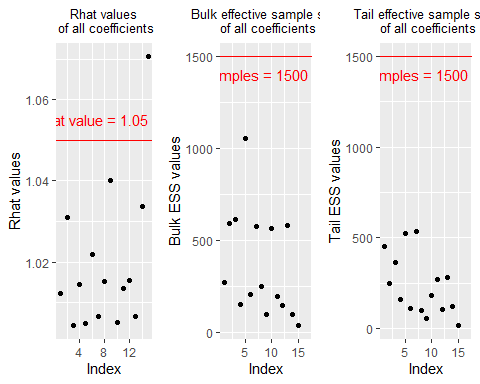
MCMC convergence chains visualization 

HMC specific convergence diagnostics

separate\_sampling$diagnostic\_summary()

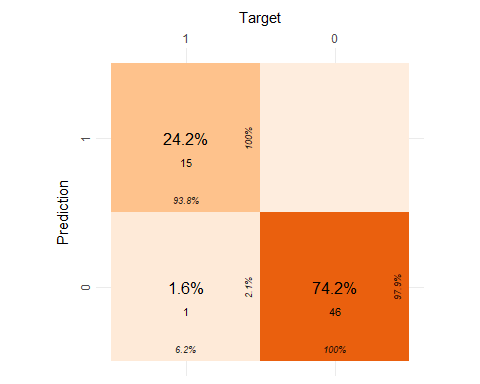
## Warning: 14 of 1500 (1.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.

## $num\_divergent  
## [1] 0 0 0  
##   
## $num\_max\_treedepth  
## [1] 1 13 0  
##   
## $ebfmi  
## [1] 0.4417014 0.4989904 0.5557695

-values and effective sample size  ## Posterior predictive checks

## Metrics China United States Germany All countries  
## 1 Accuracy 1 0.9782609 1 0.9838710  
## 2 Precision 1 0.9166667 1 0.9375000  
## 3 Recall 1 1.0000000 1 1.0000000  
## 4 F1 1 0.9565217 1 0.9677419

confusion\_matrix <- tibble("actual" = listTrue,  
 "prediction" = listPred)  
  
  
basic\_table <- table(confusion\_matrix)  
cfm <- as\_tibble(basic\_table)  
plot\_confusion\_matrix(cfm,   
 target\_col = "actual",   
 prediction\_col = "prediction",  
 counts\_col = "n", palette = "Oranges")

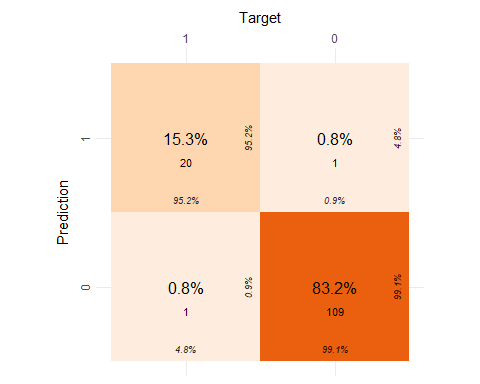


## Predictive performance assessment

## Metrics China United States Germany All countries  
## 1 Accuracy 1 0.9891304 0.9166667 0.9847328  
## 2 Precision 1 0.9444444 1.0000000 0.9523810  
## 3 Recall 1 1.0000000 0.5000000 0.9523810  
## 4 F1 1 0.9714286 0.6666667 0.9523810

## Warning in plot\_confusion\_matrix(cfm, target\_col = "actual", prediction\_col =  
## "prediction", : 'ggimage' is missing. Will not plot arrows and zero-shading.

## Warning in plot\_confusion\_matrix(cfm, target\_col = "actual", prediction\_col =  
## "prediction", : 'rsvg' is missing. Will not plot arrows and zero-shading.



## Prior sensitivity analysis

The sampling running options

separate\_sampling\_prior\_sensitivity1 <- model\_separate\_prior\_sensitivity1$sample(data = stan\_data, chains=3, iter\_warmup = 1000, iter\_sampling = 500, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 finished in 5.0 seconds.  
## Chain 2 finished in 16.4 seconds.  
## Chain 3 finished in 6.7 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 9.4 seconds.  
## Total execution time: 28.3 seconds.

## Warning: 501 of 1500 (33.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.

separate\_sampling\_prior\_sensitivity2 <- model\_separate\_prior\_sensitivity2$sample(data = stan\_data, chains=3, iter\_warmup = 1000, iter\_sampling = 500, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 finished in 6.4 seconds.  
## Chain 2 finished in 9.9 seconds.  
## Chain 3 finished in 5.1 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 7.1 seconds.  
## Total execution time: 21.6 seconds.

