BDA Project: Malicious and Benign Website URL detections

Nguyen Xuan Binh, Duong Le

January 2023

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))
```

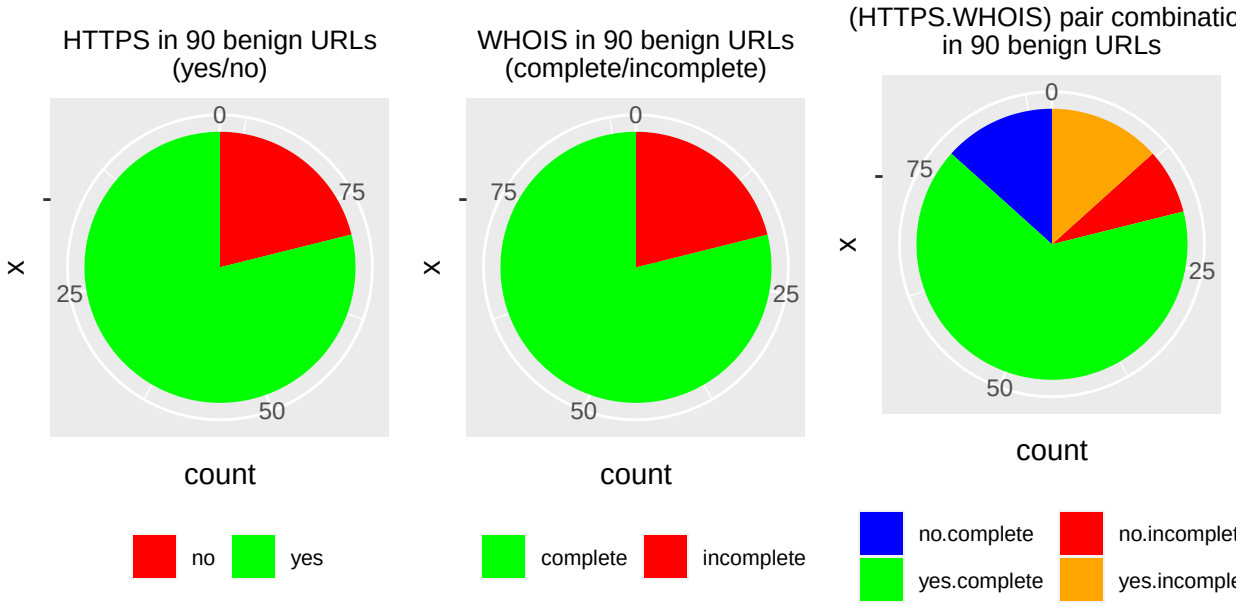
```
## Number of training data: 62
```

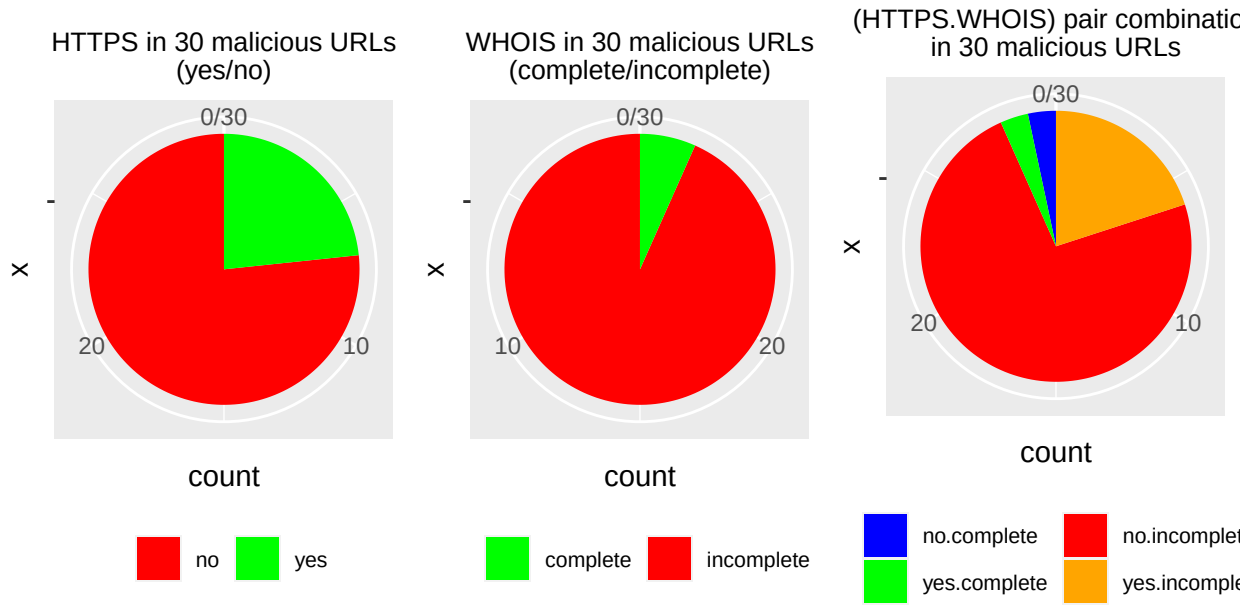
```
cat("\nNumber of testing data:",nrow(test_websites_top_3))
```

```
##
## Number of testing data: 131
```

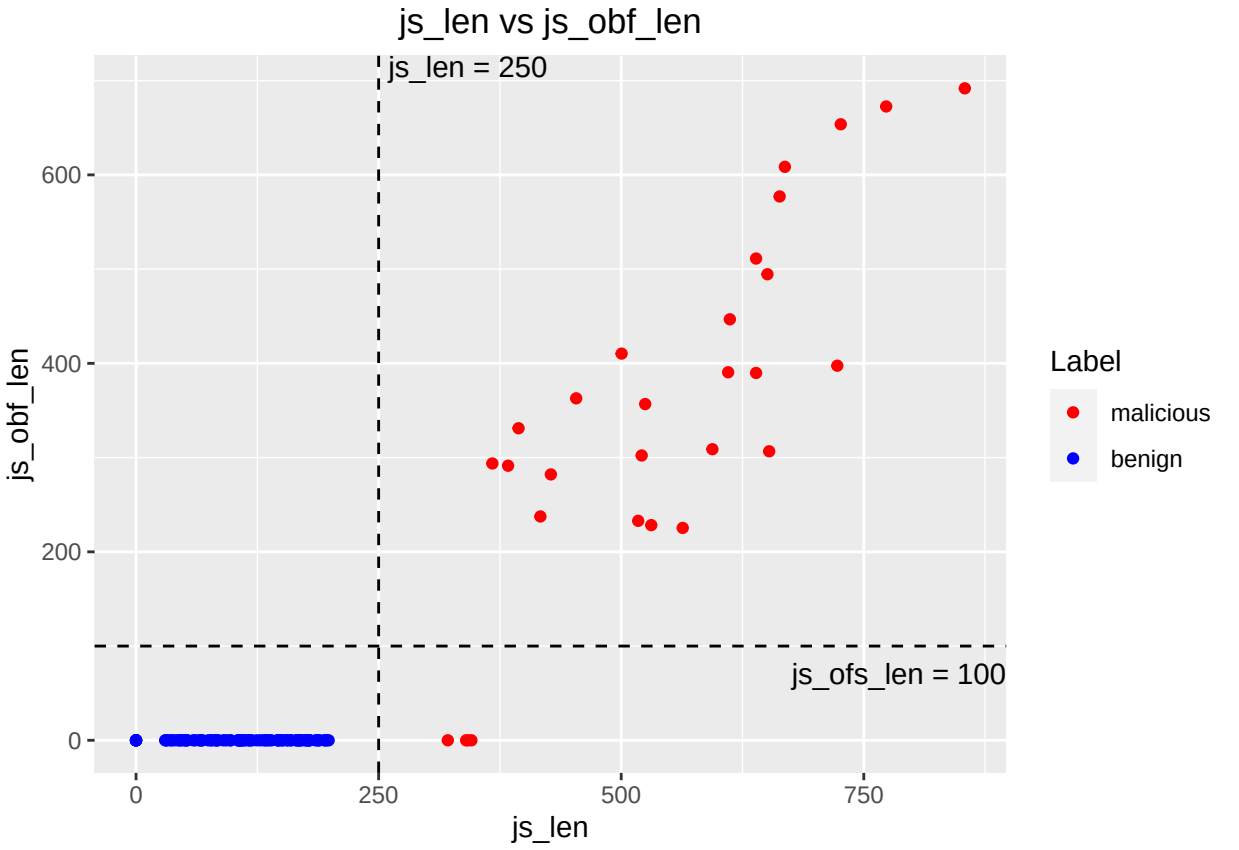
```
head(train_websites_top_3)
```

```
##   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
```


HTTPS in 90 benign URLs
(yes/no)
0
75
25
50
x
count
no
yes
WHOIS in 90 benign URLs
(complete/incomplete)
0
75
25
50
x
count
complete
incomplete
(HTTPS.WHOIS) pair combinatic
in 90 benign URLs
0
75
25
50
x
count
no.complete
yes.complete
no.incomplet
yes.incomple

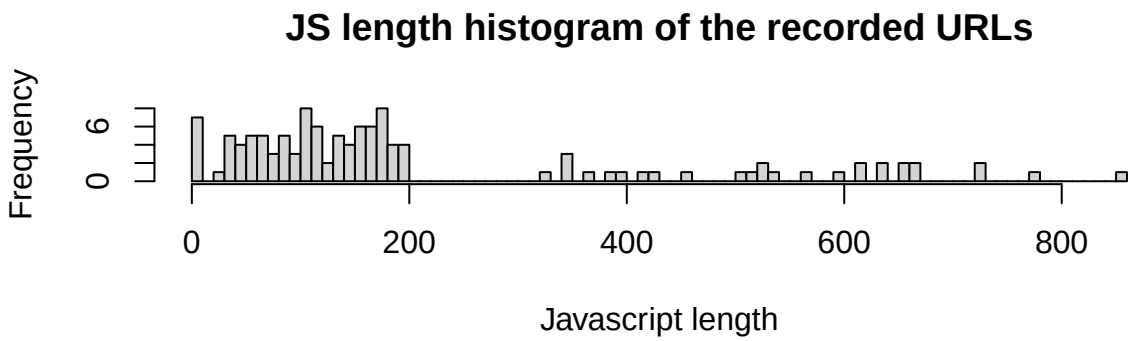


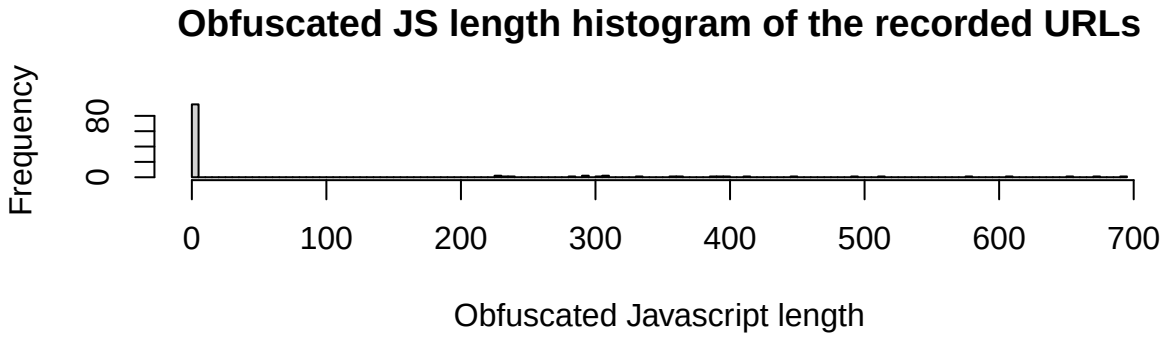