## The elpd value of the separate model 0 is

## [1] -6.885446

## The elpd value of the separate model 1 is

## [1] -6.293416

## The elpd value of the separate model 2 is

## [1] -6.161439

loo\_compare\_separate <- loo\_compare(x = list(loo\_loglike\_separate0=loo\_loglike\_separate0, loo\_loglike\_separate1=loo\_loglike\_separate1, loo\_loglike\_separate2=loo\_loglike\_separate2))  
print(loo\_compare\_separate)

## elpd\_diff se\_diff  
## loo\_loglike\_separate2 0.0 0.0   
## loo\_loglike\_separate1 -0.1 0.2   
## loo\_loglike\_separate0 -0.7 0.6

We know that elpd\_loo is the Bayesian LOO estimate of the expected log pointwise predictive density and is a sum of N individual pointwise log predictive densities. From the comparison table, elpd\_diff is the difference in elpd\_loo for two models. If more than two models are compared, the difference is computed relative to the model with highest elpd\_loo, which is true in this case, as I am comparing three models

The standard error of component-wise differences of elpd\_loo (Eq 24 in VGG2017) between two models. This SE is smaller than the SE for individual models due to correlation (i.e., if some observations are easier and some more difficult to predict for all models)

As quick rule: If elpd difference (elpd\_diff in loo package) is less than 4, the difference is small. If elpd difference (elpd\_diff in loo package) is larger than 4, then compare that difference to standard error of elpd\_diff. When the difference (elpd\_diff) is larger than 4, the number of observations is larger than 100 and the model is not badly misspecified then normal approximation and SE are quite reliable description of the uncertainty in the difference. Differences smaller than 4 are small and then the models have very similar predictive performance and it doesn’t matter if the normal approximation fails or SE is underestimated Sivula et al. (2020).

SE assumes that normal approximation describes well the uncertainty related to the expected difference. Due to cross-validation folds not being independent, SE tends to be underestimated especially if the number of observations is small or the models are badly misspecified. The whole normal approximation tends to fail if the models are very similar or the models are badly misspecified.

# Pooled model

## Model description

## Prior choice and justifications

Default protocol https is used by 81.5% of all the websites. (**https://w3techs.com/technologies/details/ce-httpsdefault?**)

There are 1.24 billion with complete WHOIS registration, while there are currently 1.7 billion websites. So the ratio of complete WHOIS website is 0.73.

## Stan code and running options

The Stan model code:

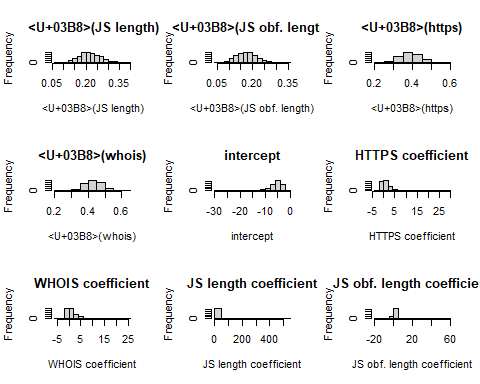
"  
  
"

The sampling running options

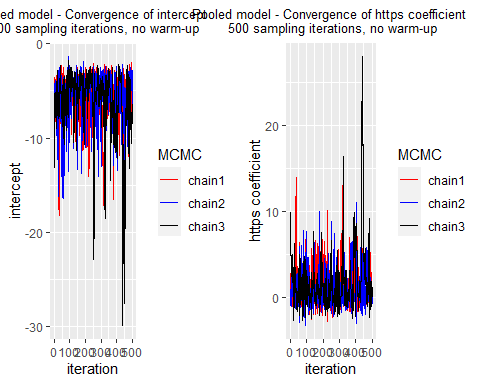
pooled\_sampling <- model\_pooled$sample(data = stan\_data, chains=3,   
 iter\_warmup = 500, iter\_sampling = 1000, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 finished in 2.3 seconds.  
## Chain 2 finished in 7.1 seconds.  
## Chain 3 finished in 7.4 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 5.6 seconds.  
## Total execution time: 17.2 seconds.