```
ggplot(data = train_websites, aes(x = js_len, y = js_obf_len, color = label)) +
  geom_point() +
  scale_color_manual(values = c("red", "blue"),
                     labels = c("malicious", "benign"),
                     guide = guide_legend(title = "Label")) +
  ggtitle("js_len vs js_obf_len") +
  xlab("js_len") +
  ylab("js_obf_len") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_vline(xintercept = 250, linetype = "dashed", color = "black") +
  geom_hline(yintercept = 100, linetype = "dashed", color = "black") +
  annotate("text", x = 260, y = Inf, label = "js_len = 250", hjust = 0, vjust = 1) +
  annotate("text", x = Inf, y = 60, label = "js_ofs_len = 100", hjust = 1, vjust = 0)
```



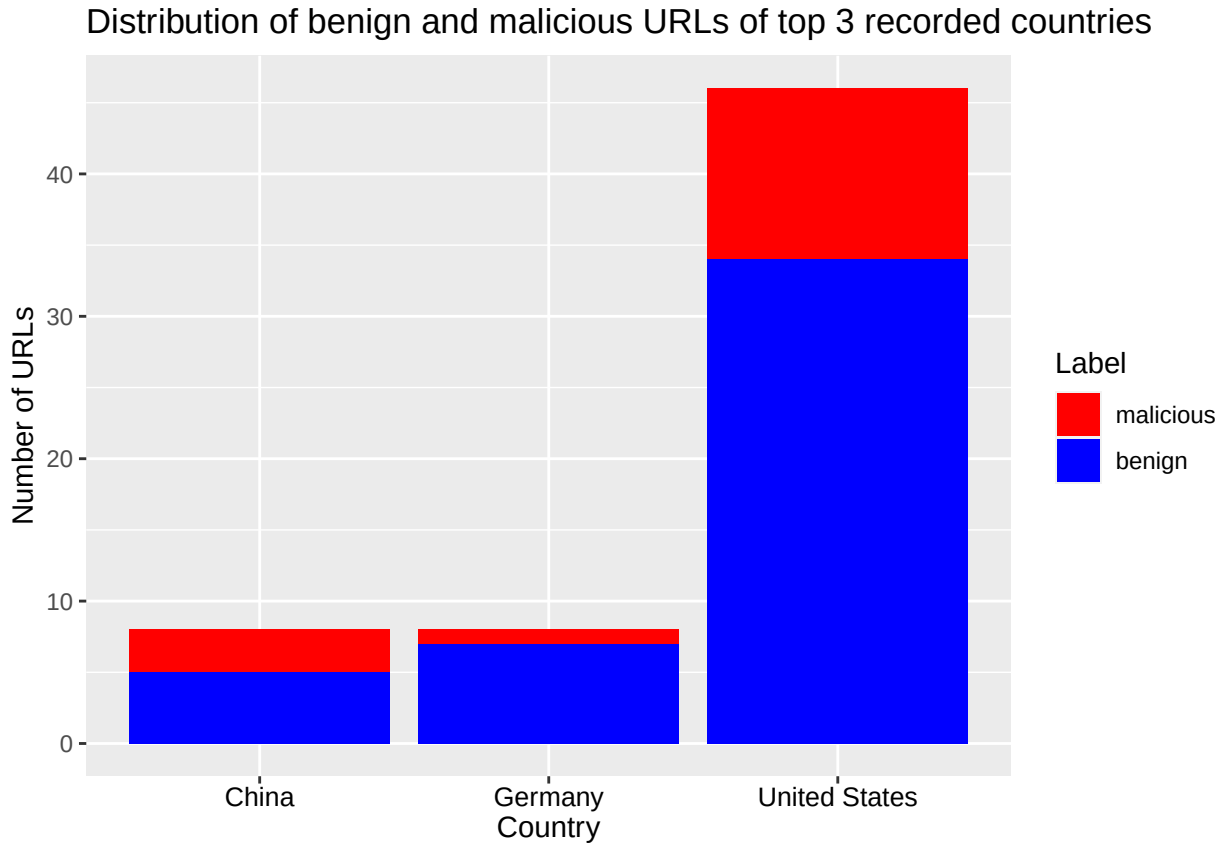

```
  #guides(color = guide_legend(title = "Label"))
```

```
# Set up the plotting grid
par(mfrow = c(2,1))

hist(train_websites$js_len, main = "JS length histogram of the recorded URLs", xlab = "Javascript lengt
hist(train_websites$js_obf_len, main ="Obfuscated JS length histogram of the recorded URLs", xlab = "Ob
```






```r
# Group the dataframe by geo_loc and count the number of rows for each country
train_websites_count <- train_websites %>%
  group_by(geo_loc) %>%
  summarize(count = n() ) %>%
  top_n(3, count) %>%
  slice_tail(n=3)

# Count the number of benign and malicious URLs for each country
train_websites_count_label <- train_websites %>%
  filter(geo_loc %in% train_websites_count$geo_loc) %>%
  group_by(geo_loc, label) %>%
  summarize(count = n())

# Plot the bar chart
ggplot(train_websites_count_label, aes(x = geo_loc, y = count, fill = label)) +
  geom_bar(stat = "identity", position = "stack") +
  scale_fill_manual(values = c("red", "blue"),
                    labels = c("malicious","benign")) +
  xlab("Country") +
  ylab("Number of URLs") +
  ggtitle("Distribution of benign and malicious URLs of top 3 recorded countries") +
  guides(fill = guide_legend(title = "Label")) +
  theme(axis.text.x = element_text(angle = 0, hjust = 0.5, vjust = 0.5, size = 10,
                                   margin = margin(r = -20, unit = "pt"),
                                   #family = "serif",
                                   lineheight = 0.9, color = "black"))
```



Loading the data:

Composing data for the Stan model

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 regress
```

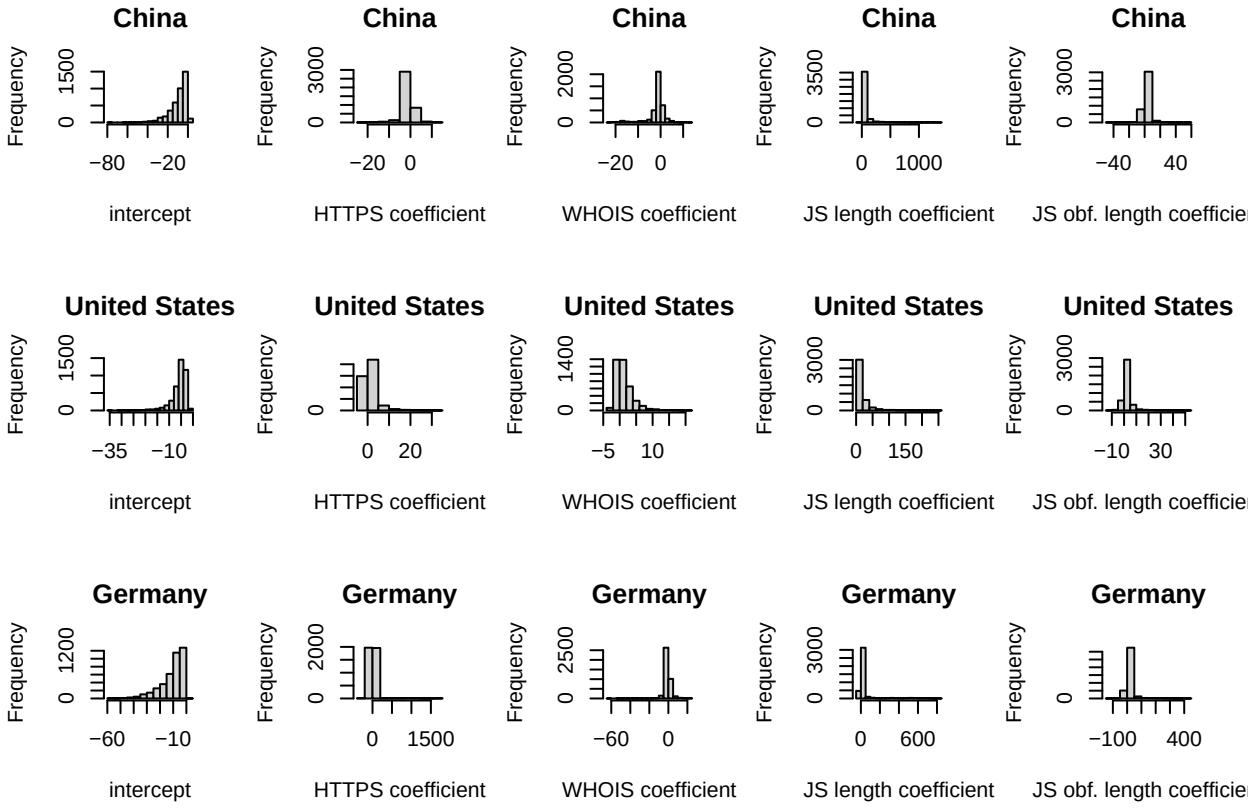
```
    for (k in 1:K){
      for (i in 1:N_list[k]){
        label_list[k, i] ~ bernoulli(inv_logit(intercept[k] + https_coeff[k] * https_list[k, i] + whois
      }
    }
}

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,
      }
    }
    // 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
      }
    }
    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]
        }
      }
    }
}

"
```

The sampling running options

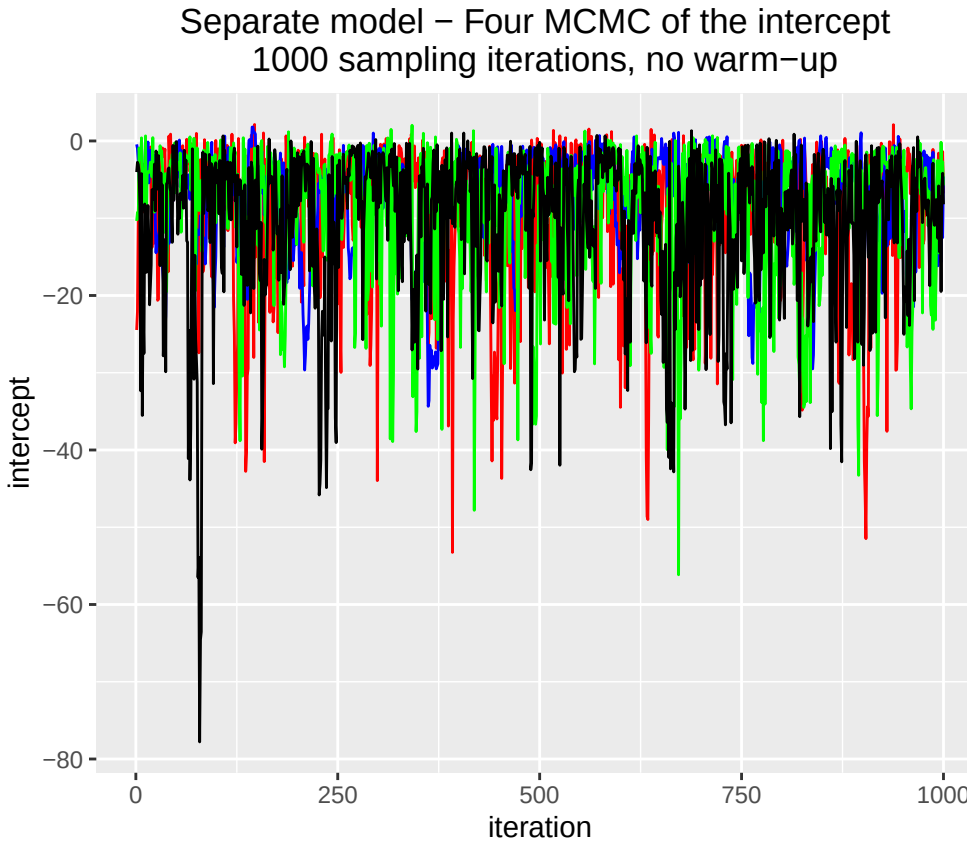
```
  separate_sampling <- model_separate$sample(data = stan_data, chains=4, iter_warmup = 1000, iter_sampli
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 finished in 12.6 seconds.
## Chain 2 finished in 31.4 seconds.
## Chain 3 finished in 28.7 seconds.
## Chain 4 finished in 15.3 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 22.0 seconds.
## Total execution time: 88.5 seconds.

## Warning: 1643 of 4000 (41.0%) transitions hit the maximum treedepth limit of 10.
## See https://mc-stan.org/misc/warnings for details.
```

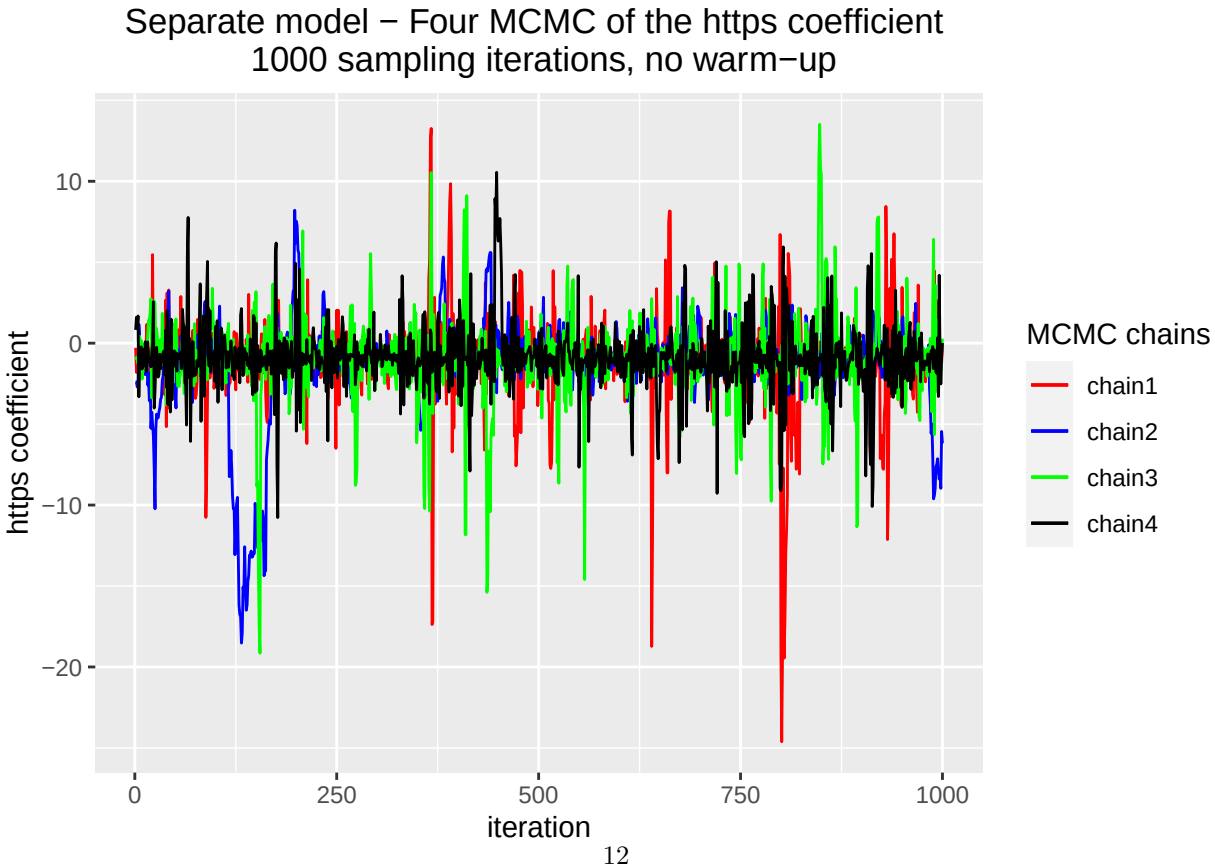
```r
# Set up the plotting grid
par(mfrow = c(3,5))

row_labels <- c("intercept", "HTTPS coefficient", "WHOIS coefficient","JS length coefficient","JS obf. l
row_names <- c("intercept","https_coeff","whois_coeff","js_len_coeff","js_obf_len_coeff")

#separate_sampling$summary()
# Loop through the countries
for (j in 1:3){
  for(i in 1:5){
      # Create the subplot
      hist(separate_sampling$draws(paste(row_names[i],"[",j,"]", sep="")),main = countries[j], xlab=row_
      # Add the country name to the top of the column
      mtext(countries[j], side = 3, line = 0.2, outer = TRUE)
    }
}
```