## Warning: 16 of 3000 (1.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.



## Convergence diagnostics

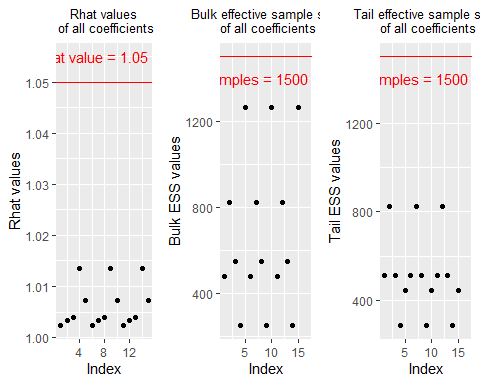
MCMC convergence chains visualization 

HMC specific convergence diagnostics

pooled\_sampling$diagnostic\_summary()

## Warning: 16 of 3000 (1.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.

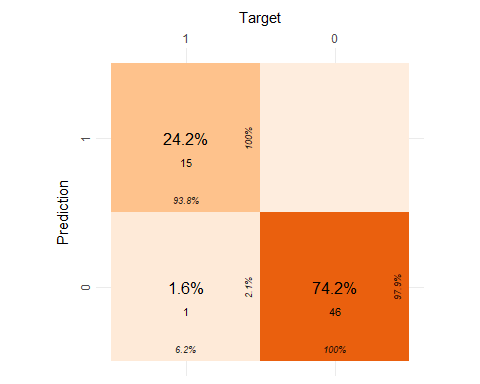
## $num\_divergent  
## [1] 0 0 0  
##   
## $num\_max\_treedepth  
## [1] 0 12 4  
##   
## $ebfmi  
## [1] 0.6186550 0.6764700 0.7654839

-values and effective sample size 

## Posterior predictive checks

## Metrics China United States Germany All countries  
## 1 Accuracy 1 0.9782609 1 0.9838710  
## 2 Precision 1 0.9166667 1 0.9375000  
## 3 Recall 1 1.0000000 1 1.0000000  
## 4 F1 1 0.9565217 1 0.9677419

confusion\_matrix <- tibble("actual" = listTrue,  
 "prediction" = listPred)  
  
  
basic\_table <- table(confusion\_matrix)  
cfm <- as\_tibble(basic\_table)  
plot\_confusion\_matrix(cfm,   
 target\_col = "actual",   
 prediction\_col = "prediction",  
 counts\_col = "n", palette = "Oranges")

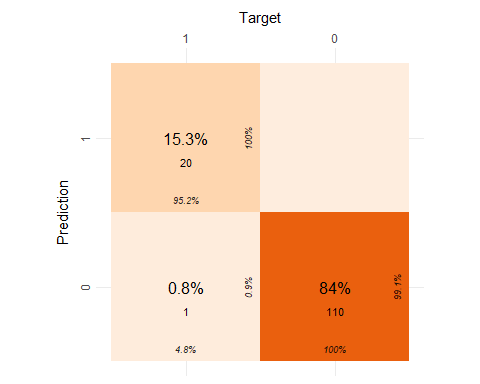


## Predictive performance assessment

## Metrics China United States Germany All countries  
## 1 Accuracy 1 0.9891304 1 0.9923664  
## 2 Precision 1 0.9444444 1 0.9523810  
## 3 Recall 1 1.0000000 1 1.0000000  
## 4 F1 1 0.9714286 1 0.9756098

## Warning in plot\_confusion\_matrix(cfm, target\_col = "actual", prediction\_col =  
## "prediction", : 'ggimage' is missing. Will not plot arrows and zero-shading.

## Warning in plot\_confusion\_matrix(cfm, target\_col = "actual", prediction\_col =  
## "prediction", : 'rsvg' is missing. Will not plot arrows and zero-shading.



## Prior sensitivity analysis

The sampling running options

pooled\_sampling\_prior\_sensitivity1 <- model\_pooled\_prior\_sensitivity1$sample(data = stan\_data, chains=3, iter\_warmup = 1000, iter\_sampling = 500, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 finished in 2.0 seconds.  
## Chain 2 finished in 6.2 seconds.  
## Chain 3 finished in 3.1 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 3.8 seconds.  
## Total execution time: 11.5 seconds.

## Warning: 2 of 1500 (0.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.

pooled\_sampling\_prior\_sensitivity2 <- model\_pooled\_prior\_sensitivity2$sample(data = stan\_data, chains=3, iter\_warmup = 1000, iter\_sampling = 500, refresh=0)

## Running MCMC with 3 sequential chains...  
##   
## Chain 1 finished in 2.4 seconds.  
## Chain 2 finished in 7.1 seconds.  
## Chain 3 finished in 3.1 seconds.  
##   
## All 3 chains finished successfully.  
## Mean chain execution time: 4.2 seconds.  
## Total execution time: 13.0 seconds.

## Warning: 2 of 1500 (0.0%) transitions hit the maximum treedepth limit of 10.  
## See https://mc-stan.org/misc/warnings for details.

## The elpd value of the pooled model 0 is

## [1] -7.402518

## The elpd value of the pooled model 1 is

## [1] -6.126398

## The elpd value of the pooled model 2 is

## [1] -5.556738

loo\_compare\_pooled <- loo\_compare(x = list(loo\_loglike\_pooled0=loo\_loglike\_pooled0, loo\_loglike\_pooled1=loo\_loglike\_pooled1, loo\_loglike\_pooled2=loo\_loglike\_pooled2))  
print(loo\_compare\_pooled)

## elpd\_diff se\_diff  
## loo\_loglike\_pooled2 0.0 0.0   
## loo\_loglike\_pooled1 -0.6 0.7   
## loo\_loglike\_pooled0 -1.8 2.0

We know that elpd\_loo is the Bayesian LOO estimate of the expected log pointwise predictive density and is a sum of N individual pointwise log predictive densities. From the comparison table, elpd\_diff is the difference in elpd\_loo for two models. If more than two models are compared, the difference is computed relative to the model with highest elpd\_loo, which is true in this case, as I am comparing three models

The standard error of component-wise differences of elpd\_loo (Eq 24 in VGG2017) between two models. This SE is smaller than the SE for individual models due to correlation (i.e., if some observations are easier and some more difficult to predict for all models)

As quick rule: If elpd difference (elpd\_diff in loo package) is less than 4, the difference is small. If elpd difference (elpd\_diff in loo package) is larger than 4, then compare that difference to standard error of elpd\_diff. When the difference (elpd\_diff) is larger than 4, the number of observations is larger than 100 and the model is not badly misspecified then normal approximation and SE are quite reliable description of the uncertainty in the difference. Differences smaller than 4 are small and then the models have very similar predictive performance and it doesn’t matter if the normal approximation fails or SE is underestimated Sivula et al. (2020).

SE assumes that normal approximation describes well the uncertainty related to the expected difference. Due to cross-validation folds not being independent, SE tends to be underestimated especially if the number of observations is small or the models are badly misspecified. The whole normal approximation tends to fail if the models are very similar or the models are badly misspecified.

# Model comparison

## The PSIS-LOO elpd value of the separate model is

## [1] -6.885446

##   
## The k-hat diagnostics of the separate model is

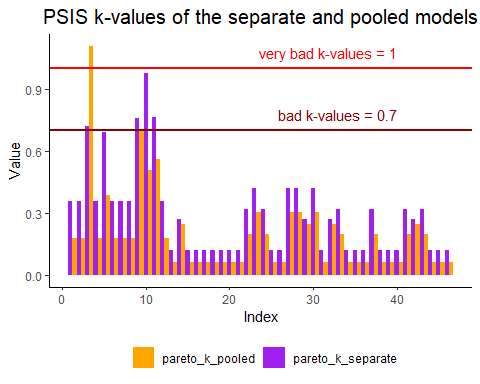
##   
## Computed from 1500 by 46 log-likelihood matrix  
##   
## Estimate SE  
## elpd\_loo -6.9 4.9  
## p\_loo 2.9 2.5  
## looic 13.8 9.9  
## ------  
## Monte Carlo SE of elpd\_loo is NA.  
##   
## Pareto k diagnostic values:  
## Count Pct. Min. n\_eff  
## (-Inf, 0.5] (good) 41 89.1% 625   
## (0.5, 0.7] (ok) 1 2.2% 1365   
## (0.7, 1] (bad) 4 8.7% 29   
## (1, Inf) (very bad) 0 0.0% <NA>   
## See help('pareto-k-diagnostic') for details.

## The PSIS-LOO elpd value of the pooled model is

## [1] -7.402518

##   
## The k-hat diagnostics of the pooled model is

##   
## Computed from 3000 by 46 log-likelihood matrix  
##   
## Estimate SE  
## elpd\_loo -7.4 6.2  
## p\_loo 3.6 3.5  
## looic 14.8 12.4  
## ------  
## Monte Carlo SE of elpd\_loo is NA.  
##   
## Pareto k diagnostic values:  
## Count Pct. Min. n\_eff  
## (-Inf, 0.5] (good) 42 91.3% 1057   
## (0.5, 0.7] (ok) 2 4.3% 2088   
## (0.7, 1] (bad) 1 2.2% 1614   
## (1, Inf) (very bad) 1 2.2% 6   
## See help('pareto-k-diagnostic') for details.



loo\_compare <- loo\_compare(x = list(loo\_loglik\_separate=loo\_loglik\_separate, loo\_loglik\_pooled=loo\_loglik\_pooled))  
print(loo\_compare)

## elpd\_diff se\_diff  
## loo\_loglik\_separate 0.0 0.0   
## loo\_loglik\_pooled -0.5 1.3

The ELPD is the theoretical expected log pointwise predictive density for a new dataset (Eq 1 in VGG2017), which can be estimated, e.g., using cross-validation. elpd\_loo is the Bayesian LOO estimate of the expected log pointwise predictive density (Eq 4 in VGG2017) and is a sum of N individual pointwise log predictive densities.