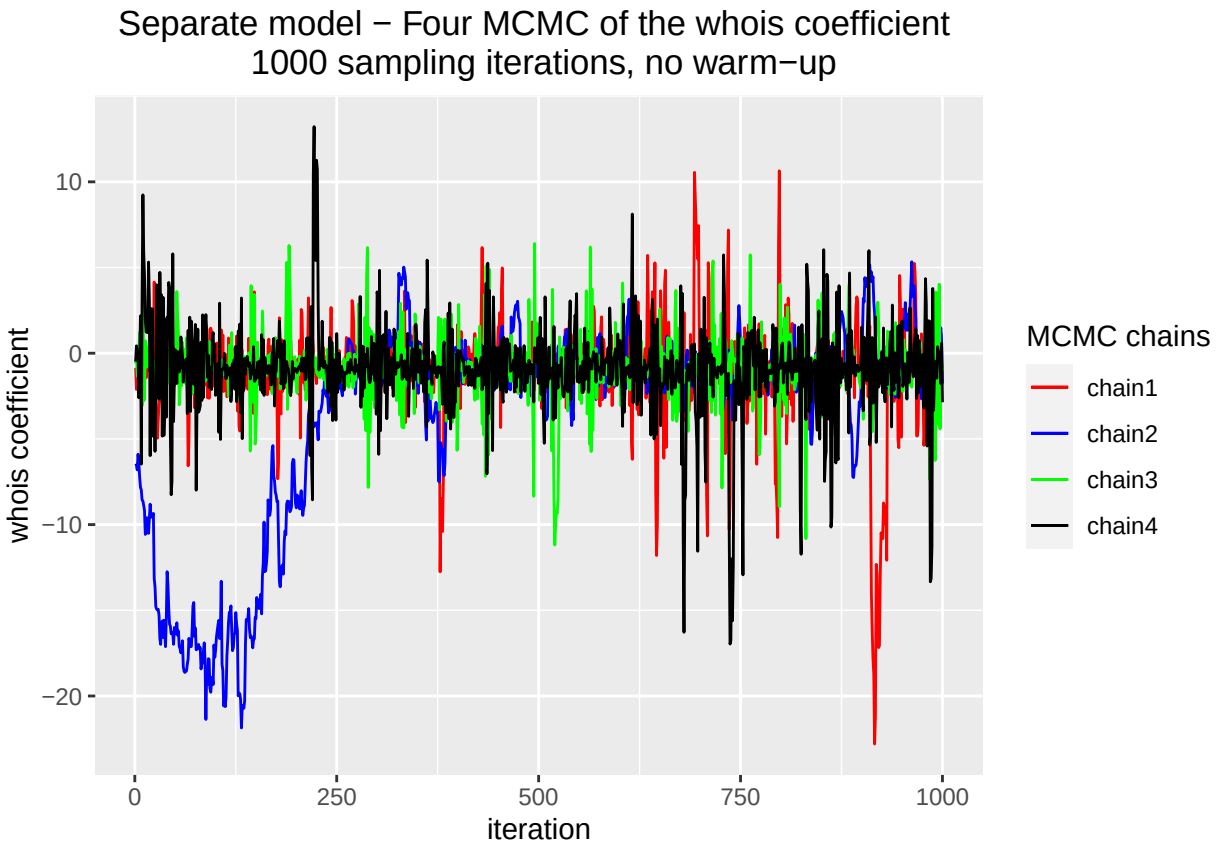


Convergence diagnostics



MCMC convergence chains visualization





HMC specific convergence diagnostics

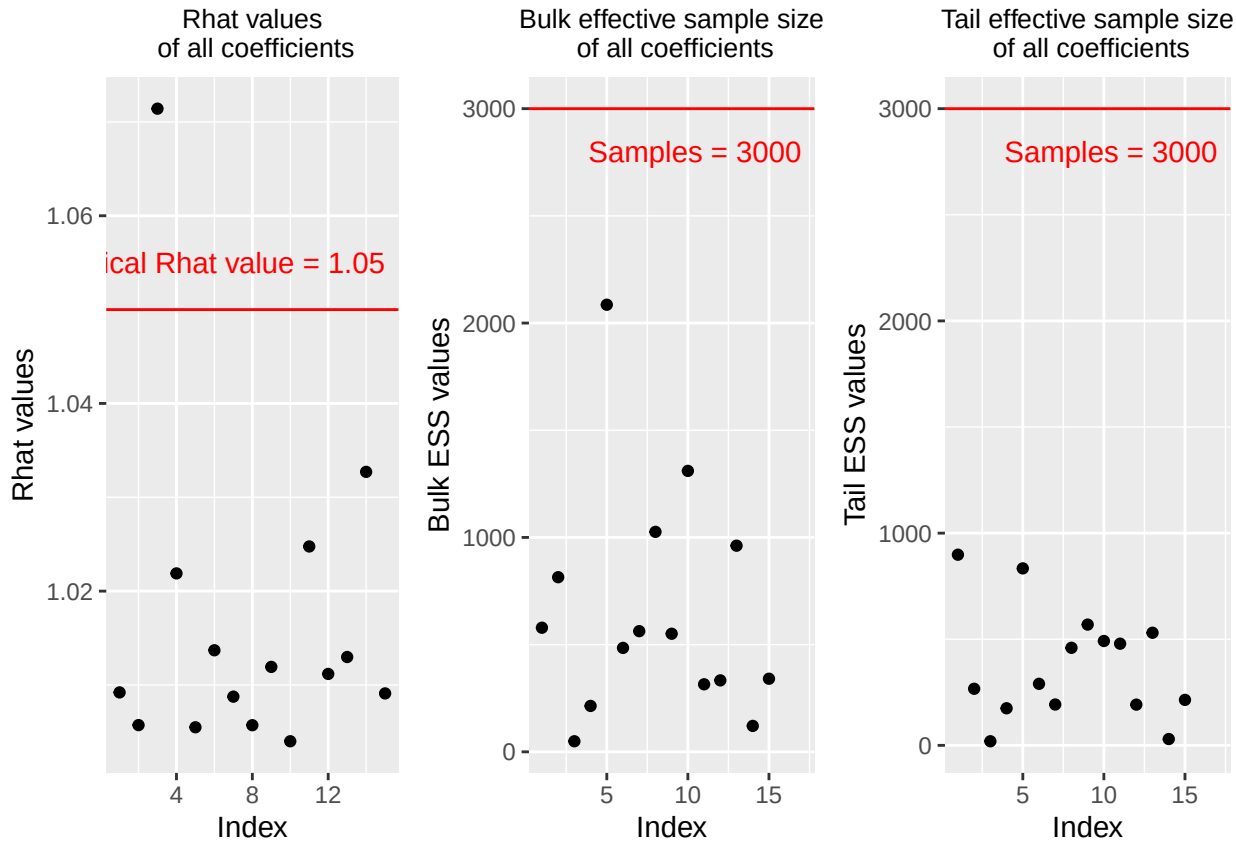
```
separate_sampling$diagnostic_summary()
```

```
## Warning: 1643 of 4000 (41.0%) transitions hit the maximum treedepth limit of 10.
## See https://mc-stan.org/misc/warnings for details.

## $num_divergent
## [1] 0 0 0 0
##
## $num_max_treedepth
## [1]   11 1000  631    1
##
## $ebfmi
## [1] 0.4549715 0.5477585 0.4022651 0.5487693
```

$\hat{R}$-values and effective sample size

```
summaryDiagnostics = data.frame()
for (i in 1:K){
  summaryDiagnosticsCountry <- separate_sampling$summary(c(paste("intercept[",i,"]", sep=""), paste("ht
 summaryDiagnostics <- rbind(summaryDiagnostics, summaryDiagnosticsCountry)

}
summaryDiagnostics$variable <- 1:15
```

```r
summaryDiagnostics <- summaryDiagnostics %>% rename(index = variable)

p1 <- ggplot(summaryDiagnostics, aes(x = index, y = rhat)) +
  geom_point() +
  geom_hline(yintercept = 1.05, color = "red") +
  annotate("text", x = 15, y = 1.055, label = "critical Rhat value = 1.05",
           hjust = 1, color = "red") +
  xlab("Index") +
  ylab("Rhat values") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.title = element_text(size = 10)) +
  ggtitle("Rhat values\n of all coefficients")

p2 <- ggplot(summaryDiagnostics, aes(x = index, y = ess_bulk)) +
  geom_point() +
  geom_hline(yintercept = 3000, color = "red") +
  annotate("text", x = 17, y = 2800, label = "Samples = 3000",
           hjust = 1, color = "red") +
  xlab("Index") +
  ylab("Bulk ESS values") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.title = element_text(size = 10)) +
  ggtitle("Bulk effective sample size\n of all coefficients")

p3 <- ggplot(summaryDiagnostics, aes(x = index, y = ess_tail)) +
  geom_point() +
  geom_hline(yintercept = 3000, color = "red") +
  annotate("text", x = 17, y = 2800, label = "Samples = 3000",
           hjust = 1, color = "red") +
  xlab("Index") +