As quick rule: If elpd difference (elpd\_diff in loo package) is less than 4, the difference is small (Sivula, Magnusson and Vehtari, 2020). If elpd difference (elpd\_diff in loo package) is larger than 4, then compare that difference to standard error of elpd\_diff (provided e.g. by loo package) (Sivula, Magnusson and Vehtari, 2020).

p\_loo (effective number of parameters) p\_loo is the difference between elpd\_loo and the non-cross-validated log posterior predictive density. It describes how much more difficult it is to predict future data than the observed data. Asymptotically under certain regularity conditions, p\_loo can be interpreted as the effective number of parameters. In well behaving cases p\_loo < N and p\_loo < p, where p is the total number of parameters in the model. p\_loo > N or p\_loo > p indicates that the model has very weak predictive capability and may indicate a severe model misspecification. See below for more on interpreting p\_loo when there are warnings about high Pareto k diagnostic values

p\_loo is called the effective number of parameters and can be computed as the difference between elpd\_loo and the non-cross-validated log posterior predictive density (Equations (4) and (3) in Vehtari, Gelman and Gabry (2017)). It is not needed for elpd\_loo, but has diagnostic value. It describes how much more difficult it is to predict future data than the observed data. Asymptotically under certain regularity conditions, p\_loo can be interpreted as the effective number of parameters. In well behaving cases p\_loo <N and p\_loo <p, where p is the total number of parameters in the model. p\_loo >N or p\_loo >p indicates that the model has very weak predictive capability.

The Pareto k estimate is a diagnostic for Pareto smoothed importance sampling (PSIS), which is used to compute components of elpd\_loo. In importance-sampling LOO (the full posterior distribution is used as the proposal distribution). The Pareto k diagnostic estimates how far an individual leave-one-out distribution is from the full distribution. If leaving out an observation changes the posterior too much then importance sampling is not able to give reliable estimate. If k<0.5, then the corresponding component of elpd\_loo is estimated with high accuracy. If 0.5<k<0.7 the accuracy is lower, but still ok. If k>0.7, then importance sampling is not able to provide useful estimate for that component/observation. Pareto k is also useful as a measure of influence of an observation. Highly influential observations have high k values. Very high k values often indicate model misspecification, outliers or mistakes in data processing.

Interpreting p\_loo when Pareto k is large If k > 0.7 then we can also look at the p\_loo estimate for some additional information about the problem:

If p\_loo << p (the total number of parameters in the model), then the model is likely to be misspecified. Posterior predictive checks (PPCs) are then likely to also detect the problem. Try using an overdispersed model, or add more structural information (nonlinearity, mixture model, etc.).

If p\_loo < p and the number of parameters p is relatively large compared to the number of observations (e.g., p>N/5), it is likely that the model is so flexible or the population prior so weak that it’s difficult to predict the left out observation (even for the true model). This happens, for example, in the simulated 8 schools (in VGG2017), random effect models with a few observations per random effect, and Gaussian processes and spatial models with short correlation lengths.

If p\_loo > p, then the model is likely to be badly misspecified. If the number of parameters p<<N, then PPCs are also likely to detect the problem. See the case study at <https://avehtari.github.io/modelselection/roaches.html> for an example. If p is relatively large compared to the number of observations, say p>N/5 (more accurately we should count number of observations influencing each parameter as in hierarchical models some groups may have few observations and other groups many), it is possible that PPCs won’t detect the problem

Online documentations: FAQ: <https://mc-stan.org/loo/articles/online-only/faq.html#elpd_interpretation> Glossaries: <https://mc-stan.org/loo/reference/loo-glossary.html>

# Discussion

## Existing issues

## Potential improvements

# Conclusion

# Reflection

# References

Sivula, Tuomas, Måns Magnusson, Asael Alonzo Matamoros, and Aki Vehtari. 2020. “Uncertainty in Bayesian Leave-One-Out Cross-Validation Based Model Comparison.” <https://doi.org/10.48550/ARXIV.2008.10296>.