  ylab("Tail ESS values") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.title = element_text(size = 10)) +
  ggtitle("Tail effective sample size\n of all coefficients")

grid.arrange(p1, p2, p3, ncol = 3)
```



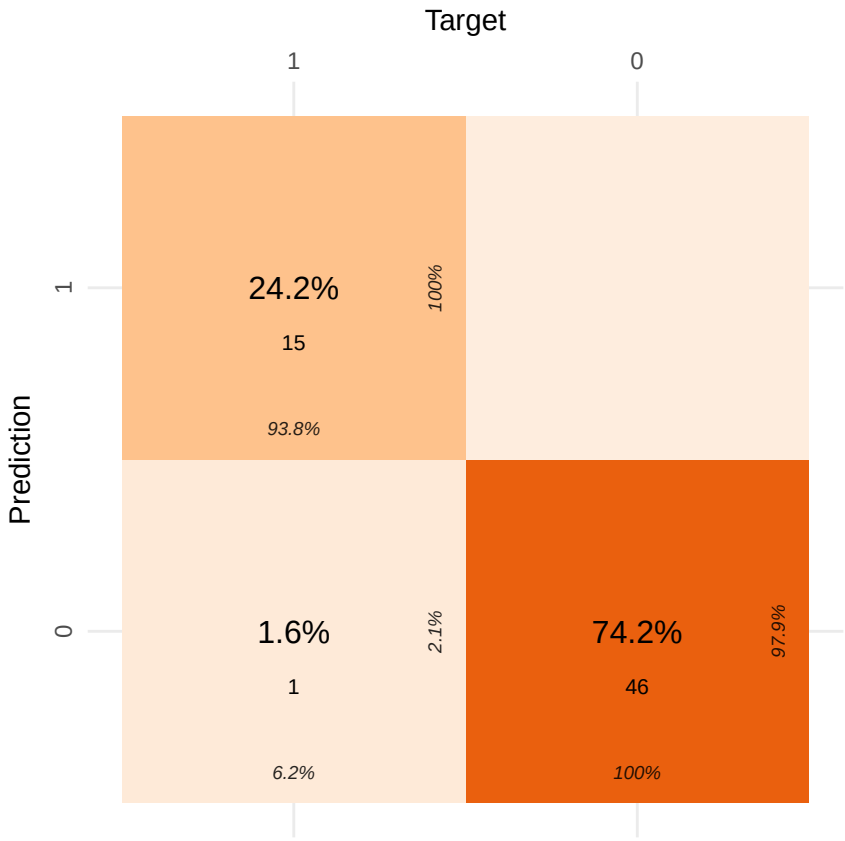
Posterior predictive checks

```
metricsName <- c("Accuracy", "Precision", "Recall", "F1")
metricsSummary <- data.frame(Metrics=metricsName)
metricsSummary$Accuracy <- NULL
metricsSummary$Precision <- NULL
metricsSummary$Recall <- NULL
metricsSummary$F1 <- NULL
#print(metricsSummary)
sumTP <- 0
sumTN <- 0
sumFP <- 0
sumFN <- 0
listTrue <- c()
listPred <- c()
for (k in 1:K){
  predicted_train = c()
  for (i in 1:N_list[[k]]){

    draws <- separate_sampling$draws(paste("label_train_pred[",k,",",i,"]", sep=""), format = "matrix")
    #print(as.vector(draws[1, 1]))
    predicted_train <- c(predicted_train, as.vector(draws[1, ]))
  }
  #draws <- separate_sampling$draws(paste("label_train_pred[",i,"]"), format = "matrix")
  true_train = label_list[[k]][1:N_list[[k]]]
  confusion_matrix <- table(predicted_train, true_train)
  #print(confusion_matrix)
```

```
  listTrue <- c(listTrue, true_train)
  listPred <- c(listPred, predicted_train)
  #print(listTrue)
  #print(listPred)
  TP <- confusion_matrix[2,2]
  TN <- confusion_matrix[1,1]
  FP <-confusion_matrix[1,2]
  FN <-confusion_matrix[2,1]
  #cat(TP,TN,FP,FN)
  sumTP <- sumTP + TP
  sumTN <- sumTN + TN
  sumFP <- sumFP + FP
  sumFN <- sumFN + FN
  accuracy <- (TP+TN)/(TP+FP+FN+TN)
  precision <- TP/(TP+FP)
  recall <- TP/(TP+FN)
  f1 <-  2*(precision*recall)/(precision+recall)
  metricsSummary[,countries[k]] <- c(accuracy, precision,recall,f1)
}
accuracy <- (sumTP+sumTN)/(sumTP+sumFP+sumFN+sumTN)
prevalence <- (sumTP+sumFN)/(sumTP+sumFP+sumFN+sumTN)
sensitivity <- sumTP/(sumTP+sumFN)
specificity <- sumFN/(sumFN+sumFP)
precision <- sumTP/(sumTP+sumFP)
recall <- sumTP/(sumTP+sumFN)
f1 <-  2*(precision*recall)/(precision+recall)
metricsSummary[,"All countries"] <- c(accuracy, precision,recall,f1)
metricsSummary
```

```
##     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")
```



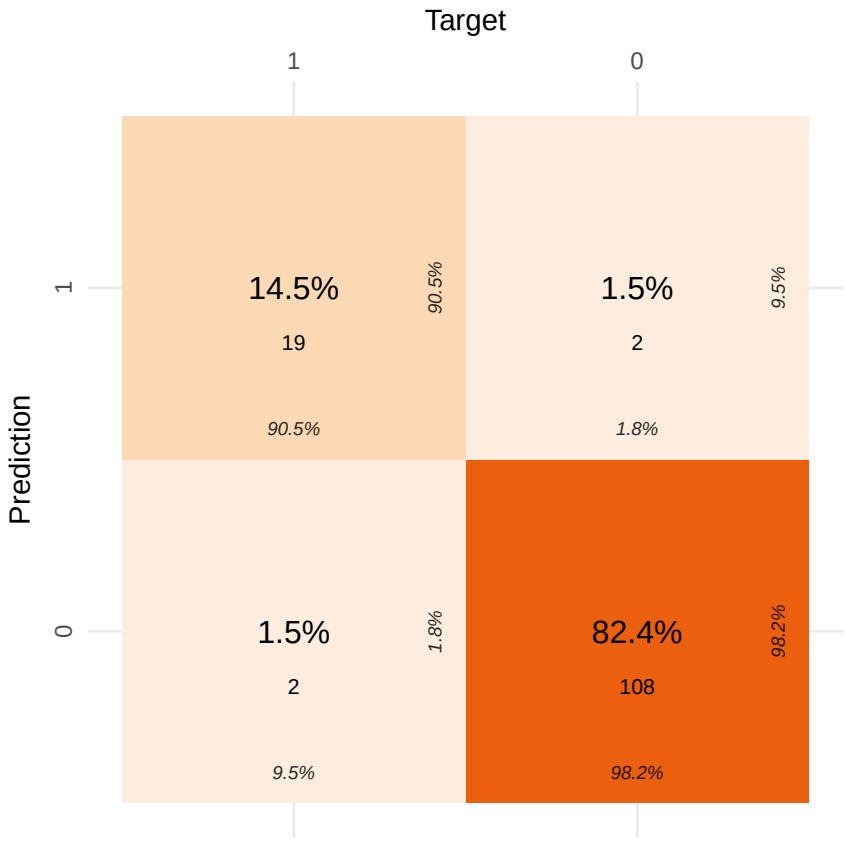
Predictive performance assessment

```
metricsName <- c("Accuracy", "Precision", "Recall", "F1")
metricsSummary <- data.frame(Metrics=metricsName)
metricsSummary$Accuracy <- NULL
metricsSummary$Precision <- NULL
metricsSummary$Recall <- NULL
metricsSummary$F1 <- NULL
#print(metricsSummary)
sumTP <- 0
sumTN <- 0
sumFP <- 0
sumFN <- 0
listTrue <- c()
listPred <- c()
for (k in 1:K){
  predicted_test = c()
  for (i in 1:M_list[[k]]){
    draws <- separate_sampling$draws(paste("label_test_pred[",k,",",i,"]", sep=""), format = "matrix")
    predicted_test <- c(predicted_test, as.vector(draws[1, ]))
  }
  true_test = label_test_list[[k]][1:M_list[[k]]]
  confusion_matrix <- table(predicted_test, true_test)
  listTrue <- c(listTrue, true_test)
```

```
  listPred <- c(listPred, predicted_test)

  TP <- confusion_matrix[2,2]
  TN <- confusion_matrix[1,1]
  FP <-confusion_matrix[1,2]
  FN <-confusion_matrix[2,1]

  sumTP <- sumTP + TP
  sumTN <- sumTN + TN
  sumFP <- sumFP + FP
  sumFN <- sumFN + FN
  accuracy <- (TP+TN)/(TP+FP+FN+TN)
  precision <- TP/(TP+FP)
  recall <- TP/(TP+FN)
  f1 <-  2*(precision*recall)/(precision+recall)
  metricsSummary[,countries[k]] <- c(accuracy, precision,recall,f1)
}
accuracy <- (sumTP+sumTN)/(sumTP+sumFP+sumFN+sumTN)
prevalence <- (sumTP+sumFN)/(sumTP+sumFP+sumFN+sumTN)
sensitivity <- sumTP/(sumTP+sumFN)
specificity <- sumFN/(sumFN+sumFP)
precision <- sumTP/(sumTP+sumFP)
recall <- sumTP/(sumTP+sumFN)
f1 <-  2*(precision*recall)/(precision+recall)
metricsSummary[,"All countries"] <- c(accuracy, precision,recall,f1)
metricsSummary
```

```
##     Metrics     China United States Germany All countries
## 1  Accuracy 0.9629630     0.9673913       1     0.9694656
## 2 Precision 0.5000000     0.9444444       1     0.9047619
## 3    Recall 1.0000000     0.8947368       1     0.9047619
## 4        F1 0.6666667     0.9189189       1     0.9047619
```

```
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")
```



Prior sensitivity analysis

The sampling running options

```
  separate_sampling_prior_sensitivity1 <- model_separate_prior_sensitivity1$sample(data = stan_data, cha
```

```
## Running MCMC with 4 sequential chains...
##
## Chain 1 finished in 19.2 seconds.
## Chain 2 finished in 11.9 seconds.
## Chain 3 finished in 20.6 seconds.
## Chain 4 finished in 5.6 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 14.3 seconds.
## Total execution time: 57.7 seconds.
```

```
## Warning: 12 of 4000 (0.0%) 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, cha
```

```
## Running MCMC with 4 sequential chains...
```

```
##
## Chain 1 finished in 18.9 seconds.
## Chain 2 finished in 25.6 seconds.
## Chain 3 finished in 10.6 seconds.
## Chain 4 finished in 7.7 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 15.7 seconds.
## Total execution time: 63.1 seconds.

## Warning: 802 of 4000 (20.0%) transitions hit the maximum treedepth limit of 10.
## See https://mc-stan.org/misc/warnings for details.
```

```
loo_loglike_separate0 <- separate_sampling$loo(variables="log_likelihood",r_eff=TRUE)
loo_loglike_separate1 <- separate_sampling_prior_sensitivity1$loo(variables="log_likelihood",r_eff=TRUE)
loo_loglike_separate2 <- separate_sampling_prior_sensitivity2$loo(variables="log_likelihood",r_eff=TRUE)

cat("The elpd value of the separate model 0 is\n")
```

```
## The elpd value of the separate model 0 is
```

```
print(loo_loglike_separate0$estimates[1][1])
```

```
## [1] -6.832074
```

```
cat("The elpd value of the separate model 1 is\n")
```

```
## The elpd value of the separate model 1 is
```

```
print(loo_loglike_separate1$estimates[1][1])
```

```
## [1] -6.475045
```

```
cat("The elpd value of the separate model 2 is\n")
```

```
## The elpd value of the separate model 2 is
```

```
print(loo_loglike_separate2$estimates[1][1])
```

```
## [1] -6.241456
```

```
loo_compare_separate <- loo_compare(x = list(loo_loglike_separate0=loo_loglike_separate0, loo_loglike_s
print(loo_compare_separate)
```

```
##                       elpd_diff se_diff
## loo_loglike_separate2  0.0       0.0
## loo_loglike_separate1 -0.2       0.3
## loo_loglike_separate0 -0.6       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 (**akiLoo?**).

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.

Model comparison

```
loo_separate <- separate_sampling$loo(variables="log_likelihood",r_eff=TRUE)
```

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
cat("The PSIS-LOO elpd value of the separate model is\n")
```

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

```
print(loo_separate$estimates[1][1])
```

```
## [1] -6.832074
```

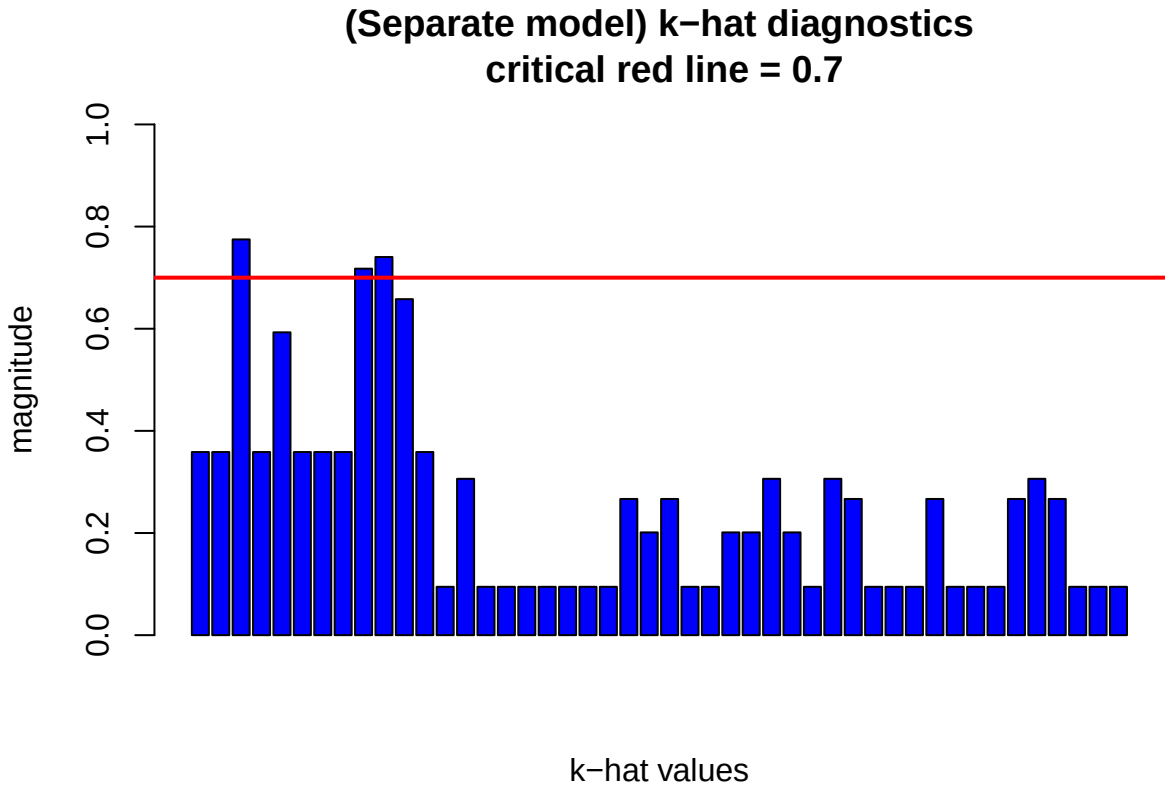
```r
pareto_k <- loo_separate$diagnostics$pareto_k
cat("\nThe k-hat values of the separate model is\n")
```

```
##
## The k-hat values of the separate model is
```

```r
print(loo_separate)
```

```
##
## Computed from 4000 by 46 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo     -6.8 4.9
## p_loo         2.9 2.4
## looic        13.7 9.7
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     41    89.1%   2069
##  (0.5, 0.7]   (ok)        2     4.3%   1257
##    (0.7, 1]   (bad)       3     6.5%   86
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```r
barplot(pareto_k, main = "(Separate model) k-hat diagnostics\n critical red line = 0.7", xlab = "k-hat
abline(h=0.7, col="red", lwd=2)
```



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